제 18 권 제 1 호
Vol. 18 No. 1

2016

제 18회 한국 소프트웨어공학 학술대회 논문집

Proceedings of the 18th Korea Conference on Software Engineering (KCSE 2016)

- **일시: 2016년 1월 27일(수) ~ 1월 29일(금)**
- **장소: 강원도 평창 한화리조트(휘닉스파크점)**

주최: 한국정보과학회, 한국정보처리학회
주관: 한국정보과학회 소프트웨어공학 소사이어티
한국정보처리학회 소프트웨어공학 연구회
한국전자통신연구원
후원: ㈜솔루션링크, ㈜코스콤, ㈜모아소프트, ㈜비트컴퓨터, 소프트웨어공학엑스퍼트그룹㈜, ㈜엔에스이, 한국산업기술시험원, 무인자율 및 적응형 SW 연구단, ㈜다한테크, SW 상시모니터링기술연구단, 시스트란 인터내셔널, ㈜티큐엠에스, ITRC 고품질융합소프트웨어연구센터, 슈어소프트테크㈜, ITRC 소프트웨어안전성보증연구센터, STA 테스팅컨설팅㈜, ㈜이에스지, 정보통신산업진흥원 소프트웨어공학센터, ㈜테스트마이다스, TTA 소프트웨어시험인증연구소

초대의 글

소프트웨어 공학인의 축제인 제18회 한국 소프트웨어 공학 학술대회(KCSE 2016)에 참가하러 오신 여러분을 환경합니다.

기업, 연구소 및 학계에서 활동하고 계신 소프트웨어공학 관련 전문가들의 모임인 한국정보과학회의 소프트웨어공학 소사이어티와 한국정보처리학회의 소프트웨어공학연구회에서는 여러 산업분야에서 소프트웨어 공학의 중요성이 날로 부각되고 소프트웨어 공학 기술이 국가 발전에 중요한 부분으로 인식되는 시점에서 소프트웨어 공학 기술의 발전과 적용을 확대하고자 산, 학, 연이 협력하여 한국 소프트웨어 공학 학술대회를 개최하게 되었습니다.

이번 학술대회에서는 특히 "**미래 IT사회로의 혁신을 선도하는 소프트웨어 공학**"을 주제로 산, 학, 연 각계에서 제출한 논문 90여편이 발표됩니다. 또한 탐색 기반의 소프트웨어 공학, 차량 전장 SW의 품질 향상을 위한 프로세스, 산업체 SW 제품 라인 개발을 포함한 총 3개의 유익한 튜토리얼이 준비되어 있습니다.

아울러 소프트웨어 공학 발전을 위해 애쓰고 계신 저명인사 3분의 기조연설도 있습니다. 본 학술대회가 소프트웨어 공학의 학문적 발전과 소프트웨어 산업기술 발전의 큰 장이 되는 만큼, 활발한 학술 및 기술 교류는 물론이거니와 격의 없는 토론이 진행될 수 있도록 여러분의 적극적인 참여를 부탁 드립니다. 이번 행사를 위해 수고해 주신 준비 위원들과 후원해 주신 기관 여러분께 진심으로 감사를 드립니다.

한국정보과학회 소프트웨어공학소사이어티 회장 권기현
한국정보처리학회 소프트웨어공학연구회 운영위원장 유철중

학술대회 준비 위원회

공동 대회장: 권기현 교수(경기대), 유철중 교수(전북대)

조직위원장: 백종문 교수(KAIST)

조직위원: 고인영 교수(KAIST), 김영철 교수(홍익대), 박수진 교수(서강대),
염근혁 교수(부산대), 유준범 교수(건국대), 이정원 교수(아주대)

학술위원장: 이병정 교수(서울시립대)

학술위원: 고인영 교수(KAIST), 김문주 교수(KAIST), 김순태 교수(전북대), 박수진 교수(서강대),
서주영 교수(아주대), 오재원 교수(가톨릭대), 유준범 교수(건국대),
윤일철 교수(한국뉴욕주립대), 윤회진 교수(협성대), 이관우 교수(한성대),
이우진 교수(경북대), 이은서 교수(안동대), 이은주 교수(경북대), 이정원 교수(아주대),
이찬근 교수(중앙대), 정우성 교수(충북대), 정인상 교수(한성대), 조은숙 교수(서일대),
진영택 교수(한밭대), 채흥석 교수(부산대), 최승훈 교수(덕성여대), 최윤자 교수(경북대),
홍장의 교수(충북대)

문의사항 연락처

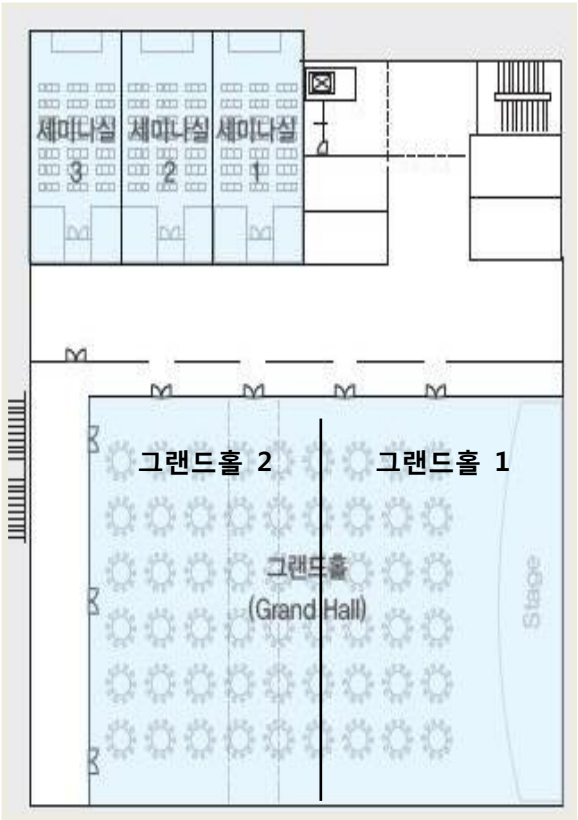
학술대회 홈페이지 : http://www.sigsoft.or.kr/KCSE2016/
조 직 : 백종문 교수 (jbaik@kaist.ac.kr, 042-350-3556)
학 술 : 이병정 교수 (bjlee@uos.ac.kr, 02-6490-2451)

행사장 안내

• 휘닉스파크 내 한화리조트 위치(레드동)

(개최장소 문의처: 02-729-3840)

• 행사장 및 식당 위치



1 F

B 1 F

KCSE 2016 프로그램

<table>
<tr><th colspan="5">1 월 27 일 (수)</th></tr>
<tr><th>시간</th><th colspan="3">행 사 내 용</th><th></th></tr>
<tr><td>10:00–12:00</td><td colspan="3">KCSE 2016 준비</td><td rowspan="2"></td></tr>
<tr><td>12:00–13:30</td><td colspan="3">KCSE 2016 등록</td></tr>
<tr><td></td><td>튜토리얼 T1: SW 공학과 최적화 알고리즘
좌장 이미연 교수(아주대) 장소: 그랜드홀 2</td><td>튜토리얼 T2: SW Process
좌장 박수진 교수(서강대) 장소: 세미나실 1</td><td>튜토리얼 T3: SW Product Line
좌장 이정원 교수(아주대) 장소: 세미나실 2</td><td></td></tr>
<tr><td>13:30-15:30
(120 분)</td><td>제목: 검색 기반 소프트웨어 공학
유신 교수(KAIST)</td><td>제목: 자동차 SW 의 안전성을 위한 프로세스 모델: Automotive SPICE 와 CMMI
한혁수 교수(상명대)</td><td>제목: 임베디드 SW product line 개발 및 진화
이근 박사(삼성전자)</td><td rowspan="5">무인자율 및 적응형 SW 연구단 워크샵

장소: 세미나실 3 (15:00-17:00)</td></tr>
<tr><td>15:30-15:40</td><td colspan="3">휴식</td></tr>
<tr><td></td><td colspan="3">개회식 사회: 백종문 조직위원장(KAIST)
장소: 그랜드홀 2</td></tr>
<tr><td>15:40-16:00
(20 분)</td><td colspan="3">개회사: 권기현 회장(한국정보과학회 소프트웨어공학소사이어티), 유철중 위원장(한국정보처리학회 소프트웨어공학연구회)
KCSE 2016 프로그램 소개: 이병정 학술위원장(서울시립대)</td></tr>
<tr><td></td><td colspan="3">기조연설 I 사회: 백종문 조직위원장(KAIST)
장소: 그랜드홀 2</td></tr>
<tr><td>16:00-17:00</td><td colspan="3">김진형 소장(소프트웨어정책연구소)</td><td></td></tr>
<tr><td></td><td colspan="4">기조연설 II 사회: 이병정 학술위원장(서울시립대)
장소: 그랜드홀 2</td></tr>
<tr><td>17:00-18:00</td><td colspan="4">심현택 소장(정보통신산업진흥원 소프트웨어공학센터)</td></tr>
<tr><td>18:00-19:00</td><td colspan="4">석 식</td></tr>
</table>

1 월 28 일 (목)

시 간	행 사 내 용				
07:00-09:00	**조식**				
	논문 발표 A				**고품질융합 소프트웨어 연구센터** 장소: 세미나실 6 (09:00-18:00)
	A1: SW 테스팅 1 좌장 최윤자 교수(경북대) 장소: 그랜드홀 2	**A2: SW 아키텍처 1** 좌장 고인영 교수(KAIST) 장소: 세미나실 1	**A3: 적응형 SW** 좌장 박수진 교수(서강대) 장소: 세미나실 2	**A4: 이슈 및 결함 관리** 좌장 권기현 교수(경기대) 장소: 세미나실 3	
09:00-11:30 (150 분)	UML 상태 다이어그램 기반 테스트 스크립트 자동 생성 도구 [학부논문] 이영우, 최현재, 채흥석(부산대) Pair-wise Component Compatibility Testing [단편논문] Anvika, 윤일철(뉴욕대) 안드로이드 애플리케이션 무반응 탐색과 회피 방안 김경민, 최은만(동국대) 다중 결함 환경에서의 SFL 성능 개선을 위한 테스트 케이스 군집화 및 가중치 부여 기법 [단편논문] 이재희, 김정호, 이은석(성균관대) Circus 명세를 통한 Go 코드 생성 신지훈, 최진영(고려대) 동시성을 포함한 액티비티 다이어그램 기반 테스트 시나리오 생성 기법 [우수논문] 백승찬, 최효린, 이병정(서울시립대), 이정원(아주대)	코드 중복 없이 안드로이드 컴포넌트에 예외 처리를 추가하는 클래스 설계 방법에 관한 연구 [우수학부논문] 김현순, 이승휘, 이화중, 최광훈(연세대), 창병모(숙명여대) 레거시 소프트웨어의 품질 개선을 위한 역방향 Tactic 분석 중심의 아키텍처 평가 [우수산업체논문] 류동국, 김재선, 김진태(엑스퍼트그룹) 논리적 UI 모델의 MVC 패턴으로의 매핑 김세화, 최기봉, 이승엽(한국외대) 소프트웨어의 계층 구조 파악을 위한 계층적 K-means 알고리즘 이선로, 이찬근(중앙대) 공중전투기동 내장형 훈련 S/W 아키텍처 설계 [산업체논문] 장영찬, 지철규, 오지현, 김천영, 홍영석(국방과학연구소) 자동차 원격 제어의 신뢰성을 높이기 위한 시스템 구조 및 동작 절차 제안 이윤성, 권영채, 이성주, 이율희, 이석원(아주대)	Using Problem Frames and Goal Modelling techniques to determine Variability in Smart Grid RTP systems Meetushi, 이석원(아주대) 모델 기반 소프트웨어공학에 입각한 요구사항 추적성 메타모델 정의법 개관 [단편논문] 변성훈, 이석원(아주대) 환경정보를 고려한 자가적응형 시스템을 위한 동적 의사결정 기술 [우수논문] 김미수, 정호현, 이은석(성균관대) 모바일 어플리케이션 GUI 사용성 저해요소 검출 도구 [단편논문] 마경욱, 박수진(서강대) 시간-주파수 논리를 활용한 클래식 형식 분석 [단편논문] 박수정, 권혁주, 권기현(경기대) 레거시(PL/SQL) 코드에서 변경 영향을 검출하는 오염 분석기 설계 방법: 요약 해석을 기반으로 [산업체논문] 김용기(삼성전자)	Embedded S/W 개발에서 Issue 해결의 어려움에 대한 고찰 임순일, 정병민, 민상윤(KAIST) 가전제품 사업에서의 소프트웨어 버그리포트 재현가능성 향상에 관한 연구 이창훈, 이미연, 민상윤(KAIST) 메모리 영역 분할을 통한 메모리 갱신 정보 기반 결함 후보 축소 기법 [최우수논문] 김관효, 최기용, 이정원(아주대) IR 기반 결함위치추적을 위한 사용정보 영향력 분석 안준, 염창선, 김정호, 이은석(성균관대) 테스트케이스 재구성을 통한 결함위치식별 성능개선 [최우수논문] 김정호, 이은석(성균관대)	
11:30-13:00	**중식**				

	논문 발표 B			
	B1: SW 테스팅 2 좌장 정우성 교수(충북대) 장소: 그랜드홀 2	**B2: SW 아키텍처 2** 좌장 홍장의 교수(충북대) 장소: 세미나실 1	**B3: 노력 추정 및 도구** 좌장 이정원 교수(아주대) 장소: 세미나실 2	**B4: 보안 및 사물인터넷(IoT)** 좌장 이찬근 교수(중앙대) 장소: 세미나실 3
13:00-15:00 (120 분)	테스팅 트렌드와 산업계 동향 [후원업체] 박현우(㈜다한테크) 효과적인 웹 애플리케이션 테스팅을 위한 DOM 관련 동적 데이터 흐름 분석 방법 김지훈, 고인영(KAIST) 회귀 시험을 위한 통계적 분석 기반 테스트 항목 우선순위화 기법 [단편논문] 조영환, 이은석 (성균관대) 인수 (Acceptance) 테스트 자동화 실무 사례 [산업체논문] 신현일, 조성민, 최근호(S-Core) 입력 커버리지를 활용한 효율적인 동적 기호 실행 탐색 기법 [우수단편논문] 김윤호, 김문주(KAIST)	내장형 소프트웨어 안전성 강화를 위한 fault injection testing [후원업체] 송한규 실장(슈어소프트테크㈜) 계층적 구조를 고려한 소프트웨어 아키텍처 편차 측정 [우수단편논문] 김희주, 이기성, 김준석, 이찬근(중앙대) 산업용 로봇 소프트웨어에서 재사용성 확보를 위한 옵저버 패턴 적용 [단편논문] 하상범, 유철중(전북대) 상호 정보량을 이용한 소프트웨어 아키텍처 모듈-뷰 복원 평가 허민재, 이찬근(중앙대) 클러스터 앙상블을 이용한 소프트웨어 아키텍처 모듈-뷰 복원 조충기, 이찬근(중앙대)	IT 서비스 운영을 위한 투입공수 추정모델 적용사례 [후원업체] 이창환 팀장(KOSCOM) 소프트웨어 코드탐색 다이어그래밍 도구 [후원업체] 이선아, 정필수(KAIST) 온라인 게임 위치 기반 그룹 분류를 통한 봇 탐지 기법에 대한 연구 이세희, 김수아, 이지형(성균관대) Matlab 기반 차량제어시스템의 측정 데이터 분석 도구의 구현 [산업체논문] 이창호(현대오트론) 카페 좌석 관리 시스템 박평우, 임종찬, 노우리, 문은미, 이석원(아주대)	블록체인 기반 서비스의 트랜잭션 검증 효율성 개선 방안 [단편논문] 고동휘, 고덕윤, 이우승, 조수환, 최수진, 이일로, 박수용(서강대) 사물 인터넷 환경에서의 그룹 사용자를 위한 그룹 구성 정보 기반 서비스 추천 방법 [최우수논문] 이진서, 고인영(KAIST) 미션-크리티컬 인메모리 DBMS 의 결함탐지를 위한 결함주입 테스트 방법 및 도구 [산업체논문] 서강익, 마영철, 이재회, 이종정, 박준호(ALTIBASE) 국내 기업의 SW 개발보안 적용사례 분석 [단편논문] 박난경, 최진영, 임종인(고려대) 사물인터넷 환경에서 지리적 응집도를 고려한 동적 서비스 검색방법 [우수논문] 백경덕, 김민협, 고인영(KAIST)
15:00-15:20	**휴식**			

	논문 발표 C			
	C1: 정형 명세 좌장 김문주 교수(KAIST) 장소: 그랜드홀 2	**C2: SW 아키텍처 3** 좌장 이미연 교수(아주대) 장소: 세미나실 1	**C3: SW 프로세스** 좌장 고인영 교수(KAIST) 장소: 세미나실 2	**C4: SW 품질** 좌장 김순태 교수(전북대) 장소: 세미나실 3
15:20-17:00 (100 분)	분산 이동 실시간 시스템의 이동성 명세와 검증을 위한 시각화 도구 최영복, 박현주, 이문근(전북대) 위치 기반 서비스에서 이동 경로의 정형 명세 및 모니터링 권혁, 정선일, 권기현(경기대) 시스템 오브 시스템즈 목표 검증을 위한 시뮬레이션 사례 연구 [단편논문] 서동원, 신동환, 박지훈, 지은경, 배두환(KAIST) 에어컨 실외기 제품의 소프트웨어 재사용성 향상을 위한 피처 기반 제품라인공학 기술 적용 연구 [산업체논문] 권혁상(㈜삼성전자), 강성원, 한영훈(KAIST)	고신뢰성 소프트웨어공학도구 솔루션 SILKROAD [후원업체] 백가현 연구원(㈜엔에스이) A Software Development Framework for Industrial Computer Vision Systems Mesfin Abebe, Cheol-Jung Yoo(전북대) 구문적 노이즈 제거를 이용한 소프트웨어 아키텍처 복원 자동화 프레임워크 [단편논문] 이기성, 이찬근(중앙대) 실시간 연동을 위한 LVC 통합연동시스템 아키텍처 설계 [산업체논문] 오지현, 김천영, 장영찬, 지철규, 홍영석(국방과학연구소)	모델 기반 Application Lifecycle Management [후원업체] 오윤민 이사(㈜이에스지) TMMi 관점의 국내 SW 테스트 성숙도 현황 [후원업체] 윤진우 수석(STA 테스팅컨설팅㈜) A Research on the Process towards developing the Framework on Systematic Investigation for Reliability improvement upon Software(SIRIUS) [산업체논문] 박삼준, 이태호, 백옥현(국방과학연구소), 오정섭, 서달미(㈜NSE) 소프트웨어 R&D 프로젝트 검증을 위한 산출물과 활동 송상민, Amarmend Dashbalbar, 이병정(서울시립대), 이정원(아주대)	한국 자본시장 전산장애 예방을 위한 SW 품질관리 추진사례 [후원업체] 이치형 부장(KOSCOM) 정적 검증 시각화를 통한 품질 지표 관리 [후원업체] 차신 센터장(테스트마이다스) 국가 사이버안보 역량 강화를 위한 사이버보안 성숙도 모델 설계 [산업체논문] 이민재(㈜티큐엠에스), 손영동(고려대) SPL 기반의 무기체계 SW 통합개발환경 운용 개념 연구 [산업체논문] 백옥현, 이태호, 박삼준(국방과학연구소) 분산시스템에서의 공유자원 가용성을 고려한 실시간 제어방법 연구 [단편논문] 장부철, 정우진, 송대기, 신진범(국방과학연구소)
17:00-17:10	**휴식**			
	기조연설 III 장소: 그랜드홀 2			사회: 백종문 조직위원장(KAIST)
17:10-18:00	**어길수 부사장(삼성전자 소프트웨어센터)**			
18:00-21:00	**우수논문상, 공로상, 감사장 수여식 및 만찬** 장소: 그랜드홀 1			

1 월 29 일 (금)

시 간	행 사 내 용			
07:00-09:00	**조식**			
	논문 발표 D			
	D1: SW 검증 좌장 유준범 교수(건국대) 장소: 그랜드홀 2	**D2: 요구공학** 좌장 김순태 교수(전북대) 장소: 세미나실 1	**D3: 온톨로지 및 서비스** 좌장 홍장의 교수(충북대) 장소: 세미나실 2	**D4: SW 품질 2 및 SE 교육** 좌장 이병정 교수(서울시립대) 장소: 세미나실 3
09:00-10:40 (100 분)	정형 기법을 이용한 NFV Policy 들의 일치성 검증 [단편논문] 구근회(고려대), 강미영, 최진영(고품질융합소프트웨어센터), 이승익(ETRI) Safety Critical System 에서 요구사항 검증 및 통합 명세화 방법제안 [단편논문] 임혜선, 이석원(아주대) 소프트웨어 연구 문서 산출물의 추적을 위한 연관성 링크 정보 모델 기반 전문가 시스템 설계 백두산(아주대), 이병정(서울시립대), 이정원(아주대) 위기대응 매뉴얼 신뢰성 향상을 위한 검증방안 연구 [단편논문] 이 혁, 최진영(고려대)	국방 전투관리체계 개발을 위한 상호작용 유즈케이스 기반 요구사항 모델링 기법 김두환, 홍장의(충북대), 김동환(LIGNex1) PbD 7 대 기본원칙과 GQM 을 활용한 프라이버시 요구사항 도출 방법론 [단편논문] 조주혜, 이석원(아주대) 지능형 화폐 인식 SW 개발의 Visualization 적용 사례 및 임베디드 SW 개발 조직의 Visualization 적용 로드맵 제안 [산업체논문] 한동준(소프트웨어 안전성 보증 연구센터), 김은비, 한혁수(상명대), 엄영석, 정대식, 조달호(KISAN) 인지적 분석을 통한 임상 의사결정 지원 시스템 인터페이스 설계 [단편논문] 고동균, 김유찬, 윤완철(KAIST)	혈액종합검사에 특화된 온톨로지 구축: 설계와 활용 제현우, 박유경, 홍원의, 이문용(KAIST) 사용자의 서비스 사용 성향을 고려한 클라우드 서비스 추천 최비오, 박준석, 황제승, 김웅수, 윤동규, 염근혁(부산대) 이기종 시맨틱 온톨로지 시스템의 통합 검색 [단편논문] 황상원, 남영광(연세대) 소프트웨어 성능 가시화를 위한 툴 체인 개발 [단편논문] 강건희, 박보경, 장우성, 황준순, 권하은, 이한솔, 이현준, 김영철(홍익대)	PageRank 와 토픽 모델링을 이용한 소프트웨어 재사용 도메인 토픽 추출 시스템 개발 이용석, 남영광, 황상원(연세대) 소프트웨어 개발 프로세스를 대상으로 수행하는 추적성 분석의 세부 관계 정의 [단편논문] 김재엽, 이동아, 유준범(건국대) Legacy 시스템의 안전성 분석기법 및 사례연구 [단편논문] 김선주, 이석원(아주대) 프로젝트 중심 소프트웨어 공학 교육의 성과 분석과 개선 방안 [단편논문] 최은만, 김하영(동국대)
10:40-10:50	**휴식**			

	논문 발표 E			
	E1: SW 테스팅 3 좌장 이정원 교수(아주대) 장소: 그랜드홀 2	**E2: 정적 분석** 좌장 유준범 교수(건국대) 장소: 세미나실 1	**E3: SW 융합** 좌장 이찬근 교수(중앙대) 장소: 세미나실 2	**E4: SW 품질 3** 좌장 이병정 교수(서울시립대) 장소: 세미나실 3
10:50-12:10 (80 분)	DO-178C 기반 소프트웨어 동적 테스트 방안 [산업체논문] 윤상은, 이종민, 정영은(TTA) 안전 무결성 기준을 활용한 테스트 케이스 우선순위 [단편논문] 송옥수, 김보운, 최병주(이화여대) 확장된 라운드 트립 기법을 이용한 발사통제장치 테스트 기법 [단편논문] 배정호, 안세준, 장부철, 구봉주 (국방과학연구소) 안드로이드 어플리케이션 개발 생산성을 위한 권한 자동 검사 기법 [단편논문] 노승학, 인호(고려대)	안드로이드 이미지 로딩 라이브러리 비기능 품질 속성 비교 조성래, 정연철, 민상윤(KAIST) 코드마인드: 온더플라이 소스코드 정적분석 도구 [산업체논문] 신승철, 이욱세, 노상훈, 김제민(코드마인드) 유사도 메트릭을 이용한 컴포넌트 상호연관성 추출 [우수단편논문] 이재권, 김기섭, 정우성(충북대)	다양한 이해관계자들 간 정보공유를 위한 산업용 로봇 소프트웨어 설계 기법 [단편논문] 박진용, 유철중(전북대) 스마트 환경에서 디지털 사이니지와 이기종 디바이스간의 상호운용성을 지원하기 위한 메시지 플로팅 시스템 차민재, 이동우, 남태우, 염근혁(부산대) Automotive 소프트웨어 Security 향상을 위한 Coding Guideline 에 대한 연구 우충기, 정지현, 민상윤(KAIST)	소프트웨어 품질관리를 위한 이종 코드 변환 프레임워크 연구 [단편논문] 손현승, 김영철(홍익대) 역공학을 이용한 오픈소스의 소스코드 요구사항 도출 및 Best Practices 와의 비교 분석을 통한 소스코드의 품질평가 [단편논문] 위대한, 이석원(아주대) 소프트웨어 프로세스 인증제도의 국방 적용방향 [단편논문] 김영봉 유천수(한국국방연구원) 원자력 계측제어 소프트웨어의 안전성 분석을 위한 Safety Case 의 Arguments 개발 절차 [우수단편논문] 이동아, 유준범(건국대학교), 이장수 (한국원자력연구원)
12:10-12:30	**폐회식** 장소: 그랜드홀 2			사회: 이병정 학술위원장(서울시립대)
13:00-18:00		**SW 상시모니터링기술연구단** 장소: 세미나실 1 (13:00-18:00)		

* 위 일정은 사정에 따라 변경될 수가 있습니다.

KCSE 2016 튜토리얼

튜토리얼 T1: SW 공학과 최적화 알고리즘

- 일시: 1월 27일(수) 13:30~15:30
- 장소: 그랜드홀 2
- 제목: 검색 기반 소프트웨어 공학(Search Based Software Engineering)
- 연사: 유신 교수(KAIST)

- 튜토리얼 초록:

  Search-Based Software Engineering은 SE문제를 meta-heuristic optimisation 알고리즘을 이용해서 푸는 일군의 기법을 의미한다. 본 튜토리얼은 기본적인 meta-heuristic optimisation 알고리즘의 소개에서 시작해서 state-of-the-art인 응용 문제들의 case study를 다루는 것 까지를 범위로 삼아 기존에 최적화 알고리즘을 사용해보지 못한 사람도 SE 문제에 이를 적용해볼 수 있도록 하는 것을 목표로 한다.

- 유신 교수 약력:

  2015년~현재 KAIST 조교수
  2012~2015 University College London, UK, 조교수
  2009년 King's College London, UK, 전산학 박사
  2006년 King's College London, UK, 소프트웨어공학 석사
  연구분야: Search-Based Software Engineering, Software Testing (regression testing), Fault Localisation, Self Adaptation

튜토리얼 T2: SW Process

- 일시: 1월 27일(수) 13:30~15:30
- 장소: 세미나실 1
- 제목: 자동차 SW의 안전성을 위한 프로세스 모델: Automotive SPICE 와 CMMI
- 연사: 한혁수 교수(상명대)

- 튜토리얼 초록:

  자동차의 전자화와 스마트화로 인해 자동차 SW의 안전성(Safety)에 대한 관심이 높아지고 있다. 본 발표에서는 자동차 SW의 안전성을 확보하기 위하여 ISO 26262와 더불어 현업에서 적용되고 있는 프로세스 모델인 Automotive SPICE를 살펴보고자 한다.

  프로세스 모델의 필요성에 대해 살펴보고 Automotive SPICE의 참조모델과 심사모델의 개요에 대해 설명한다. 강의 후반에는 CMMI와의 차이점을 분석하고, 프로세스 모델을 조직에 성공적으로 적용하기 위한 가이드라인도 살펴본다.

◆ 한혁수 교수 약력:

2015. 06 ~ 현재 ITRC 소프트웨어 안전성 보증 연구센터 센터장
2014. 02 ~ 현재 상명대학교 ICT융합대 학장
2012. 3 ~ 2014. 2 소프트웨어공학 소사이어티 회장
2011 ~ 현재 NIPA 소프트웨어프로세스 품질인증심의회 심의위원
2004. 05 ~ 현재 공인 Introduction to CMMI 강사
2004. 03 ~ 현재 공인 CMMI 심사원
2003. 3 ~ 2004. 2 한국소프트웨어진흥원 소프트웨어공학센터 소장
1992. 2 University of South Florida, 전산학 박사
1987. 8 서울대학교 계산통계학과 (석사)

연구분야: SW 프로세스, SW 품질, SW 안전성 분석, SW 교육, HCI

튜토리얼 T3: SW Product Line

◆ 일시: 1월 27일(수) 13:30~15:30
◆ 장소: 세미나실 2
◆ 제목: 임베디드 SW product line 개발 및 진화
◆ 연사: 이근 박사(삼성전자)

◆ 튜토리얼 초록:

본 튜토리얼에서는 Embedded SW의 주요 특성 및 그 특성으로 인한 개발 라이프 사이클의 문제점 고찰하고, Embedded SW Product-line 개발에 적용되는 기술과 주요 이슈를 살펴보며, Embedded SW Product-line 유지보수에 적용되는 기술과 주요 이슈에 대하여 논한다.

◆ 이근 박사 약력:

2015.7 ~ Present Bio Processor Solution Development
2013.1 ~ 2015.7 AP/CP Quality Management & Infrastructure
2008.7 ~ 2013.1 SW Platform Engineering
2007.3 ~ 2009.12 Release System, Code Analysis Tool Development
2007.3 ~ 2008.6 SW Process, Agile Development Method
1999 ~ 2005 University of Southern California, USA, SE MS, PhD
1993 ~ 1999년 Korea Information System (1993 ~ 1999) System Integration
1933년 Sogang Uiversity, Master in Mathematics

KCSE 2016 기조연설

기조연설 I

- 일시: 1월 27일(수) 16:00~17:00
- 장소: 그랜드홀 2
- 제목: 소프트웨어중심사회의 도래와 소프트웨어 안전의 확보
- 연사: 김진형 소장(소프트웨어정책연구소)

기조연설 II

- 일시: 1월 27일(수) 17:00~18:00
- 장소: 그랜드홀 2
- 제목: 미래 ICT사회를 선도할 소프트웨어공학人의 미래
- 연사: 심현택 소장(정보통신산업진흥원 소프트웨어공학센터)

기조연설 III

- 일시: 1월 28일(목) 17:10~18:00
- 장소: 그랜드홀 2
- 제목: 산업계의 소프트웨어 동향과 삼성전자의 방향
- 연사: 어길수 부사장(삼성전자 소프트웨어센터)

UML 상태 다이어그램 기반 테스트 스크립트 자동 생성 도구[1]

이영우, 최현재, 채흥석

부산대학교 정보컴퓨터공학부
부산광역시 금정구 부산대학로 63 번길 2(장전동)
amorepooh@gmail.co.kr, gonoki@pusan.ac.kr, hschae@pusan.ac.kr

요약: 모델 기반 테스트는 테스트 모델을 분석하여 테스트 케이스를 자동 생성하는 기법이다. 모델 기반 테스트를 통해 생성한 테스트 케이스는 수동으로 작성한 테스트 케이스와 달리 테스트 케이스 작성자에 무관하게 일정 이상의 테스트 품질을 가진다. 또한 모델 기반 테스트는 테스트 모델을 재사용을 통해 테스트 케이스 생성을 자동화함으로써 테스트 케이스 생성 비용을 절감할 수 있다. 이러한 장점으로 인해 최근에 모델 기반 테스트가 많이 수행되고 있다. 본 연구에서는 테스트 모델을 기반으로 테스트 스크립트를 자동으로 생성하는 도구를 개발하였다. 본 도구는 테스트 모델을 분석하여 테스트 경로와 테스트 데이터를 생성함으로써 테스트 케이스를 생성한다. 그리고 생성된 테스트 케이스를 JUnit 테스트 스크립트로 자동 생성한다. Traffic Light Controller 테스트 모델을 대상으로 적용한 결과 전이 커버리지를 100% 만족하는 실행 가능한 JUnit 테스트 스크립트를 자동 생성하였다.

핵심어: UML 상태 다이어그램, 경로 선택, 테스트 데이터 분석

1. 서론

모델 기반 테스팅은 작성된 모델을 기반으로 시스템의 행위를 분석하고, 테스트 스크립트를 생성하여 테스트를 수행하는 기술이다[1].

테스트 케이스를 수동으로 작성하면 개인의 능력에 따라 테스트 케이스의 품질이 달라질 수 있으며, 테스트 케이스를 생성하는 데 많은 비용이 발생한다. 또한 시스템이 변경되었을 때 추가 비용이 발생할 수 있다[2]. 이러한 문제점들을 극복하기 위해 최근에는 모델 기반 테스트가 많이 수행되고 있다[3,4]. 모델을 기반으로 테스트를 수행하면, 테스트 케이스 생성이 자동화가 가능하므로 일정 이상의 테스트 케이스의 품질을 확보할 수 있으며, 테스트 케이스 작성을 위한 비용을 절감할 수 있다[5]. 또한 자동으로 생성된 테스트 스크립트를 이용해 테스트를 자동화함으로써 테스트를 수행하는 데 필요한 비용도 절감할 수 있다.

본 연구에서는 모델을 기반으로 테스트 스크립트를 자동으로 생성하는 도구를 개발하였다. 상태 다이어그램을 입력으로 받아 모델을 분석한 뒤, 전이 커버리지를 기반으로 테스트 경로를 선택한다. 선택한 테스트 경로에 포함된 조건들을 분석하여 테스트 데이터를 결정한 뒤, 이를 바탕으로 테스트 스크립트를 생성한다.

본 논문의 구성은 다음과 같다. 2 장에서 본 도구에서 사용한 도구인 Concolic Solver 에 대해 소개한다. 3 장에서는 모델 기반 테스트 스크립트 생성 도구에 대해서 소개하며, 4 장에서는 예제 모델을 이용한 도구 실행 결과에 대해 서술한다. 5 장에서는 관련 기존 도구에 대해서 소개하며, 6 장에서는 결론 및 향후 연구에 대해 서술한다.

2. Concolic Solver

Concolic Solver 란 테스트 스크립트의 입력 값을 결정하는 하이브리드 도구이다[6]. 하이브리드 도구라 불리는 이유는 심볼릭 실행과 제약 해결의 2 가지 작업으로 구성되기 때문이다.

심볼릭 실행은 프로그램의 특정 부분을 실행하기 위한 입력 값을 결정하기 위해, 프로그램을 분석하는 것을 말한다[7]. 연산이나 행위의 수행에 따라 변수 값에 지속적으로 변경이 발생하기 때문에 프로그램의 특정 부분을 실행하는 입력 값을 생성하기 위해서는 변수의 변경을 추적하기 위한 분석이 필요하다. 심볼릭 실행을 통해 문장을 변경함으로써 변수의 변경을 고려하여 프로그램의 구조 분석을 수행할 수 있다.

제약 해결은 논리식들을 분석하여 모든 논리식을 참으로 만드는 해를 찾아내는 것을 말한다[8]. Concolic Solver 에서는 제약 해결을 통해 심볼릭 실

[1] 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성 지원사업의 연구결과로 수행되었음 (IITP-2015-R0992-15-1014)
본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성 지원사업의 연구결과로 수행되었음" (IITP-2015-H8501-15-1017)

행의 결과로 전달된 논리식들을 참으로 만드는 테스트 데이터를 분석한다. [그림 1]은 Concolic Solver 의 동작 예를 나타낸다.

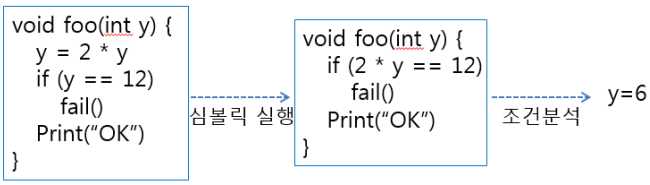


그림 1 Concolic Solver 동작 예

예를 들어 Concolic Solver 를 이용하여 3 행의 fail() 함수를 실행하는 테스트 데이터 생성 절차를 설명한다. 좌측 코드에 대해 심볼릭 실행을 수행하면, 1 행의 "y = 2 * y"로 인해 2 행 이후의 y 값은 "2 * y"로 대체된다. 그에 따라 2 행의 조건 "y == 12"는 "2 * y == 12"로 치환된다. 심볼릭 실행을 수행하고 나서 제약 해결을 통해 fail()함수를 실행하기 위한 입력 값을 분석한다. 제약 해결을 통해 테스트 데이터 6 이 결정된다.

3. 모델 기반 테스트 스크립트 생성 도구

본 연구에서 개발한 상태 다이어그램으로부터 테스트 스크립트를 자동으로 생성하는 도구의 구조와 각 모듈에 대해 설명한다.

3.1 구조도

모델 기반 테스트 스크립트 생성 도구는 테스트 모델 분석, 테스트 경로 선택, 테스트 데이터 분석, 테스트 스크립트 생성으로 구성된다. [그림 2]는 모델 기반 테스트 스크립트 생성 도구의 구조를 표현한다.

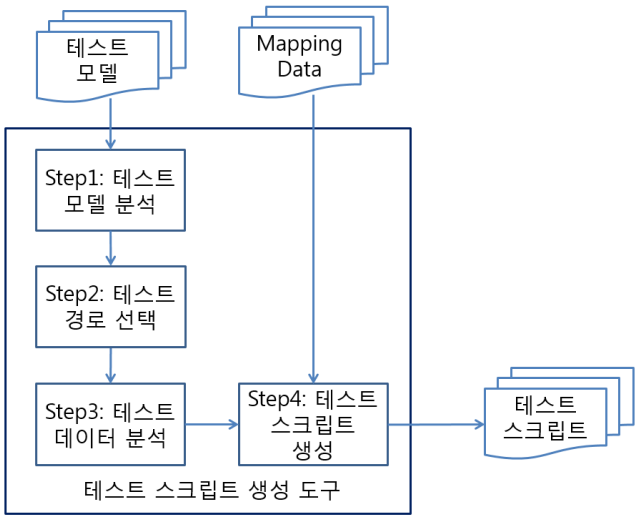


그림 2 모델 기반 테스트 스크립트 생성 도구 구조

모델 기반 테스트 스크립트 생성 도구는 입력으로 테스트 모델과 Mapping Data 를 사용한다. 테스트 모델은 모델링 도구인 Magic Draw 를 이용하여 그린 상태 다이어그램을 이용한다. Mapping Data 는 상태 다이어그램으로 그린 시스템의 추가 정보를 명시한 csv 파일이다.

본 연구에서 개발한 도구의 동작은 다음 4 가지 단계로 구성된다. 모델 분석 과정은 Magic Draw 에서 제공하는 API 를 이용하여 상태 다이어그램을 추상화된 중간 모델로 변환하고 테스트 모델 평탄화 기법을 통해 중간 모델을 확장하는 단계이다. 테스트 경로 선택은 중간 모델로부터 테스트 경로를 추출하는 단계이다. 테스트 데이터 분석은 추출된 테스트 경로가 포함하는 모든 조건들에 대해 Concolic Solving 을 수행하여 추출한 테스트 경로를 실행 가능하게 하기 위한 테스트 데이터를 결정하는 단계이다. 테스트 스크립트 생성은 추출된 테스트 경로와 테스트 데이터를 이용하여 실행 가능한 JUnit 테스트 스크립트를 생성하는 단계이다.

본 도구의 동작을 설명하기 위해 Elevator 테스트 모델을 이용한다. Elevator 테스트 모델은 정지 상태인 Idle 과 이동 중 상태인 Moving 으로 구성된다. [그림 3]은 Elevator 시스템의 테스트 모델을 표현한다.

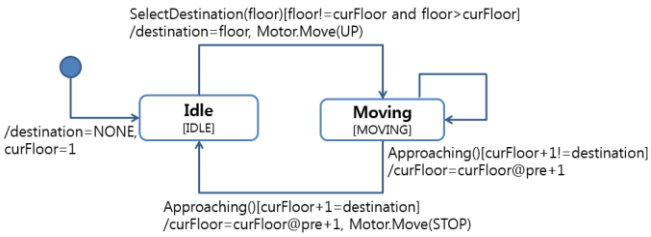


그림 3 Elevator 상태 다이어그램

Elevator 시스템은 Idle 상태와 Moving 상태를 가지며, 목표 층을 선택하는 SelectDestination(floor) 연산과 목표 층에 접근하는 Approaching() 연산을 가진다. 3.2 절부터 3.5 절까지는 Elevator 시스템을 이용하여 각 단계 별로 본 도구의 동작 흐름을 예를 통해 보여준다.

3.2 Step1: 테스트 모델 분석

테스트 모델 분석은 테스트 중간 모델 구축과 모델 평탄화의 2 가지 단계로 구성된다. 테스트 중간 모델 구축은 Magic Draw 도구를 이용해 작성된 테스트 모델 파일을 분석하여 추상화된 중간 모델을 구축하는 것이다. 중간 모델 구축을 통해 테스트 모델 분석을 용이하게 하고 다른 모델링 도구와의 확장성을 향상시킨다. 모델 평탄화 기법은 UML 표기법을 이용하여 작성된 테스트 모델을 FSM 형태로 단순화시

키는 방법을 말한다[9]. UML 의 복합 상태, 직교 상태, 하위머신 상태를 단순한 상태로 변환함으로써 평탄화 수행 전보다 더 많은 테스트 케이스를 생성할 수 있다.

Elevator 테스트 모델에는 테스트 모델 분석 단계에서 평탄화할 하위 상태가 없다. 따라서 모델 평탄화 과정 없이 중간 모델 구축 과정만 수행하여 모델을 분석하였다.

3.3 Step2: 테스트 경로 선택

모델을 기반으로 테스트 스크립트를 생성할 때 사용하는 대표적인 전략으로 전이 커버리지 기반이 있다. 전이 커버리지 기반 생성 전략은 상태 다이어그램을 구성하는 모든 전이를 1 회 이상 방문하도록 테스트 경로를 탐색하는 기법이다. 본 연구에서 개발한 도구는 위에서 언급한 전이 커버리지를 기반으로 한 테스트 경로 탐색 전략을 지원한다.

Elevator 테스트 모델을 이용한 테스트 경로 선택 단계에서는 테스트 모델의 전이 조건들을 분석하여 테스트 경로를 선택한다. [표 1]은 선택한 4 개의 전이에 포함된 조건들을 보여준다.

표 1 Elevator 모델의 전이에 포함된 조건

전이	조건
Initial State - Idle	/destination=NONE, curFloor=1
Idle - Moving	[floor!=curFloor and floor>curFloor] /destination=floor, Motor.Move(UP)
Moving - Moving	[curFloor+1 != destination] /curFloor=curFloor@pre+1
Moving - Idle	[curFloor+1 = destination] /curFloor=curFloor@pre+1, Motor.Move(STOP)

본 연구에서는 전이 커버리지를 테스트 경로 선택 기준으로 이용하였다. 테스트 경로 선택 결과 Elevator 테스트 모델의 모든 전이를 방문하는 테스트 경로인 Idle(e:SelectDestination) → Moving(e:Approaching) → Moving(e:Approaching) → Idle 가 선택되었다. 위 경로에서 Idle 과 Moving 은 상태명을 의미하고 SelectDestination 과 Approaching 은 괄호된 상태에서 발생된 이벤트명을 뜻한다.

3.4 Step3: 테스트 데이터 분석

실행 가능한 테스트 스크립트를 생성하기 위해서는 테스트 경로뿐만 아니라 테스트 경로를 실행 가능하게 하는 테스트 데이터도 필요하다. 테스트 데이터 분석에서는 테스트 경로에 포함된 조건들을 대상으로 Concolic Solver 를 사용하여 모든 조건을 만족하는 해를 찾아낸다. Concolic Solver 의 심볼릭 실행 모듈은 자체 개발하였으며, 제약 해결 도구는 기존의 z3 Solver 를 사용하였다.

Elevator 테스트 모델을 이용한 테스트 데이터 분석 단계에서는 선택한 테스트 경로를 실행 가능하게 하기 위한 테스트 데이터를 생성하였다. 전이에 따른 변수 값 변경을 추적하기 위해 심볼릭 실행을 수행하였다. [표 2]는 [표 1]의 조건들에 대해 심볼릭 실행을 수행한 결과를 보여준다.

표 2 Elevator 모델의 심볼릭 실행 결과

전이	조건
Initial State - Idle	/destination=NONE, curFloor=1
Idle - Moving	[floor!=1 and floor>1] /destination=floor, Motor.Move(UP)
Moving - Moving	[1+1 != floor] /curFloor=curFloor@pre+1
Moving - Idle	[1+1+1 = floor] /curFloor=curFloor@pre+1, Motor.Move(STOP)

제약 해결 도구인 z3 Solver 를 이용하여 심볼릭 실행을 수행한 결과의 모든 조건을 만족하는 해를 결정한다. 두 번째 전이에서 floor 는 1 이 아니며, 1 보다 커야 한다는 조건과 세 번째 전이에서 floor 는 2 가 아니라는 조건, 마지막 네 번째 전이에서 floor 는 3 이라는 조건을 분석하여 모든 조건을 만족시키는 테스트 데이터로서 floor 값이 3 으로 결정되었다.

3.5 Step4: 테스트 스크립트 생성

선택한 테스트 경로와 테스트 데이터 그리고 시스템의 추가 정보를 명시한 Mapping Data 를 이용하여 실행 가능한 테스트 스크립트를 생성한다. 본 연구에서는 자바의 대표적인 테스트 프레임워크인 JUnit 에서 동작하는 테스트 스크립트를 생성한다.

Elevator 테스트 모델을 이용한 테스트 스크립트생성 단계에서는 단계 2 에서 선택된 테스트 경로와 단계 3 에서 결정된 테스트 데이터 그리고 Elevator 테스트 모델의 Mapping Data 를 이용하여 실행 가능한 테스트 스크립트를 생성하였다. [표 3]은 Elevator 테스트 모델의 Mapping 데이터를 보여준다.

표 3 Elevator 모델의 Mapping Data

상태 다이어그램에 나타난 연산/속성	구체화된 연산/속성
Elevator	Elevator elevator = new Elevator()
SelectDestination (floor)	elevator.SelectDestination(floor:Int) :void
Approaching()	elevator.Approaching():void
IDLE	elevator.IDLE:Int=0
MOVING	elevator.MOVING:Int=1
curFloor	elevator.curFloor:Int
destination	elevator.destination:Int

선택된 테스트 경로와 결정된 테스트 데이터를 테스트하는 JUnit 테스트 스크립트를 생성하였다. [표 4]는 Elevator 테스트 모델을 테스트하는 테스트 스크립트를 나타낸다.

표 4 Elevator 모델로부터 생성한 테스트 스크립트

```
@Test
  public void test1() {
      Elevator elevator = new Elevator();
      assertTrue(elevator.state==elevator.IDLE);
      elevator.SelectDestination(3);
      assertTrue(elevator.state==elevator.MOVING);
      elevator.Approaching();
      assertTrue(elevator.state==elevator.MOVING);
      elevator.Approaching();
      assertTrue(elevator.state==elevator.IDLE);
  }
```

테스트 경로를 이루는 전이의 이벤트가 발생할 때마다 이벤트에 해당하는 객체의 메소드가 호출된다. 테스트 데이터 분석을 통해 결정된 테스트 데이터 floor=3 은 SelectDestination()연산자의 매개변수로 입력된다. 각 전이가 수행된 후에 도달하는 시스템의 상태를 assertion 문장을 이용해서 시스템의 동작을 테스트 하였다.

4. 적용 사례

본 장에서는 Traffic Light Controller(TLC) 예제 모델을 이용하여 모델 기반 테스트 스크립트 생성 도구의 적용 사례에 대해 설명한다. TLC 모델은 UML 기반 명세, 확인과 검증 도서인 [10]의 예제에서 발췌한 것이다. 본 도구는 상태 다이어그램과 Mapping Data 를 이용하여 JUnit 테스트 스크립트를 생성한다.

4.1 예제 테스트 모델

본 연구에서는 TLC 시스템의 상태 다이어그램에 대해 모델 기반 테스트 스크립트 생성 도구를 적용하여 JUnit 테스트 스크립트를 생성하였다. [그림 4]는 TLC 상태 다이어그램으로써 교통 신호를 제어하는 제어기의 행위를 표현하는 모델이다.

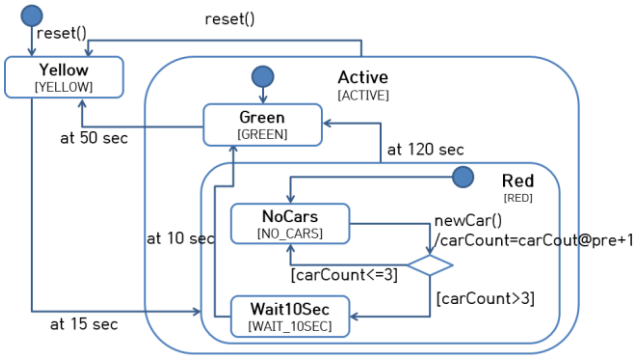


그림 4 TLC 상태 다이어그램

TLC 시스템은 Yellow, Green, NoCars, Wait10Sec 상태를 가진다. 그리고 newCar 이벤트와 reset 이벤트을 가지며 시간 경과에 따라 이벤트가 발생한다.

4.2 예제 Mapping Data

상태 다이어그램은 시스템의 행위를 정의한 모델이므로, 시스템의 속성이나 연산에 대한 정보가 부족하다. 따라서 상태 다이어그램만 이용하여 테스트 스크립트를 생성하기 어렵기 때문에 상태 다이어그램에서 그린 시스템에 대한 부족한 정보를 Mapping Data 에 명시하여 테스트 스크립트를 생성하는 데 활용한다. [표 5]는 TCL 상태 다이어그램 모델의 추가 정보를 명시한 Mapping Data 를 보여준다.

표 5 TLC 모델의 Mapping Data

상태 다이어그램에 나타난 연산/속성	구체화된 연산/속성
TLC	TLC tlc = new TLC()
reset()	tlc.reset():void
newCar()	tlc.newCar():void
carCount	tlc.carCount:Int
YELLOW	tlc.YELLOW:Int=0
ACTIVE	tlc.ACTIVE:Int=1
RED	tlc.RED:Int=2
GREEN	tlc.GREEN:Int=3
NOCARS	tlc.NOCARS:Int=4
WAIT10SEC	tlc.WAIT10SEC:Int=5

TLC 시스템의 Mapping Data 는 총 10 개의 연산과 속성으로 구성된다. 첫 번째 열에는 TLC 시스템의 상태 다이어그램에 대한 이벤트, 액션 또는 조건이 명시되어 있으며, 두 번째 열에는 상태 다이어그램의 각 요소들에 대응되는 객체의 연산자와 변수, 리턴 타입, 매개변수의 타입 그리고 초기값이 명시되어 있다.

4.3 결과

TLC 테스트 모델로부터 테스트 경로와 테스트 데이터를 분석하고 Mapping Data 를 이용하여 실행 가능한 JUnit 테스트 스크립트를 생성하였다. 생성 결과 6 개의 테스트 케이스가 생성되었다. [표 6]은 TLC 모델로부터 생성한 JUnit 테스트 스크립트를 보여준다.

표 6 TLC 모델로부터 생성한 테스트 스크립트

```
@Test
public void test1() {
    TLC tlc = new TLC();
    tlc.carCount = 3;
    tlc.reset();
```

```
        assertTrue(tlc.state==tlc.YELLOW);
        Thread.sleep(15);
        assertTrue(tlc.state==tlc.NOCARS);
        tlc.newCar();
        assertTrue(tlc.state==tlc.WAIT10SEC);
        Thread.sleep(10);
        assertTrue(tlc.state==tlc.GREEN);
        Thread.sleep(50);
        assertTrue(tlc.state==tlc.YELLOW);
    }
    @Test
    public void test2() {
        TLC tlc = new TLC();
        tlc.carCount = 3;
        tlc.reset();
        assertTrue(tlc.state==tlc.YELLOW);
        Thread.sleep(15);
        assertTrue(tlc.state==tlc.NOCARS);
        tlc.newCar();
        assertTrue(tlc.state==tlc.WAIT10SEC);
        Thread.sleep(10);
        assertTrue(tlc.state==tlc.GREEN);
        tlc.reset();
        assertTrue(tlc.state==tlc.YELLOW);
    }
    @Test
    public void test3() {
        TLC tlc = new TLC();
        tlc.carCount = 3;
        tlc.reset();
        assertTrue(tlc.state==tlc.YELLOW);
        Thread.sleep(15);
        assertTrue(tlc.state==tlc.NOCARS);
        tlc.newCar();
        assertTrue(tlc.state==tlc.WAIT10SEC);
        Thread.sleep(120);
        assertTrue(tlc.state==tlc.GREEN);
    }
    @Test
    public void test4() {
        TLC tlc = new TLC();
        tlc.carCount = 3;
        tlc.reset();
        assertTrue(tlc.state==tlc.YELLOW);
        Thread.sleep(15)
        assertTrue(tlc.state==tlc.NOCARS);
        tlc.newCar();
        assertTrue(tlc.state==tlc.WAIT10SEC);
        tlc.reset();
        assertTrue(tlc.state==tlc.YELLOW);
    }
    @Test
    public void test5() {
        TLC tlc = new TLC();
        tlc.carCount = 2;
        tlc.reset();
        assertTrue(tlc.state==tlc.YELLOW);
        Thread.sleep(15)
        assertTrue(tlc.state==tlc.NOCARS);
        tlc.newCar();
        assertTrue(tlc.state==tlc.NOCARS);
        Thread.sleep(120)
        assertTrue(tlc.state==tlc.GREEN);
    }
    @Test
    public void test6() {
        TLC tlc = new TLC();
        tlc.carCount = 2;
        tlc.reset();
        assertTrue(tlc.state==tlc.YELLOW);
        Thread.sleep(15)
        assertTrue(tlc.state==tlc.NOCARS);
        tlc.newCar();
        assertTrue(tlc.state==tlc.NOCARS);
        tlc.reset();
        assertTrue(tlc.state==tlc.YELLOW);
    }
```

TLC 모델은 총 6 개의 테스트 케이스를 이용해서 전이 커버리지를 100% 만족시킨다. 전이 커버리지를 만족하는 테스트 경로를 선택하고, 각 경로 별로 조건을 분석하여 테스트 데이터(carCount = 2 또는 carCount = 3)를 찾아낸 뒤, 이를 바탕으로 테스트 스크립트를 생성하였다. 각 전이를 실행한 후, 도달하는 상태를 assertion 문장을 이용해서 시스템의 동작을 테스트하였다. [그림 5-10]은 각 테스트 메소드 별 테스트 실행 전이를 보여준다.

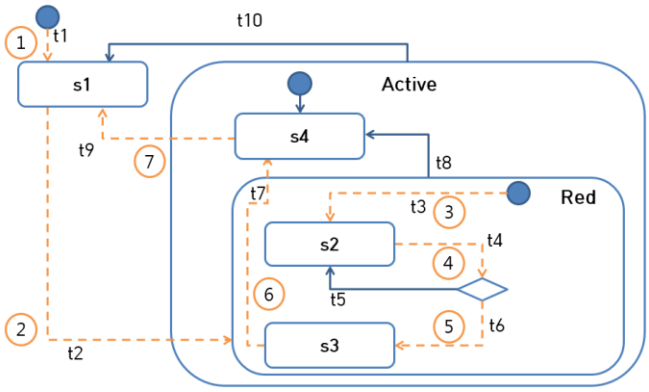


그림 5 test1 이 커버하는 전이

테스트 경로 t1→t2→t3→t4→t6→t7→t9 를 실행하기 위한 테스트 데이터는 carCount=3 이다. 위 테스트 경로와 테스트 데이터를 실행하기 위해 test1 테스트 메소드를 자동 생성하였다. test1 테스트 메소드 실행 결과 [그림 5]에서 점선으로 표현한 전이를 실행하였다.

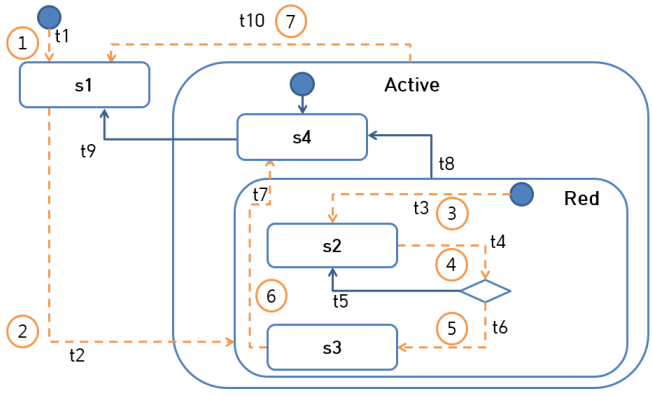


그림 6 test2 가 커버하는 전이

테스트 경로 t1→t2→t3→t4→t6→t7→t10 를 실행하기 위한 테스트 데이터는 carCount=3 이다. 위 테스트 경로와 테스트 데이터를 실행하기 위해 test2 테스트 메소드를 자동 생성하였다. test2 테스트 메소드 실행 결과 [그림 6]에서 점선으로 표현한 전이를 실행하였다.

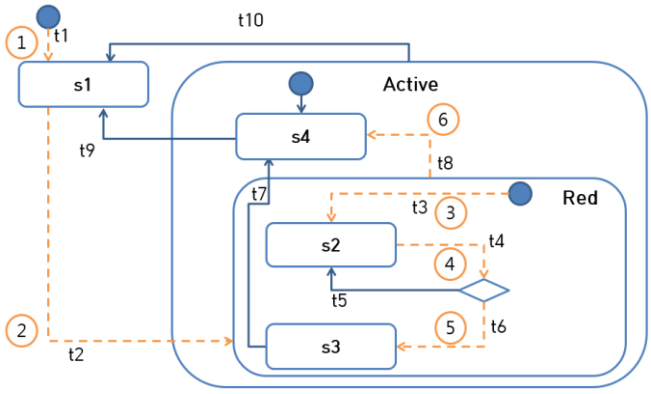


그림 7 test3 이 커버하는 전이

테스트 경로 t1→t2→t3→t4→t6→t8 를 실행하기 위한 테스트 데이터는 carCount=3 이다. 위 테스트 경로와 테스트 데이터를 실행하기 위해 test3 테스트 메소드를 자동 생성하였다. test3 테스트 메소드 실행 결과 [그림 7]에서 점선으로 표현한 전이를 실행하였다.

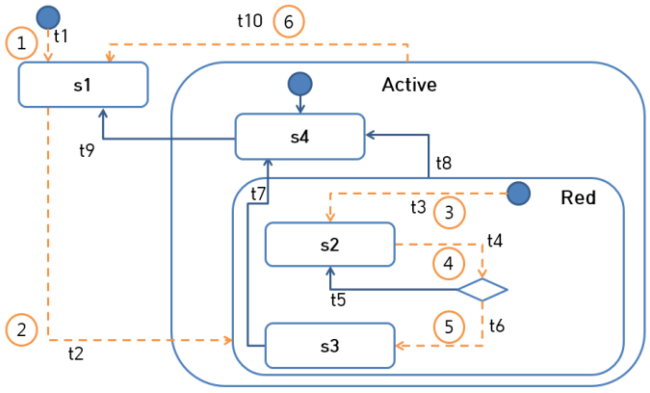


그림 8 test4 가 커버하는 전이

테스트 경로 t1→t2→t3→t4→t6→t10 를 실행하기 위한 테스트 데이터는 carCount=3 이다. 위 테스트 경로와 테스트 데이터를 실행하기 위해 test4 테스트 메소드를 자동 생성하였다. test4 테스트 메소드 실행 결과 [그림 8]에서 점선으로 표현한 전이를 실행하였다.

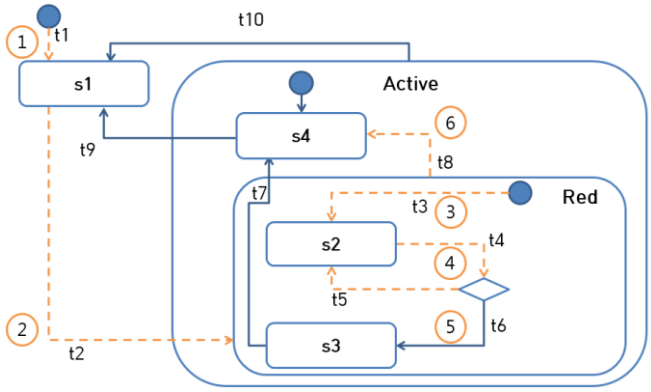


그림 9 test5 가 커버하는 전이

테스트 경로 t1→t2→t3→t4→t5→t8 를 실행하기 위한 테스트 데이터는 carCount=2 이다. 위 테스트 경로와 테스트 데이터를 실행하기 위해 test5 테스트 메소드를 자동 생성하였다. test5 테스트 메소드 실행 결과 [그림 9]에서 점선으로 표현한 전이를 실행하였다.

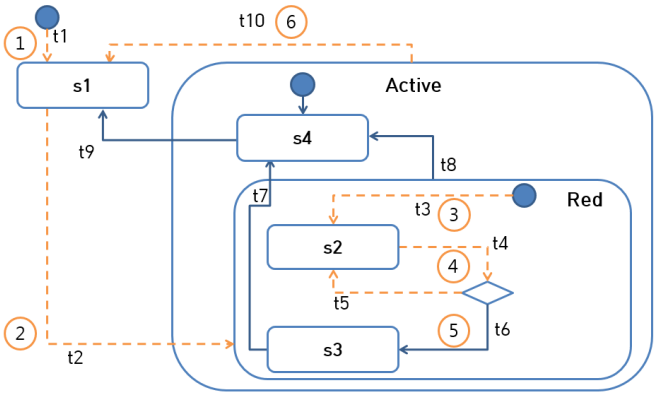


그림 10 test6 이 커버하는 전이

테스트 경로 t1→t2→t3→t4→t5→t10 를 실행하기 위한 테스트 데이터는 carCount=2 이다. 위 테스트 경로와 테스트 데이터를 실행하기 위해 test6 테스트 메소드를 자동 생성하였다. test6 테스트 메소드 실행 결과 [그림 10]에서 점선으로 표현한 전이를 실행하였다.

5. 관련 도구

기존의 모델 기반 테스트 케이스 자동 생성 도구로 Spec Explorer, SinelaboreRT, TestOptimal 이 있다. [표 7]은 기존의 모델 기반 테스트 도구들과 본 연구에서 개발한 도구와의 차이점을 나타내었다.

표 7 기존의 모델 기반 테스트 도구

도구명	비교 기준	
	테스트 데이터 생성	스크립트 생성
Spec Explorer[11]	변수 값 직접 입력 변수 별 값 범위 직접 입력	Visual Testing Tools unit test 형태의 테스트 스크립트 생성
SinelaboreRT[12]	지원 안 함	스크립트를 생성하지 않음
TestOptimal[13]	변수 값 직접 입력	Selenium API 를 호출하는 테스트 스크립트 생성

기존의 도구들을 살펴보면 본 연구에서 개발한 도구와 동일하게 공통적으로 테스트 경로 선택 과정을 수행한다. 그러나 테스트 스크립트의 입력으로 사용되는 테스트 데이터를 자동으로 생성하는 것이 아니라, 수동으로 사용자에게 입력을 요구한다. 출력 결과는 특정 프레임워크가 동작하기 위한 테스트 스크립트를 생성하거나 스크립트를 생성하지 않았다. 본 연구에서 개발한 도구는 선택한 경로가 포함하는 모든 조건들을 만족하는 테스트 데이터를 찾아내며, 출력으로 실행 가능한 JUnit 테스트 스크립트를 생성함으로써 테스트 자동화가 가능하다.

6. 결론 및 향후 연구

테스트를 수행하기 위한 테스트 케이스를 생성할 때, 사람이 수동으로 생성을 하면 개인의 역량에 따라 테스트 케이스의 품질이 달라지며, 테스트 케이스를 생성하는 데 많은 비용이 발생한다.

본 연구에서는 이러한 문제를 해결하기 위해 테스트 모델로부터 테스트 스크립트를 자동 생성하는 도구를 개발하였다. 모델을 기반으로 자동으로 테스트 스크립트를 생성하면 일정 이상의 품질을 가진 테스트 스크립트를 생성할 수 있으며, 테스트 스크립트를 생성하는 데 필요한 비용을 절감할 수 있다. 또한 시스템이 변경되더라도 적은 비용으로 테스트 스크립트를 생성해낼 수 있다. 자동으로 생성한 테스트 스크립트를 기반으로 테스트를 수행하면, 테스트를 자동화 할 수 있으며, 이로 인해 테스트를 수행하는 데 필요한 비용 또한 절감할 수 있다.

도구의 입력으로는 상태 다이어그램을 이용하며, 모델을 분석한 뒤, 전이 커버리지 기반으로 테스트 경로를 선택한다. 선택한 경로에 존재하는 모든 조건을 만족하는 테스트 데이터를 찾아낸 뒤, 이를 바탕으로 JUnit 테스트 스크립트를 생성한다. Traffic Light Controller 테스트 모델을 대상으로 실험한 결과 전이 커버리지를 100% 만족하기 위해 6 개의 테스트 케이스를 가지는 실행 가능한 JUnit 테스트 스크립트를 자동 생성하였다.

향후 연구에서는 보다 완전한 테스트를 위하여 프로토콜 상태 머신을 이용한 테스트 케이스 및 테스트 스크립트 생성 연구를 수행할 계획이다. 또한 변수 값 초기화를 통한 실행 가능한 경로 선택이 아닌 동적 경로 분석을 통해 실제로 전이가 발생하는 경로를 선택하는 기법을 연구할 계획이다.

참고문헌

[1] Korel, Bogdan, Luay H. Tahat, and Boris Vaysburg, "Model based regression test reduction using dependence analysis," Software Maintenance, Proc, International Conference on, IEEE, 2002.

[2] Zhu, Hong, Patrick AV Hall, and John HR May, "Software unit test coverage and adequacy," Acm computing surveys (csur) Vol.29, pp.366-427, 1997.

[3] Holt, Nina Elisabeth, Lionel C. Briand, and Richard Torkar, "Empirical evaluations on the cost-effectiveness of state-based testing: An industrial case study," Information and Software Technology, Vol.56, pp.890-910, 2014.

[4] Shafique, Muhammad, and Yvan Labiche, "A systematic review of state-based test tools," International Journal on Software Tools for Technology Transfer Vol.17, pp.59-76, 2013.

[5] Dick, Jeremy, and Alain Faivre, "Automating the generation and sequencing of test cases from model-based specifications," FME'93: Industrial-Strength Formal Methods. Springer Berlin Heidelberg, 1993.

[6] Majumdar, Rupak, and Koushik Sen, "Hybrid concolic testing," Software Engineering, 2007. ICSE 2007, 29th International Conference on. IEEE, 2007.

[7] King, James C, "Symbolic execution and program testing," Communications of the ACM, Vol.19, pp. 385-394, 1976.

[8] Jaffar, Joxan, and J-L. Lassez, "Constraint logic programming," Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. ACM, 1987.

[9] 최현재, 채흥석, "평탄화를 이용한 모델 기반 테스트 케이스 생성 도구," In Proceedings of 2015 Korea Conference on Software Engineering (KCSE 2015), pp. 201-204, 2015.

[10] Drusinsky, Doron, Practical Uml-based Specification, Validation, and Verification of Mission-critical Software: Space Exploration and Defense Software Examples in Practice, pp.20, Dog ear publishing, 2011.

[11] Microsoft, Spec Explorer, visualstudiogallery.msdn.microsoft.com/271d0904-f178-4ce9-956b-d9bfa4902745, 2010.

[12] SinelaboreRT, www.sinelabore.com/doku.php, 2015.

[13] LLC, TestOptimal, testoptimal.com, 2015.

Pair-wise Component Compatibility Testing

Anvika

Dept. of Computer Science
State University of New York, Korea
119 Songdo-Moonhwa-ro, Incheon, South Korea
anvika.anvika@stonybrook.edu

Ilchul Yoon

Dept. of Computer Science
State University of New York, Korea
119 Songdo-Moonhwa-ro, Incheon, South Korea
icyoon@sunykorea.ac.kr

Abstract: The concept of reusability has attributed to modern software systems, which are built with multiple third-party off-the-shelf components, each of which can have different versions. These interdependencies are complex and subject to changes in small time intervals, which creates a huge combination space of all the components involved, thereby making it infeasible to test all potential configurations of components. In the previous work we have modelled the entire configuration space of a multi-component based software system and tested all direct inter-component dependencies by sampling and testing in parallel.

In this paper we propose to extend the configuration sampling strategy to not just the direct dependencies but also include the indirect dependencies, while keeping the number of sampled configurations small, since indirect dependencies can affect the functionality of components. In the new strategy, we sample configurations to test pair-wise combinations of components under direct and indirect dependencies. We believe that the approach will be useful to reveal compatibility bugs that exist between specific component versions in a configuration.

Keywords: pair-wise testing, compatibility, component

1. Introduction

Testing component-based software system is a complex and time consuming task. Particularly, it is challenging to test the compatibility of the software on diverse environments that can consist of different versions of software components, including multiple compilers and third-party components (libraries or tools). That is, the build and execution environment are extremely heterogeneous. Each possible combination of components and their versions is a configuration that might contain unique errors [1]. Restricting the user environments (configurations) to a limited set of tested environments and releasing the software with undetected compatibility bugs, can degrade overall user experience.

Compatibility testing is performed to ensure that a software system can build and function properly in heterogeneous environments [2]. However, the large configuration space leads developers to test only a few popular configurations, leaving bugs to escape to the field. In our previous work [1] we modelled the entire configuration space of a software system and developed a system called Rachet to produce and test configurations that cover all direct dependencies between components. Given a set of component versions, their dependencies and a set of constraints set, Rachet can automatically generate the configuration space and test whether components can be built correctly over all configurations on which the components are designed to run. This approach was also extended to incrementally test components [3].

Testing direct dependencies could reduce the number of test cases drastically when compared to exhaustive testing. However, testing direct dependencies are not enough to test the correct behavior of components over diverse configurations, since the behavior can also be affected by indirect dependencies, especially when the components are reused by binding them dynamically at run time. If a compatibility bug exists between two components under an indirect dependency in a configuration, it cannot be detected by our previous work.

Pairwise testing has laid its basis on the fact that many faults are not due to complex configurations, but due to simple pair-wise interactions. It has the advantage of reducing the number of tests to be executed, because pair-wise bugs represent majority of combinatorial bugs. In this paper, we present an algorithm that samples configurations that test all pair-wise relationships between components under direct and also indirect dependencies, so as to validate both the correct build and functional behavior of components on user configurations, under the

assumption that compatibility bugs arise in many cases from mismatch between two components deployed in a configuration.

The rest of the paper is organized as follows. We give a brief description on the pair-wise test case generation strategy in Section 2 and then explain the algorithm to sample configurations in Section 3. The last section compares the number of configurations produced incrementally and exhaustively.

2. Background: Pair-Wise Test Generation

Pair-wise test case generation is a combinatorial testing method in which each pair of input parameters to a system is tested for all the possible discrete combinations of those parameters. It selects a subset of test cases that covers all possible pairs of combinations with the constraints of reducing the number of total test cases to be run, increasing the test effectiveness by spreading the range of coverage, while still maintaining a small set of test cases [4]. It has the advantage of reducing the number of tests to be executed, because pairwise bugs represent majority of combinatorial bugs [1].

There are several studies, both in computational and algebraic way[4], for pairwise test case generation, like Automatic Efficient Test Generator (AETG) [5][6] and IPO (In-Parameter-Order) [7][8]. The IPO algorithm has lesser space and time complexity, is deterministic in nature and gives comparable results to the other methods. This is because it generates test set's one column (parameter) at a time. The ATEG algorithm is commercial and needs an expensive license if used [11]. The nature of IPO, combined with the above advantages explains its utility for this research problem and in this paper we modified the IPO algorithm to generate configurations that test all pair-wise relationships between component versions under dependencies. For a system with two or more input parameters, the IPO first generates the pairwise test set for the first two parameters and extends the test set to the other additional parameters. The process consists of two steps: (1) Horizontal growth and (2) Vertical growth. We briefly explain the steps with an example component-based system model given in Figure 1. Figure 1 shows that component A requires component D and one of either B or C, as a direct prerequisite for building A. Similarly, D requires F and F requires G.

Horizontal Growth extends test cases by adding a new parameter to the existing test set. For example: Component A has version A_1 and Component B has three versions B_1, B_2, and B_3. From the versions, we can consider (A_1, B_1), (A_1, B_2) and (A_1, B_3) as a pair-wise test set for A and B. If we extend this test set by adding component C that has C_1 and C_2 as its versions, we get (A_1, B_1, C_1), (A_2, B_2, C_2) respectively. But there are several uncovered pairs {(B_1, C_2), (B_2, C_1), (B_3, C_1) and (B_3, C_2)}.

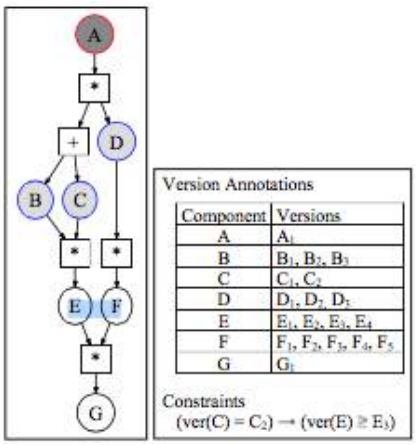


Figure 1 - An Example Component Dependency Model

Vertical Growth adds new tests to cover pairs not covered by horizontal growth. In the running example, to cover (B_1, C_2) we add a configuration (_, B_1, C_2) and then set a version for the component A, which gives (A_1, B_1, C_2) because the component A has only one version in the model. Similarly, for the pairs (B_2, C_1), (B_3, C_1) and (B_3, C_2) we can generate (A_1, B_2, C_1), (A_1, B_3, C_1) and (A_1, B_3, C_2) as the other tests. Thus, a total of 6 test sets are generated to cover all the pairs for components A, B and C. Although there is not an exemplary difference when compared to the 11 test sets generated by exhaustive testing, but it is seen that for a system with n components each having v versions the size of test set grows logarithmically for n and quadratically for v. This means that the difference would increase exponentially as the number of nodes or components to be tested increase. It is noteworthy that the minimum number of configurations are at least greater than the maximum number of version pairs between two components under dependency. In the example above, the number of configurations are greater or equal to 6, which is the version pairs between B and C.

3. Configuration Sampling Method

We describe in this section a method to generate configurations that test all version pairs of components under dependencies, given a software model represented as a component dependency model, which is an acyclic graph, as shown in Figure 1. Specifically, we apply a customized IPO algorithm to generate

configurations. In the method, the component dependency model is traversed either horizontally, vertically or both. Horizontal traversal is to cover direct dependencies between components. We horizontally traverse a graph and find all the direct dependencies that have to be considered to test the correct build of a component over a set of prerequisite components. Vertical traversal is to cover indirect dependencies that have to be tested for ensuring the correct functionality of a component in a configuration that contains other components under indirect dependencies. We traverse each directed path from each node(s) without any incoming edge to all node(s) without any outgoing edge.

In the algorithm 1 shown in figure 2, *rnode* is a relation node which defines the relationships between components, *path* is a Boolean variable that determines whether there is a directed path between two components, and *dpath* determines if two components are under direct dependency. π is the variable that represents the set of version pairs between different components and their versions (e.g., there are 4 pairs between A and B in the running example).

We apply the IPO strategy to traverse the dependency graph, starting from the root node of the graph. While traversing the graph in depth-first we check whether the relation node between components is AND or XOR.

```
if (rnode = "*") then
    if (path(n,p) && (!dpath(n , p))) then
        pairwise(n , p , π)
    end
    else
        do not include all pairwise combinations in π
    end
end
else if ( rnode = "+") then
    pairwise(n , p , π)
end
if (dpath(n , d)) then
    pairwise(n , d , π)
end
else
    do not include all pairwise combinations in π
end
```

Figure 2 – Configuration Sampling Algorithm

If the *rnode* is an AND, we check whether there is a directed path between components to be tested. In Figure 2, *n* is a component close to the root node and *p* is a set of components indirectly required for building *n* – that is, this is to test vertical relationships between components in the graph. If the condition is true, all pairwise combinations between the versions of these components are computed by the *pairwise* function and are kept in π. If the relation node is an XOR, any of the components can be used to build the dependent component as described earlier, but the pairwise testing needs to be applied with every XOR related component and the dependent component -- i.e., in the running example, A needs to be tested with both B and C, since one user might build A with B in the user machine and the other might build A with C. This is repeated to include every component in the dependency model. Next, we produce version pairs to cover all the direct dependencies between components – it tests horizontal relationships between components.

Horizontal coverage information is computed after the vertical coverage as many version pairs in the horizontal coverage are subsumed in the vertical coverage, reducing the number of test cases to be generated. The direct dependency coverage involves checking if the components have a direct path between them – i.e., there is a path that has no other component between them and are just connected by relation nodes [1]. Other components which do not have a direct dependency relation are not included in the pairwise test set, i.e. not all the version combinations are included in the set of version pairs.

```
//Horizontal Growth(T, a_i)
if (|T| ≤ q) then
    for (1 ≤ j ≤ |T|) extend the j^th test by adding values of v_j
end
else
    for ( 1 ≤ j ≤ q) extend the j^th test in T by adding values of v_j
    for(q < j ≤ |T|) extend the j^th test in T by adding one value of a_i
    such that it covers maximum pairs in π.
end
// Vertical Growth(T, π)
while (T does not cover all pairs between a_i and each of a_1, a_2, ··· , a_{i-1})
do
    for each pair in π
    if (T' has a pair which contains " -" as the value of a_k and v as the
    value of a_i) then
        Use w to replace "-"
    end
    else
        Add a new test to T' where w has a value for a_k and v for a_i and
        "-" as other parameter values
    end
    T= T U T'
end
```

Figure 3 - Horizontal and Vertical Growth

The graph is traversed breadth-wise, starting from the root node, as it helps cover all the nodes and maximize the coverage with a small set of configurations. This is achieved by adding nodes to a queue and generate configurations incrementally by adding components in the order in the queue. In the horizontal growth shown in Figure 3, T is the configuration set under construction, a_1, a_2, …, a_{i-1} are components added in the configuration, q is the total number of versions of the component a and v_1, v_2, …, v_q are the versions of the component a. T' is an empty set considered for the vertical growth.

The horizontal coverage as mentioned above adds new component, the first "if" condition determines the case where the new node to be added has fewer versions than the number of configuration in *T* -- e.g., in the running example, there must be 4 configurations that cover version pairs between A and B, and when we add C we just need to use three of the configurations for three versions of the component C. If the number of versions of newly added component is greater than the number of configurations, we perform the vertical growth to cover the version pairs not included in *T* during the horizontal growth.

At each stage of the growth, it is important to find a configuration that covers the maximum number of uncovered version pairs -- i.e. before expanding a configuration set with the new component version, the effect it has on the number of uncovered pairs should be measured. The one which covers the maximum of the new pairs should be chosen. Since this is quite significant to keep the generated configurations low and also it is not computationally feasible to generate each such configuration and choose an optimal one, a greedy approach can be considered.

5. Conclusion

The success of the pair-wise test case generation technique is based on the hypothesis that most defects in combinatorial testing are either single-mode defects (the component under test simply does not work) or double-mode defects (the pairing of two components fails even though all others perform correctly) [9]. This characteristic fits well for the component compatibility testing problem. In this paper we presented our initial effort for systematically generating configurations that test all pair-wise combinations between components under direct and indirect dependencies. We plan to further revise the algorithm and evaluate its benefit by conducting experiments with real-world software systems.

Acknowledgement

This research was supported by the National Research Foundation of Korea (NRF-2013010695).

References

[1] Ilchul Yoon, Alan Sussman, Atif Memon, and Adam Porter, "Effective and Scalable Software Compatibility Testing", International Symposium on Software Testing and Analysis, pp. 63-74, Jul. 2008.

[2] Maity, S., Nayak, A., Zaman, M., Srivastava, A., and Goel, N., "An Efficient Test Generating Algorithm for Pair-wise Testing", In Proc. of the 14th Int. Symposium on Software Reliability Engineering, 2003.

[3] Ilchul Yoon, Alan Sussman, Atif Memon, and Adam Porter, "Towards Incremental Software Compatibility Testing", International ACM SIGSOFT Symposium on Component Based Software Engineering, pp. 119-128, Jun. 2011.

[4] Cassiano B. A. L. Monteiro, Luiz Alberto Vieira Dias, and Adilson Marques da Cunha, "A case study on pairwise testing applications", In Proc. of the 11th International Conference on Information Technology, 2014.

[5] Mats Grindal, Birgitta Lindstrom, Jeff Offutt, and Sten F. Andler, "An Evaluation of Combination Strategies for test case selection", Empirical software Journal, vol. 11, no. 4, pp. 583-611 , 2006 .

[6] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design", IEEE Transactions On Software Engineering, volume 23, 1997.

[7] Kuo-Chung Tai and Yu Lei, "A test generation strategy for pairwise testing", IEEE transactions on Software Engineering, 28(1), 2002.

[8] Y. Cui, L. Li, and S. Yao, "A New strategy for pairwise test case generation", In Proc. of the 3rd International Symposium on Intelligent Information Technology Application, 2009.

[9] R. Kuhn, Y. Lei and R. Kacker, "Practical combinatorial testing: Beyond pairwise", IEEE IT Professional, vol. 10, no. 3 , pp.19 -23 , 2008.

[10] James Bach and Patrick J. Schroeder, "Pairwise Testing: A Best Practice That Isn't", PNSQC, 2004.

[11] Soumen Maity and Amiya Nayak, "Improved test case generation algorithms for pairwise testing", In Proc. of the 16th IEEE International Symposium on Software Reliability Engineering, 2005.

안드로이드 애플리케이션 무반응 탐색과 회피 방안

김경민 0 최은만

동국대학교 컴퓨터공학과
서울 중구 필동로 1 길 30
kmkgmn@naver.com, emchoi@dongguk.edu

요약: 안드로이드 애플리케이션의 UI 테스팅에서 무반응 문제의 탐색과 제거는 매우 중요하다. ANR(Application Not Responding)은 애플리케이션이 일정 시간 동안 응답이 없는 상태로, 내부적으로 어떤 스레드의 처리에 많은 시간이 소요될 경우 UI 스레드가 반응을 하지 않는 상황을 말한다. 안드로이드 플랫폼은 ANR 문제점 해결을 위해 AsyncTask 클래스를 지원한다. 하지만 AsyncTask 또한 무한루프가 발생하거나 액티비티와 상속문제가 발생하는 문제점이 있다. 이 논문에서는 Logcat 과 DDMS(Dalvik Debug Monitor Service) Method Profiling 을 사용하여 ANR 이 발생하는 지점 또는 발생 가능한 코드를 탐색한다. 그리고 기존의 AsyncTask 에서 핸들러를 조합하여 액티비티 라이프사이클을 구분하는 메소드를 만들어 비동기 프로그래밍으로 리팩토링 하는 방안을 제안한다.

핵심어: ANR, 안드로이드, 스레드, AsyncTask, 핸들러 DDMS

1. 서론

매년 수십억 개의 애플리케이션들이 스마트 폰, 태블릿 PC 등에서 사용되고 있고[1], 그 중에서 가장 많이 사용되는 플랫폼은 안드로이드다. 안드로이드 스마트폰을 사용하다 보면 종종 ANR(Application Not Responding)이라는 메시지와 함께 팝업 창이 뜬다. 이 팝업 창은 일반적인 오류로 인해 발생하는 Sorry 팝업과는 다르다. Sorry 팝업은 흔히 프로그램에서 예외상황(Exception)이 발생하여 프로세스가 소멸되면서 뜨는 팝업이고, 닫기 버튼 하나만 존재한다. 이는 명백하게 프로그램 오류이고 로그를 이용하여 쉽게 원인을 파악할 수 있다. 하지만 ANR 오류, 즉 UI 스레드가 응답할 수 없는 상황이 발생할 경우는 어떤 프로세스가 응답을 받지 않아 대기중인 상태이다. 예를 들어 UI 스레드에서 파일을 읽고 저장하거나 네트워크로 데이터를 전송, 수신하는 등의 느린 작업을 처리하는 경우에 발생한다. 보통 안드로이드에서 정한 제약조건은 애플리케이션이 입력에 대해 5 초안에 반응하지 않는 경우 ANR 팝업이 발생하게 되고 해당 애플리케이션을 강제로 종료시킨다[2].

이런 오류를 피하고 애플리케이션이 부드러운 응답성을 유지하기 위해 무겁고 시간이 많이 드는 작업은 Main 스레드에서 Worker 스레드로 옮기는 것이 좋다.

이 논문의 목적은 ANR 이 발생하는 지점을 찾고 수정하여 프로세스가 부드러운 응답성을 유지할 수 있도록 하는 것이다. 이를 위하여 ANR 이 발생한 곳을 찾아서 수정하거나 발생 가능한 지점을 탐색하여 미리 예방하는 것이 필요하다. 그러기 위해 onCreate() 와 onResume() 과 같은 주요 메소드들은 최소한의 작업만 하도록 관리한다. 그리고 네트워크 또는 데이터베이스 작업, 비트맵 연산과 같이 잠재적으로 오랜 시간 동안 실행되는 기능들이나, 계산이 많이 요구되는 작업들은 백그라운드에서 Worker 스레드를 이용하여 처리한다

이 논문은 총 다섯 장으로 구성이 되어있다. 2 장에서는 ANR 의 발생하는 기본적인 개념과 현재 안드로이드 플랫폼에서 사용되고 있는 AsyncTask 방법들과 문제점을 설명하고 UI 스레드와 Worker 스레드를 분리하는 방법에 대해서 살펴본다. 3 장에서는 ANR 이 발생하는 지점을 DDMS 를 사용하여 탐색하는 방법을 설명하고 4 장에서는 ANR 이 발생한 지점에 대해 핸들러를 이용한 리팩토링 방법과 Worker 스레드를 통한 스케쥴링 방법을 제안한다. 마지막으로 5 장에 결론을 맺는다.

2. 기본 개념

모바일 장비는 다양한 환경에서 사용이 된다. 상황에 따라 최대한 신속하게 반응해야 하므로 성능(Performance)보다 더 중요시 해야 할 것이 있는데 바로 반응성(Responsiveness)이다. 안드로이드에서도 반응성을 최우선으로 중요하게 생각하여 항상 백그라운드에서 프로그램의 반응성을 체크한다.

반응성을 체크하는 중에 5초 이상 반응하지 않는 상황이 발생하면 프로그램에 반응이 없는 상태로 간주하여 사용자에게 즉각 알려준다. 사용자가 기다리는 불편함을 방지하기 위해 OS가 직접 개입하는 것이다. 보통 OS가 개입하는 경우가 2가지가 있는데 첫 번째는 애플리케이션이 5초 이상 사용자의 입력에 반응하지 않을 때, 두 번째로 브로드캐스트 리시버가 10초 내로 리턴 하지 않을 때 발생한다. 이런 상황이 발생하면 시스템은 ANR 팝업 띄우고 이 팝업을 통하여 프로그램을 강제 종료할 수 있다. 안드로이드에서 ANR이 발생하는 조건은 2가지 이외에도 존재한다. 예를 들어 ActivityManager-Service 클래스가 ANR이 발생하는 타임아웃을 선언하는 경우에도 존재한다.

위 제약에 따라 ANR 피하려면 모든 작업을 5초 이내로 끝내야 한다. 하지만 아주 간단한 프로그램이 아닌 이상 현실적으로 불가능하기 때문에 일반적인 해법으로 시간이 걸리는 작업은 스레드를 분할하여 진행한다. 또는 비동기적인 호출을 통하여 내부에서 스레드를 돌리는 것이다.

안드로이드 프로그램은 기본적으로 Single 스레드 구조이고 작업시간이 5초를 초과하는 작업은 무조건 스레드로 넘긴다. 여기서 확실히 5초를 초과하는 작업뿐만 아니라 5초를 넘길 가능성이 있는 모든 작업이 대상이다.

액티비티, 서비스, 브로드캐스트 리시버 등 안드로이드의 모든 애플리케이션 컴포넌트들은 main 스레드에서 시작된다. 따라서 이들 컴포넌트가 시간이 많이 드는 처리를 수행한다면, 서비스와 화면에 보이는 액티비티를 포함한 다른 모든 컴포넌트들이 블록이 되고 더 이상의 작업을 수행할 수 없게 된다.

애플리케이션 자체 내에서 입력 값(input event) 또는 Intent 들을 처리 할 수 있는 기능을 주지 않기 때문에 UI 스레드에서 실행되는 작업 중에 시간이 오래 걸리는 작업들에서 ANR 팝업이 뜬다. 따라서 UI 스레드에서 실행되는 어떤 메소드라도 그 스레드에서 최소한의 작업만 수행해야 한다.

하지만 이 Task들을 처리하기 전에 이 스레드들을 분할하는 방법이 필요하다. 스레드를 분할하기 위해 안드로이드에서 제공되는 Strict Mode API를 이용하여 분할한다. Strict Mode API는 특정 스레드에서 일어날 수 없는 일들을 규약으로 정할 수 있다. 이를 이용하여 Main 스레드에서 네트워크 접근이나 대량의 연산 같은 비효율적인 작업을 하는 것을 감지하여 알려주고 프로그램이 부드럽게 작동할 수 있게 도와줄 뿐만 아니라 빠른 응답을 할 수 있게 해준다. 그리고 PenaltyLog() 메소드를 이용하여 Logcat에 출력이 가능하기 때문에 찾아낸 문제점을 Handler를 통하여 리팩토링을 진행한다. 또 스레드를 새로 생성할 때 마다 고유의 ID를 생성하여 고유 스레드 생성을 할 수 있고 작업에 특정 우선순위를 지정해야 될 경우, 실행 시간과 오래 걸리고 많은 자원을 사용하여 다른 작업을 차단해야 되는 경우 등을 생각하여 따로 관리 할 수 있다.

2.1 기존의 ANR 회피방법

ANR을 회피하기 위한 방법은 여러 가지가 있다. 첫 번째로 안드로이드 API에서 지원하는 비동기 클래스인 AsyncTask를 사용한다.

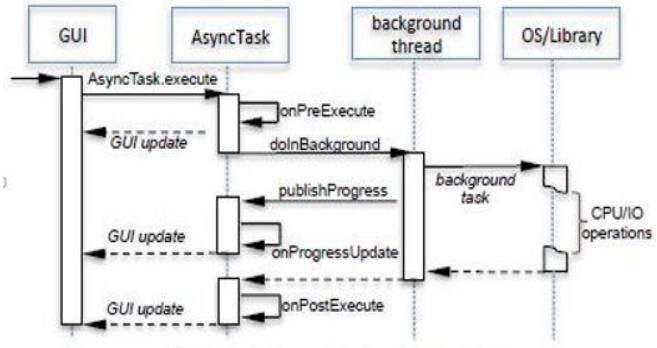


그림 1 AsyncTask 실행구조

그림 1은 AsyncTask의 실행구조를 보여준다. AsyncTask를 시작하면 가장 먼저 onPreExecute 함수가 실행된다. 그러면 AsyncTask 클래스에서 doInBackground 함수가 실행된다. 이 함수는 오래 걸리는 작업을 처리힌다.

만일 doinBackground 함수 처리 도중에 UI에 사용자 작업이 필요하다면 PublishProgress 함수를 호출한다. 그러면 호출이 일어날 때마다 onProgress-update 함수가 실행된다. onProgressupdate 함수는 doInBackground 함수의 진행 용도로 사용된다.

만일 doInBackground 처리하는 도중에 AsyncTask

를 중단하게 되면 onCancelled 함수가 실행되고 정상적으로 doInBackground 처리가 완료되면 onPostExecute 함수가 실행되고 스레드가 종료됨을 알리는 구조로 되어있다.

두 번째로 기존의 스레드 클래스나 핸들러 스레드 클래스를 사용하기 원한다면 스레드의 우선순위를 변경하여 사용한다.

우선순위를 변경하는 방법은 Process.setThread-Priority()에 상수를 전달하면 된다. UI 스레드에서 클래스나 핸들러 스레드 클래스를 사용하면 이 스레드의 우선순위가 main 스레드와 같아지기 때문이다.

2.2 스레드 분할 방법

스레드를 분할하기 위해서 안드로이드에서는 Struct Mode 라는 클래스를 사용한다. 각 액티비티의 UI 메인 스레드의 빠른 응답성을 위해 고안된 디버깅 Logic 으로 메인 스레드에 특정한 규약을 정하여 이를 위반하는 것에 대한 특정 조치(Panalty)를 부여할 수 있다. 스레드 규약의 종류로 Disk Read/Write, Network Usage 가 기본적으로 존재하며, 사용자 임의로 규약을 정하여 위반 사항을 찾을 수 있다. 규약의 종류로는 액티비티 Leak, Cursor Leak, Object Leak 이 존재하고 특정조치에 대한 행동의 종류로는 Log, Dialog, Dropbox, Death, Surface Show 등이 있어 다양한 방법으로 위반 사항을 표시할 수 있다. Strict Mode 를 통해 찾아낸 문제점은 비동기 프로그래밍을 통해서 해결이 가능하다. 한마디로 StrictMode 코드를 수정해 주는 것이 아니라 정한 규약을 알리는 기능을 함으로써 스레드의 기능을 구분하고 분할 할 수 있게 도와준다.

2.3 기존의 ANR 회피방법 문제점

AsyncTask 도 완벽하지는 않다. AsyncTask 는 액티비티의 라이프 사이클을 따르지 않기 때문에 액티비티가 종료되어도 여전히 실행 중인 상태이다. 이로 인해 애플리케이션의 충돌이 발생할 수가 있다.

그리고 버전 별 차이로 인해 오류가 발행할 수 있다. Android 1.5 버전에서는 순사실행이 적용되었으나 1.6 버전에서는 병렬실행을 지원하고 다시 4.0 버전에서부터 순차실행을 지원하고 executeOnExecutor() 를 통해 병렬실행 방식을 지원하고 있다[3].

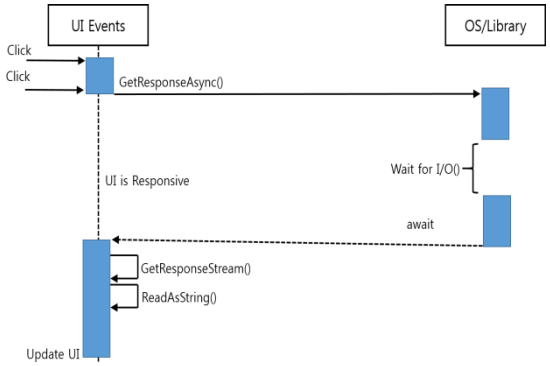


그림 2 AsyncTask 실행 문제

그림 2 처럼 만약 A, B 2 개의 AsyncTask 가 있고 한번에 호출을 한다. 간단한 걸 조회하는 A Task 가 있고, 복잡한 걸 조회하는 B Task 가 있다. A 를 먼저 호출하면 A 가 먼저 돌아올 거라고 가정하고 코드를 작성하면 문제가 발생할 수 있다. 호출하는 순서대로 결과가 돌아오지 않는 경우가 될 때 대기 상태가 발생할 수 있다.[4], [5]

3. ANR 탐색

3.1 Logcat 을 이용한 탐색

대부분의 에러 발생시에는 그림 3 과 같이 Logcat 을 이용하여 Buffer 에 있는 log message 들을 보여준다. 강제 종료가 되어도 Buffer 에 log 가 남아있기 때문에 어디서 문제가 있는지 찾을 수 있다.

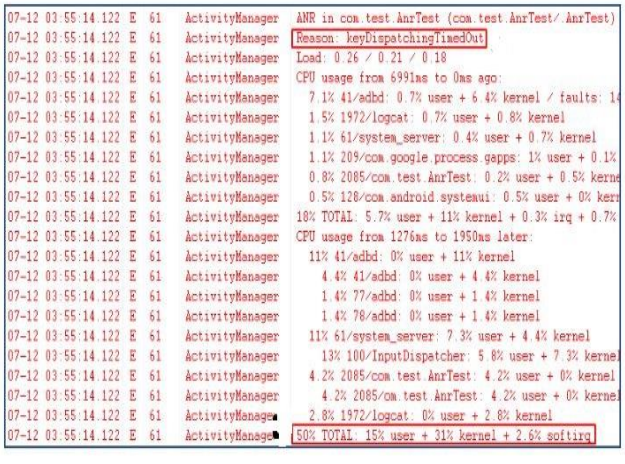


그림 3 Logcat

그림에서의 ANR 발생 원인은 keyDispatchingTimed-Out 이다. 즉 사용자가 버튼을 누르는 동작을 취했는데 5 초 동안 반응이 없었다는 것이다. Activity 에서

ANR 이 발생한 경우의 대부분은 바로 keyDispatchingTimedOut 이다.

그리고 그림 3 에서 표시된 2 번을 보면 CPU 점유율은 50%인데 50% 밖에 CPU 가 동작하지 않는 상황에 ANR 이 발생했다는 것은 현재 Activity 가 문제 있다고 판단하고 수정해야 한다. 만약 100%의 점유율에서 ANR 이 발생했다면 Background 로 돌아가는 다른 Process 의 부하가 커서 현재 Activity 를 지연시켰다고도 생각할 수 있다. 그러므로 현재 상태의 CPU 점유율은 꼭 확인하고 어떤 Process 가 점유율이 높은지 확인해야 한다.

3.2 DDMS 를 이용한 디버깅

DDMS(Dalvik Debug Monitor Service)에서 Method Profiling 을 이용한 방법이다. 애플리케이션 디버깅을 수행하여 얻을 수 있는 CPU 점유율 반복 호출된 횟수를 통해 ANR 이 발생할 가능성이 높은 곳을 탐색할 수 있다.

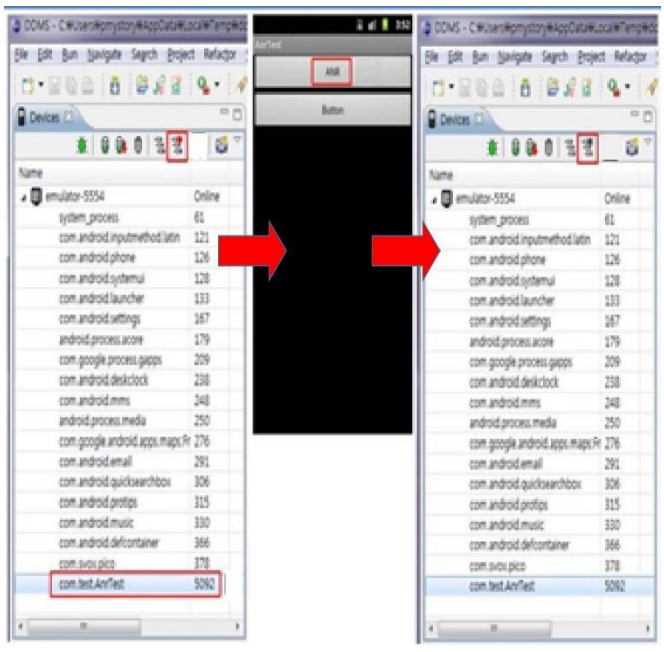


그림 4 DDMS Method Profiling 실행 과정

그림 4 와 같이 디버깅할 프로세스를 선택하여 Method Profiling 을 진행한다. 처리하면 아래와 같이 호출된 함수 목록이 화면에 뜨게 된다.

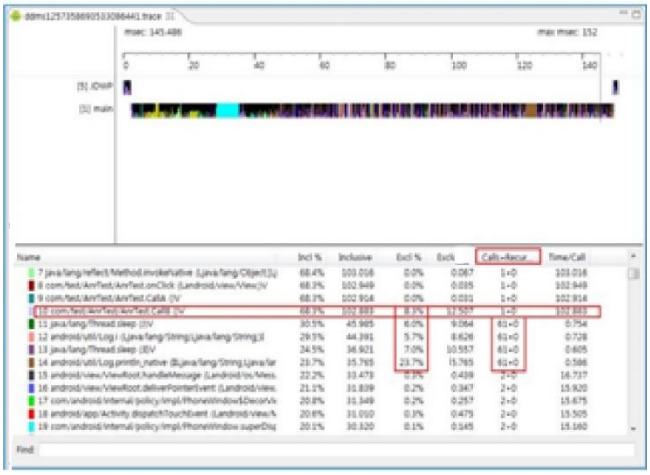


그림 5 DDMS 로그

그림 5 의 1,2 번을 통하여 호출된 수를 보면 알 수 있듯이 61 번이 호출된 그룹이 있다. 3 번 Call Stack 을 보면 Call B()함수로 호출된 이후로 반복된 것을 알 수 있다. 4 번을 보면 CPU 를 소모한 그룹 또한 2 번과 비슷하다. 그 그룹을 정리하면 sleep 와 Log.i 이다.

4 해결방법

4.1 AsyncTask 의 액티비티 상속

안드로이드는 응답시간을 초과하면 스레드가 중지된다. 문제는 액티비티에서 응답시간을 초과하면 다이얼로그 창이 나오면서 표시를 해주지만 다른 클래스에서 응답시간이 초과하면 아무 반응 없이 프로그램이 종료되게 된다. 단순히 스레드를 이용하여 이 문제점을 해결하는 것은 불가능하다. 스레드에서 처리하면 이론상으로는 백그라운드에서 실행되지만, 종료 시점이 맞물리게 되면 추가적인 문제가 발생할 수 있다. 그래서 AsyncTask 를 이용해야 되는데 문제는 AsyncTask 는 반드시 Subclass 에서 상속을 해야만 사용할 수 있기 때문에 이 문제를 해결하기 위해 AsyncTask 와 핸들러를 조합해서 사용해야 한다.

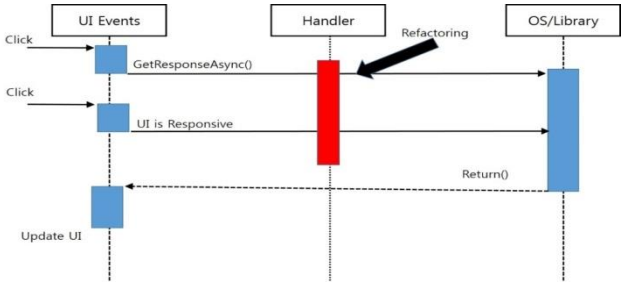


그림 6 Handler 를 통한 리팩토링

그리고 액티비티 라이프 사이클을 구분할 수 있는 함수를 만들어서 AsyncTask 의 주요한 세가지 동작 (onPreExecute(),doInBackground(),onPostExecute()) 을 개선해야 한다. 여러 스레드에서 단일 개체의 메서드를 호출할 경우, 호출에 동기화 적용이 필요한 영역(인스턴스 메서드 및 정적 메서드)에만 동기화를 수행한다. 만약 예외처리를 해야 될 경우에는 블록을 해제하고 마지막에 Finish()등을 통하여 종료한다

4.2 Worker 스레드를 통한 스케쥴링

실행 순서를 스케쥴링하는 방안을 적용할 수도 있다. 안드로이드는 복수의 스레드가 하나의 UI 위젯에 동시에 접근해서 일어날 수 있는 잠재적인 문제를 해결하기 위해 다음과 같은 절차로 Work 스레드가 UI 위젯에 접근하게 애플리케이션을 구성한다.

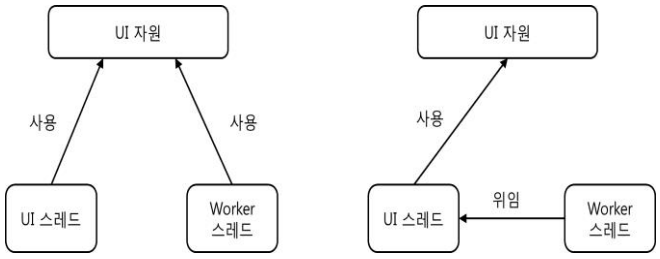


그림 7 worker 스레드

그림 7 처럼 UI 스레드와 Worker 스레드가 동시에 명령을 내림으로 혼선이 유발된다.

Worker 스레드는 UI 스레드를 통해서만 UI 자원을 제어 가능하게 한다. 문제는 싱글 스레드 이상의 효율을 내지 못한다는 것이다. 로직 스레드의 요청을 큐에서 꺼내서 처리하는데, 이 큐에 쌓이는 속도가, 데이터를 처리하는 속도보다 오래 걸린다면 전체적인 처리속도가 계속 늦어져 결국엔 사실상 아무 일도 못하는 상태가 될 된다. 그렇기 때문에 로직을 멀티 스레드로 분리하는 작업이 필요하다.

로직을 멀티 스레드로 처리하고 스레드끼리 중복된 데이터를 사용할 일이 있을 때, 동기화 객체를 사용해서 관리하고, 비동기 작업은 별도 스레드에서 처리하는 방식이다. 하지만 이 방법 또한 로직을 멀티 스레드로 하면서 겹치지 않도록 해야 한다. 로직을 멀티 스레드로 처리하고, 이 상황에서 스레드끼리 겹칠만한 일들과 비동기 처리를 별도 스레드에 맡기고, 로직 스레드에서는 다른 스레드와 겹치지 않는 일을 함으로써 Lock-Free 한 상태로 스레드를 관리한다. 그리고 로직 스레드에서는 자신에게 주어진 데이터에만 접근하게 설정한다.

5. 결 론

ANR 문제를 해결하기 위해서 고안된 AsyncTask 는 액티비티의 라이프 사이클을 따르지 않는다. 즉 액티비티가 종료 되어도 Asynctask 는 종료되지 않고 여전히 실행이 되어 이로 인해 애플리케이션의 충돌이 발생하였다. 그리고 버전 차이로 인한 문제점도 있기 때문에 핸들러를 통한 리팩토링 방안을 생각해 보았다. 그리고 Worker 스레드를 통한 순차적 방식으로 문제점을 해결해보고자 하였다. 하지만 스레드 간의 데이터가 쌓이는 속도나 CPU 의 환경에 따라 ANR 이 발생할 수 있다. 이 부분에 대한 연구가 필요하다.

참고문헌

[1] Dany Dig, “Refactoring for Asynchronous Execution on Mobile Devices,” IEEE Software, 0740-7459, November 2015.pp 52-61

[2] “Keeping Your App Responsive”, http://developer.android.com/training/articles/perf-anr.html

[3] “The dark side of AsyncTask”, May 2015, http://bon-app-etit.blogspot.com/2013/04/the dark-side-of-asynctask.html.

[4] Yu-Lin, Semih Okur, Danny Dig, “Study and Refactoring of Android Asynchronous Programming,” 30th IEEE/ACM International Conference on Automated Software Engineering ASE’ Nov, 2015. pp 224-235

[5] Yu-Lin, Danny Dig, “Refactoring for Android Asynchronous Programming,” 30th IEEE/ACM International Conference on Automated Software Engineering ASE’ Nov, 2015. pp 836-841

다중 결함 환경에서의 SFL 성능 개선을 위한 테스트 케이스 군집화 및 가중치 부여 기법

이재희, 김정호, 이은석

성균관대학교 전자전기컴퓨터공학과
경기도 수원시 장안구 서부로 2066
{ljhsoft, jeonghodot, leees}@skku.edu

요약: 결함 위치 식별(fault localization)은 디버깅 과정에서 가장 많은 시간을 소요 하는 것으로 알려져 있고, 이를 자동화 하기 위한 많은 연구가 수행되고 있다. 그러나 대부분의 기존 연구들은 단일 결함(single fault) 환경을 가정하고 있어 다중 결함(multiple fault)이 존재하는 실제 환경에 적용하기에는 많은 제약이 따르고 있다. 병렬 디버깅 기법(parallel debugging)은 기존에 제안된 다중 결함 식별 기법 중 가장 높은 성능을 보이는 기법 중 하나로 알려져 있으나, 기존 연구에서 제시된 군집화(clustering) 기법은 결함 간 경합(bug race) 발생하고 및 군집 내 다중 결함 존재로 인한 비효율이 발생하고 있다. 본 논문에서는 실패한 테스트 케이스가 단일 또는 다중 결함으로 발생한 것인지에 대한 정보를 추출한 후 이를 군집 정확도(cluster purity) 향상 및 가중치 부여에 사용하여 다중 결함 환경에서의 결함 위치 식별을 보다 효율적으로 수행할 수 있는 기법을 제안한다.

핵심어: 다중 결함 위치 식별, 테스트 케이스 군집화, 가중치 부여, 병렬 디버깅

1. 연구배경

결함 위치 식별(fault localization)은 디버깅 과정 중에서 가장 어렵고 많은 시간을 소요 하는 활동으로[1], 이를 자동화 하기 위한 다양한 연구가 수행되고 있다. 스펙트럼 기반 결함 위치 식별(Spectrum-based fault localization) 기법은 다른 기법들에 비하여 적은 연산 비용 대비 정확한 결과를 도출하는 것으로 알려져 다양한 연구가 진행되고 있는 분야 중 하나이다. 하지만 대부분의 기존 연구들은 단일 결함(single fault) 존재 상황을 가정 후 해당 결함을 효율적으로 식별하는 연구에 집중되어 다중 결함(multiple fault) 이 존재하는 실제 환경에서는 제안된 기법의 효율성이 떨어지는 문제가 발생하고 있다[6]. 이를 해결하기 위해 다양한 연구가 진행되어 다중 결함을 식별 할 수 있는 기법들이 제안 되었고[7][12], 그 중 병렬 디버깅[8]은 다중 결함 환경에서 효율적으로 동작하고, 실제 적용이 타 기법 대비 상대적으로 용이한 것으로 알려져 있다. 그러나 해당 논문에서 제시된 결함 군집화(clustering) 기법은 군집 정확도(clustering purity) 문제로 결함 간 경합(bug race)가 발생하고 군집 내 다중 결함 잔존으로 인한 비효율이 존재하여, 보다 정확한 군집화 및 디버깅 효율화에 대한 필요성이 제기된다. 본 논문에서는 최근 연구가 진행되고 있는 다중 결함 테스트 케이스 판별 기법[9]을 사용하여 군집 정확도를 향상시키고 다중 결함 테스트 케이스에 가중치를 부여하여 다중 결함 환경에서의 결함 위치 식별을 좀 더 효율적으로 수행할 수 있는 기법을 제안한다.

2. 관련연구

2.1 스펙트럼 기반 결함 위치 식별 방법

결함 위치 식별 연구가 중요해지면서 다양한 방식의 관련 연구들이 진행되어 왔다. 이중 스펙트럼 기반 결함 위치 식별 방법은 별도의 모델 구축이나 학습 등의 과정이 필요 없이 테스트 케이스의 실행정보와 구문(statement, method, component)의 커버리지(coverage)만을 이용하여 통계적으로 의심도를 계산하는 방법이기 때문에 다른 기법 대비 상대적으로 적용 비용 및 계산량으로 정확한 결과를 도출하는 것으로 알려져 있다.

Tarantula[2]는 초기에 제안된 스펙트럼 기반 결함 위치 식별 기법으로 각 테스트 케이스 별 구문 커버리지와 테스트 케이스 실행 성공/실패 여부를 이용(a_{ep}, a_{ef}, a_{np}, a_{nf}) 하여 각 구문 별 의심도 (Suspicious Score)를 계산하고 개발자가 어떤 구문부터 검사(inspection) 해야 하는 지를 나타내는 순위 정보(Ranking)를 제공 한다.

$$\frac{\frac{a_{ef}}{a_{ef}+a_{nf}}}{\frac{a_{ef}}{a_{ef}+a_{nf}}+\frac{a_{ep}}{a_{ep}+a_{np}}}$$

수식 1 Tarantula 기법 의심도 계산 식

수식 1 및 그림 1 은 Tarantula 기법의 의심도 계

산 공식 및 동작 원리를 보여준다. 이후 제안된 Ochiai[3]는 생물학 분야에서 사용되는 계수(coeffient)를 결함 위치 식별에 이용한 것으로, 기존에 제안된 기법들 보다 정확한 결과를 도출하는 것으로 알려져 있다. 최근에는 제안된 SLF 기법들의 특성을 분류 및 군집화 하여, 상황 별 최적화된 기법을 제공하는 hybrid 기법 [13] 등의 관련 연구가 활발히 진행되고 있다.

	TC1	TC2	TC3	TC4	TC5	TC6	
void Example (int x, int y){	5,1	9,1	9,0	10,1	1,2	4,7	Tarantula
1: if(x>y){	●	●	●	●	●	●	0.50
2: x = x-2; // x = x+2;	●	●	●	●			0.62
3: printf("x>y");	●	●	●	●			0.62
4: }else{					●	●	0.00
5: y = y+2;}					●	●	0.00
6: if(x<y+6){	●	●	●	●	●	●	0.50
7: printf("x<y+6");}	●				●	●	0.71
} Pass/Fail	F	P	P	P	P	P	

그림 1 스펙트럼 기반 결함 위치 식별 기법 (Tarantula)

2.2 다중 결함 존재 시 결함 영향도 분석

다중 결함이 존재하는 경우에 결함 간의 영향도를 분석하고, 이러한 요소가 기존 제안된 SFL 기법들의 성능에 어느 정도 영향을 주는지에 대한 다양한 연구가 진행되어 왔다.

[4]는 결함간 영향 발생 유형을 분류하여 각 프로젝트 별로 어떠한 양상으로 영향이 나타나는가를 연구하였고, [6]는 해당 연구를 확장하여 결함 간 영향 발생 유형을 독립적 (independent), 증폭 (Synergy), 상쇄 (Obfuscation), 동시발생 (Multiple) 의 4 가지 유형으로 구분한 후 결함의 개수가 증가 할수록 결함간의 영향도가 비례하여 증가하여 결함 해결 비용이 상승함을 실험적으로 증명하였다.

2.3 다중 결함 환경에서의 결함 위치 식별 기법

다중 결함 위치 식별을 위한 다양한 기법들이 제안 되었다. 이를 분류하면 다음과 같이 크게 3 가지 방법으로 나누어 볼 수 있다.

- 순차 디버깅 (Sequential Debugging)
- 병렬 디버깅 (Parallel Debugging)
- 동시 디버깅 (Simultaneous Debugging)

2.3.1 순차 디버깅 (Sequential Debugging)

순차 디버깅은 기존 단일 결함 식별에 사용하던 기법을 그대로 적용하여 한번에 하나씩 결함을 식별하여 수정(Run → Inspect → Fix 과정을 반복적으로 수행) 해나가는 방식이다. 기존 기법을 변경 없이 적용 가능하다는 장점이 있고 다중 결함 존재 시에도 가장 의심도가 높은 결함(Top1)을 발견하는 비용은 결함의 개수가 증가하여도 동일하게 유지된다는 연구[5]에 따라 해당 방법을 사용하여도 단일 결함 해결 자체의 성능에는 큰 문제가 없다. 하지만 디버깅 수행 시 존재하는 결함의 전체 개수를 알 수 없어 디버깅 완료 시점을 판단할 수 없고, 여러 명이 동시에 디버깅을 진행이 필요하거나 특정 결함을 해결하고자 하는 경우에는 해결 방법이 없어 실제 프로젝트에서는 적용하기 어려운 방식이다.

2.3.2 병렬 디버깅 (Parallel Debugging)

병렬 디버깅은 Jones 가 순차 디버깅 방법의 단점을 보완하기 위해 제안 한 방식으로 실패한 테스트 케이스를 동일 결함으로 인해 발생한 것끼리 군집화 하여 각 군집 별로 결함 위치를 식별함으로써 동시에 여러 개의 결함을 병렬로 식별할 수 있는 방법이다[8]. 그림 2 는 순차 디버깅과 병렬 디버깅의 수행 절차를 비교하여 보여준다.

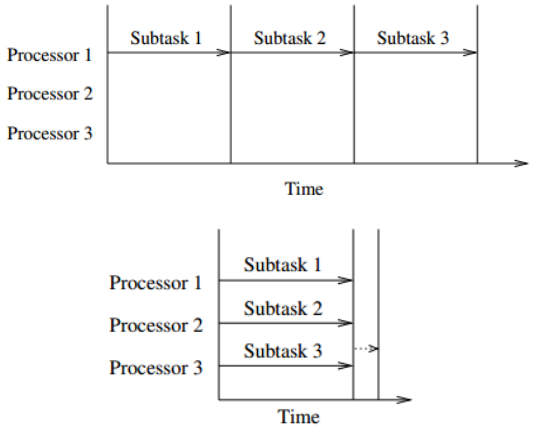


그림 2 순차(상) 및 병렬 디버깅(하) 기법 비교

병렬 디버깅은 동시에 여러 개의 결함 식별이 가능하여 전체적인 디버깅 시간을 감소 시킬 수 있고 결함 간 영향을 완화하여 디버깅 비용을 감소시키며, 특정 결함에 대한 디버깅이 가능하다는 장점이 있다. 하지만 정확하게 군집화가 되지 않은 경우 결함 간 경합(bug race)가 발생하여 디버깅 비용이 증가하고 군집화 내 다중 결함이 잔존하여 오히려 순차 디버깅 대비 비효율이 발생할 수 있다. Wolfgang [10]는 Jones 의 기존 연구를 확장하여 Linear Integer Programming 알고리즘을 활용함으로써 군집간 중복 영역을 제거하여 결함 간 경합 문제를 해결하는 기법을 제안하였으나 주요 실행 정보 및 TC 가 유실(lost)되는 문제가 발생한다.

2.3.3. 동시 디버깅 (Simultaneous Debugging)

동시 디버깅은 프로그램 내 존재하는 여러 개의 결함을 한번에 식별하는 방식으로 최근 점차 관련

연구가 증가하고 있다. 성공적으로 실행되어 높은 정확도 제공 시 적용 비용이 적고 전체 결함 해결 비용이 감소할 수 있으나, 일반적으로 다른 방법들에 비하여 많은 계산 량이 필요하고 구현이 복잡하여 실제 프로젝트 대상 적용은 쉽지 않은 상황이다.

대표적인 동시 디버깅 방식으로는 기존 모델 기반 접근법과 스펙트럼 기반 기법을 접목하여 효율성과 정확성을 동시에 확보하고자 하는 스펙트럼 기반 추론 기법(spectrum based reasoning)[7], 소스코드 레벨에서의 논리적 정합성 검증 방법을 이용한 SAT (Satisfiability) 기반 접근 방법[11], 유전 알고리즘(Genetic Programming)을 이용한 기법[12] 등이 있다.

그림 3 은 순차 디버깅과 동시 디버깅의 수행 절차를 비교하여 보여준다.

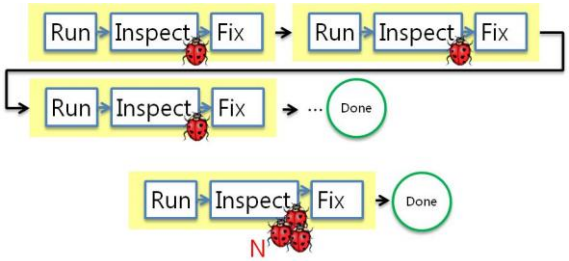


그림 3 순차(상) 및 동시(하) 디버깅 기법 비교

3. 제안 기법

기존 접근 방법 중 실제 프로젝트에 적용 가능성이 높은 것은 병렬 디버깅으로, 기존 기법들은 2.3.2 에서 언급한 문제 이외에도 다중 결함에 대한 가중치를 고려하지 않고 모든 실패한 TC 에 동일한 중요도를 부여하여 진행되었다. 본 논문에서는 이를 개선하기 위하여 실패된 테스트 케이스가 단일 결함 또는 다중 결함으로 발생되었는지 여부를 추출 (실패한 테스트 케이스와 가장 가까운 성공한 테스트 케이스와의 거리(distance) 계산 결과 활용 기법[9] 사용) 후 이를 사용하여 군집 정확도(cluster purity)를 향상시키는 방법(3.1) 및 다중 결함으로 인해 실패된 테스트케이스에 가변 가중치를 부여하는 기법(3.2)을 제안한다. 그림 4 는 제안된 기법의 전체적인 흐름을 보여준다.

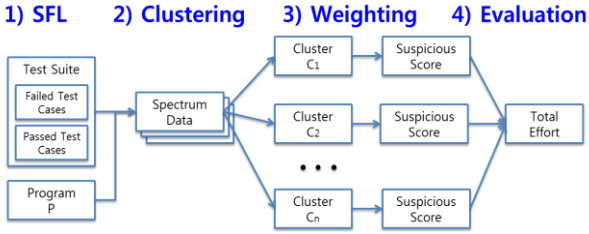


그림 4 결함 기반 테스트 케이스 군집 효율화를 통한 다중결함 식별 절차

3.1 군집 정확도 향상 기법

기존 기법[8]은 테스트 케이스 간 순위 정보의 유사도 (Jaccard Similarity 이용)를 이용하여 군집화를 진행하나, 본 연구에서는 추가로 다중 결함 정보 판별 과정에서 나타나는 잠재 결함 식별자(potential fault indicator)를 이용[9]하여 정확도를 향상 시킨다. 진행 순서는 다음과 같다.

1. 먼저 전체 테스트 케이스 중 실패된 테스트 케이스와 성공한 전체 테스트 케이스를 이용하여 각 실패된 케이스 별 결함 의심도 순위를 계산한다
2. 실패된 테스트 케이스 별 다중 결함 실패 여부를 판별을 진행한다.
3. 2 의 과정에서 추출된 잠재 결함 식별자 정보를 기준으로 같은 식별자를 가지는 테스트 케이스를 동일 군집으로 판단하여 테스트 군집화를 진행하고 군집 정확도를 측정한다.
4. 해당 테스트 케이스 군집과 나머지 성공한 테스트 케이스와 합쳐 각 군집 별 결함 위치 식별을 진행한다.

$$\frac{|t \in C \&\& t \text{ actullay has executed only fault } f|}{|t \in C|}$$

수식 2 군집 정확도(cluster purity)

수식 2 는 해당 군집(C)이 내의 테스트 케이스(t)가 얼마나 동일한 결함(f)로 인한 것인지를 나타내는 군집 정확도(cluster purity) 나타낸다.

3.2 가변 다중치 부여 기법

동일 수의 실행 구문을 가지는 실패한 테스트 케이스를 비교했을 때 단일 결함으로 인한 실패 테스트 케이스 보다 다중 결함으로 인한 실패 테스트 케이스의 단위 실행 구문이 보다 실패할 확률이 높다는 점을 착안하여 해당 테스트 케이스에 가중치를 부여하는 방법으로, 앞서 제안된 다중 결함 여부 추출 과정에서 사용했던 거리(distance)에 따라 가변 다중치(υ)를 부여한다. 각각의 구문 s 에 대한 가중치가 부여된 계산 수식은 다음과 같다.

$N_{ep}(s)$: s 를 지나는 가중치 부여된 성공 TC
$N_{ef}(s)$: s 를 지나는 가중치 부여된 실패 TC
$N_{np}(s)$: s 를 지나지 않는 가중치 부여된 성공 TC
$N_{nf}(s)$: s 를 지나지 않는 가중치 부여된 실패 TC
(성공 테스트 케이스: ω =1, 실패 테스트 케이스: $\omega = \upsilon$)

$$\frac{\frac{N_{ef}}{N_{ef}+N_{nf}}}{\frac{N_{ef}}{N_{ef}+N_{nf}}+\frac{N_{ep}}{N_{ep}+N_{np}}}$$

수식 3 가중치 부여된 Tarantula 의심도 계산 식

수식 3 은 이러한 계산 수식을 이용하여 가중치가 부여된 Tarantula 기법 의심도 계산식을 나타낸다

본 연구의 세부 목표는 다음과 같다.

1. 향상된 군집화 및 가변 가중치 부여 기법을 제안하고 실제 프로젝트 대상 실험을 진행하여 제안된 기법의 활용 가능성을 확인한다.
2. 향상된 군집화 및 가변 가중치 부여 기법 적용으로 군집 정확도 및 다중 결함 위치 식별 비용이 효율화 되었음을 확인한다.

4. 실험 계획

Software Repository Infrastructure 에서 제공 되 space 를 대상으로 실험을 진행 중이다. space 는 다른 subject 대비 큰 LOC(6445)를 가지고, 많은 결함 버전(38 개) 및 테스트 케이스(13585 개)를 제공하여 SFL 및 테스트 관련 연구에 많이 사용되고 있다.

실험 조건은 다른 관련 연구와 비교 가능하도록 제공된 단일 결함 버전을 조합하여 다중 결함을 생성(최대 17 개)하고, 테스트 케이스 또한 커버리지 조건을 만족하는 범위 내에서 랜덤하게 선택(최대 1000 개)하여 실험에 사용할 계획이다.

5. 향후 연구

본 연구에서 제안된 알고리즘 사용 시 기존 방법보다는 군집 정확도가 향상 될 것으로 보이지만, 테스트 케이스의 작성 방식에 따라 이상적인 군집화가 불가능한 경우가 존재할 것으로 예상된다. 이를 위해 향후 테스트 케이스의 정적/동적 실행 경로 슬라이스 기반 기법을 활용하여 다중 결함 발생 테스트 케이스를 복수의 단일 결함 테스트 케이스로 분리하는 연구를 수행 할 계획이다.

또한 기존의 연구들은 가장 의심도가 높은 결함을 먼저 식별하는 방식(most suspicious first) 중심으로 진행 중이나, 다중 결함 환경에서는 결함 간 상호 영향이 발생하므로 다른 결함에 영향을 가장 많이 미치는 주요 결함을 먼저 식별 및 해결하는 방식(most dominant first)에 대한 연구를 수행할 계획이다.

Acknowledgement

이 논문은 2015 년도 정부(교육과학기술부)의 재원으로 한국연구재단-차세대정보컴퓨팅개발사업의 지원을 받아 수행된 연구임(No. 2015045358)

참고문헌

[1] Vessey, Iris. "Expertise in debugging computer programs: A process analysis." International Journal of Man-Machine Studies 23.5 (1985)

[2] Jones, James A., and Mary Jean Harrold. "Empirical evaluation of the tarantula automatic fault-localization technique." Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. ACM, 2005.

[3] Abreu, Rui, Peter Zoeteweij, and Arjan JC Van Gemund. "On the accuracy of spectrum-based fault localization." Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION, 2007. TAICPART-MUTATION 2007. IEEE, 2007.

[4] Debroy, Vidroha, and W. Eric Wong. "Insights on fault interference for programs with multiple bugs." Software Reliability Engineering, 2009. 20th International Symposium on. IEEE, 2009.

[5] DiGiuseppe, Nicholas, and James A. Jones. "On the influence of multiple faults on coverage-based fault localization." Proceedings of the 2011 international symposium on software testing and analysis. ACM, 2011.

[6] DiGiuseppe, Nicholas, and James Jones. "Fault interaction and its repercussions." Software Maintenance (ICSM), 2011 27th IEEE International Conference on. IEEE, 2011.

[7] Abreu, Rui, Peter Zoeteweij, and Arjan JC Van Gemund. "Spectrum-based multiple fault localization." Automated Software Engineering, 2009. ASE'09. 24th IEEE/ACM International Conference on. IEEE, 2009.

[8] Jones, James A., James F. Bowring, and Mary Jean Harrold. "Debugging in parallel." Proceedings of the 2007 international symposium Fn Software testing and analysis. ACM, 2007.

[9] Yu, Zhongxing, Chenggang Bai, and Kai-Yuan Cai. "Does the Failing Test Execute a Single or Multiple Faults? An Approach to Classifying Failing Tests.", ICSE 2015

[10] Hogerle, Wolfgang, Friedrich Steimann, and Marcus Frenkel. "More Debugging in Parallel." Software Reliability Engineering (ISSRE), IEEE 25th International Symposium on. IEEE, 2014.

[11] Gopinath, Divya, Razieh Nokhbeh Zaeem, and Sarfraz Khurshid. "Improving the effectiveness of spectra-based fault localization using specifications." Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on. IEEE, 2012.

[12] Naish, Lee, and Kotagiri Ramamohanarao Neelofar. "Multiple Bug Spectral Fault Localization Using Genetic Programming."

[13] Kim, Jeongho, Jonghee Park, and Eunseok Lee. "A new hybrid algorithm for software fault localization." Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication. ACM, 2015

Circus 명세를 통한 Go 코드 생성

*신지훈, **최진영

*고려대학교 컴퓨터전파통신공학부, **고려대학교 정보보호대학
서울특별시 성북구 안암로 145
{jeehoon, choi}@formal.korea.ac.kr

요약: 정형 언어인 Circus 로 작성된 Circus 명세는 refinement calculus 를 통해 정형 설계서를 도출한다. 이 때에 정형 검증은 명세의 정확성을 높일 수 있으며, 정제(refinement) 과정을 통해 명세서에 부합하는 설계서를 작성할 수 있다. 하지만 일반적 코딩을 통해서는 설계서까지 보증한 정확성을 코드로까지 이어가기가 힘들다. 본 연구에서는 이를 보증하기 위한 코드 생성 기법을 제안한다. 코드 생성 기법에는 변환 전략 및 규칙들을 포함한다. 변환 전략은 각각의 설계 항목(예. 프로세스, 시스템 상태, 오퍼레이션, 액션 등)을 변환하는 방법들을 포함한다. 변환 규칙은 Circus 명세 전체 또는 일부분을 입력 받아 대응하는 go 코드로 도출하는 변환 함수로 정의된다. 본 연구에서는 제안하는 방법을 사용하여 go 코드를 도출하는 간단한 사례를 보인다. [1]

핵심어: 정형기법, Circus 명세, Golang, 코드변환

1. 서론

정형기법[1]을 사용한 소프트웨어 개발에서 소프트웨어 명세자는 소프트웨어를 명세하고 명세가 만족해야 하는 속성을 검증한다. 이후 refinement calculus[2]와 같은 방법을 통해 이 명세로부터 정제하여 소프트웨어 설계서를 도출하고 코드를 개발한다. 설계서를 도출할 때는 정형화된 방법을 사용하고 각 정제된 명세 및 설계서가 기존 명세를 만족하는지 정확성을 검증함으로써 기존 명세서에 부합하는지를 확인할 수 있다.

하지만 이 설계서를 토대로 개발자가 작성한 코드는 설계서를 만족하는 지 보증하기가 힘들다. 1) 개발자가 설계서와 설계서 작성에 사용된 언어에 대한 완벽한 이해가 없다면 올바른 코드 작성이 힘들며 2) 개발자가 설계서에 사용된 정형 언어에 대한 전문성이 있다 해도 코드 작성 시 실수가 유입될 수 있다. 만약 원자력, 의료, 항공 및 철도 시스템과 같이 안전필수 시스템 개발일 경우, 이런 오류들은 심각한 문제를 야기할 수 있다.

설계서로부터 정확한 코드로 도출하기 위해 제안된 몇몇의 방법이 있다. 첫 번째로는 design by contract[3]이다. Design by contract 에서는 설계자가 정형적이고 명료하며 검증 가능한 언어를 사용하여 소프트웨어 컴포넌트의 인터페이스 명세를 정의한다. 개발자는 설계서를 바탕으로 코드를 작성하고 작성된 코드가 인터페이스 명세를 만족하는 지를 검증한다. Eiffel, Ada 검증을 위한 SPARK, Java 검증을 위한 JML, C# 검증을 위한 SPEC#과 같이 코드 검증을 위해 인터페이스 명세 언어가 있다. 인터페이스 명세를 통해 코드가 만족한다는 것을 검증함으로써 코드를 보증할 수 있지만, 인터페이스 명세 역시 설계자가 수동으로 작성하며 인터페이스 명세가 설계서를 부합하는지 검증하는 것이 힘들기 때문에 인터페이스 명세 자체에 오류가 삽입될 수 있는 단점이 있다. 두 번째 방법은 자동 코드 생성이다. 자동 코드 생성은 개발자가 코드 작성에 개입하지 않기 때문에 사람의 실수가 들어갈 위험을 미연에 방지 할 수 있다. 설계서로부터 코드를 자동으로 생성하기 위해서는 명세 및 설계로부터 코드를 도출하는 변환 방법이 요구된다. 정형명세로부터 코드를 도출하는 연구로는 JCircus[4], Aterier B[5]의 BART(B method Automatic Refinement Tool), B2Code[6], SCADE KCG 코드 생성기, ESPARK[7] 등이 있다.

본 연구의 목적은 자동 코드 생성을 위한 변환 방법을 제시하는 것이다. 본 연구에서 변환은 Circus 문법 일부분에 대해 수많은 변환 함수들을 통해 이루어진다. 이 변환 함수들은 전체 Circus 명세 혹은 일부 Circus 명세를 입력 받아 대응하는 Go 코드 생성 또는 다른 변환 함수로의 전달하는 역할을 한다. 이 방법을 통해 수동으로 작성된 코드보다 더 정확하고 안전한 코드를 생성할 수 있다.

본 논문에서는 Circus 명세로부터 Go 코드를 도출하는 변환 전략 및 규칙들을 제안한다. 변환 규칙들은 함수들로 정의되며 이는 구문적으로 적용된다. 정의한 함수들은 반복적으로 적용함으로써 점진적으로 Go 코드를 생성한다. 2 장에서는 변환에 사용되는 Circus 언어와 Golang 에 대해 소개한다. 3,4 장에서

- 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성 지원사업의 연구결과로 수행되었음" (IITP-2015-H8501-15-1012)

는 명세를 변환하기 위한 변환 전략 및 규칙들은 제안한다. 5 장에서는 누산기(accumulator)를 사례연구로 제안하는 방법을 적용하여 Go 코드 도출 과정을 설명한 후 본 논문을 마무리 한다.

2. 배경

2.2 Circus

Circus(Concurrent and Integrated Refinement CalculUS)[8]는 Z notation, CSP[6] 및 refinement calculus 를 통합한 정형 명세 언어이다. Circus 는 Z 와 CSP 를 통합함으로써 동시성(concurrent) 시스템의 데이터와 행위 측면을 모두 나타낼 수 있다는 장점이 있다.

Circus 명세는 채널 선언, 타입 선언 및 프로세스로 구성된다. 프로세스는 상태와 상태를 전이시키는 오퍼레이션 프로세스 전체 제어 행위를 나타내는 액션으로 구성된다.

$$\textbf{channel } in, out : \mathbb{N}$$

$$\begin{aligned}
&\textbf{process } GCD_Euclidean \mathrel{\widehat=} \textbf{begin}\\
&\quad \textbf{state } GCDState \mathrel{\widehat=} [a, b : \mathbb{N}]\\
&\quad InitState \mathrel{\widehat=} x, y : \mathbb{N} \bullet a, b := x, y\\
&\quad UpdateState \mathrel{\widehat=} a, b := b, a \bmod b\\
&\quad GCD \mathrel{\widehat=} \mu X \bullet \textbf{if } b = 0 \rightarrow out!a \rightarrow Skip\\
&\qquad\qquad\qquad [\!] \; b \neq 0 \rightarrow UpdateState;\ X\\
&\qquad\qquad\qquad \textbf{fi}\\
&\quad \bullet in?x \rightarrow in?y \rightarrow InitState(x, y);\ GCD\\
&\textbf{end}
\end{aligned}$$

그림 1 Circus 명세 예시

위의 Circus 명세는 유클리드 호제법을 나타낸다. a, b 의 최대 공약수를 구하는 알고리즘으로 GCD_Euclidean 프로세스는 a, b 의 값을 상태로 가지며 초기화시키는 InitState 와 updateState, GCD 오퍼레이션과 이들의 제어 정보인 액션 그리고 통신을 위한 채널로 구성되어 있다.

in 채널로부터 x 와 y 의 값을 받아서 a 와 b 에 각각 입력하는 초기화(InitState)를 수행한다. 이후 GCD 를 수행한다. GCD 는 b 의 값이 0 이면 a 값을 out 최종 값으로 출력하며 종료하고 그렇지 않을 경우 UpdateState 를 수행한다. 또한 다시 GCD 를 재귀적으로 수행한다. UpdateState 에서는 a 의 값은 b 로 b 의 값은 a mod b 의 결과 값을 할당한다.

2.2 Go language

Go language(Golang)[9]는 2009 년 구글에서 개발한 프로그래밍 언어이다. 기본적인 문법은 C 에서 가져왔으며 이에 가비지 컬렉션, 타입 안전성 등의 특징을 추가하였다. Golang 에서 동시성은 CSP 에 기반을 두고 있다.

o 타입 추론 (ex. x := 0 -> x int)
o 고차 함수 지원
o 멀티 리턴 지원
o 동시성 지원
- 고루틴을 이용한 경량 프로세스
- 채널
- select 구문
o 런타임 때 교착상태(deadlock) 탐지 및 race condition 방지 지원

```
func f() (int, int) {
    return 5, 6
}

func main() {
    x, y := f()
}
```

```
package main

import (
    "fmt"
    "time"
)

func pinger(c chan string) {
    for i := 0; ; i++ {
        c <- "ping"
    }
}
func printer(c chan string) {
    for {
        msg := <- c
        fmt.Println(msg)
        time.Sleep(time.Second * 1)
    }
}
func main() {
    var c chan string = make(chan string)

    go pinger(c)
    go printer(c)

    var input string
    fmt.Scanln(&input)
}
```

그림 2 Golang 코드 특징

그림 2 좌측 Golang 코드는 멀티 리턴을 보여주고 있다. 함수 f 는 입력 없이 int 형 값 2 개를 리턴한다. 함수 바디를 살펴보면 int 형 값인 5 와 6 을 각각 리턴하는 것을 볼 수 있다. 메인 함수에서는 함수 f 를 호출하여 x 와 y 변수에 각각 5, 6 을 할당한다.

Circus 의 오퍼레이션에서는 멀티 출력을 지원한다. 이는 Circus 의 멀티 출력을 하는 오퍼레이션에서 Go 코드로 변환 시 직접적으로 변환을 가능케 한다. 만약 멀티 출력을 지원하지 않는 언어로 변환 할 경우 두 개 이상의 출력을 하나의 데이터로 나타낼 수 있는 타입을 정의하여 한번 더 정제해야 하는 수고스러움이 따른다

그림 2 우측 코드는 Go 코드의 동시성을 보여주고 있다. 메인 함수 내의 함수 호출 앞에 go 키워드를 붙이면 동시적으로 동작하게 된다. 즉 그림에서 메인 함수와 pinger 함수 printer 함수는 동시적으로 동작하게 된다. Go 코드에서 함수 호출 앞에 go 키워드를 붙이면 해당 함수는 자체적 스택 스페이스를 가지는 고루틴(goroutine)으로 실행한다.

3. Circus2Go 변환

3.1 변환 전략

본 절에서는 Circus 명세를 구성하는 채널, 프로세스(상태 스키마, 오퍼레이션 스키마, 액션 등)들이 Go 코드로 변환되는 전략을 설명한다. 그림 3 은 Circus 명세의 각 구성요소가 변환되는 Go 코드를 보여주는 개념도이다.

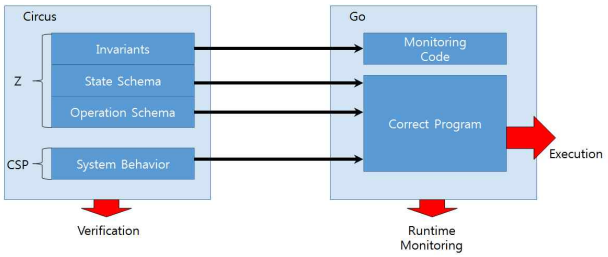


그림 3 Circus2Go 변환 개념도

Circus 명세에서 시스템 상태를 나타내는 상태 스키마와 상태를 전이시키는 오퍼레이션 스키마 프로세스의 행위를 나타내는 액션부분은 실제 동작하는 프로그램 코드로 변환된다. 시스템이 항상 만족해야 하는 불변속성들(Invariants)와 타입 제약사항(필요시)들은 프로그램 동작 시 위배하는 지 동시적으로 모니터링하는 모니터링 코드로 변환된다.

(1) 프로세스

Circus 명세 내의 프로세스는 Go 코드의 메인 함수 안의 함수로 도출된다. 아래의 그림에서 Circus 명세 내 2 개의 프로세스(Controller, Ring)을 확인 할 수 있다. 이는 오른편의 메인 함수 내의 함수(ControllerMain, RingMain)으로 도출된다. 이 함수들 앞의 붙은 go 키워드는 이 프로세스 들이 동시적(concurrent)으로 동작함을 의미한다.

그림 4 프로세스 변환 전략

(2) 프로세스 액션

프로세스의 액션은 각 프로세스 함수 내의 로컬 함수들의 콜과 제어문으로 구성된다. 아래의 그림에서 Controller 프로세스의 액션은 처음 ControllerInit 으로 초기화 후 InputController 와 OutputController 중 하나가 반복적으로 수행된다. 이는 Go 코드의 ControllerMain 함수 내에 나타나져 있다. 처음 ControllerInit() 함수를 수행 후 무한히 반복하며 InputController() 와 OutputController() 함수를 수행한다. 아래와 같이 결정적 선택에 있어 함수를 수행하는 조건문은 함수 내에 작성된다.

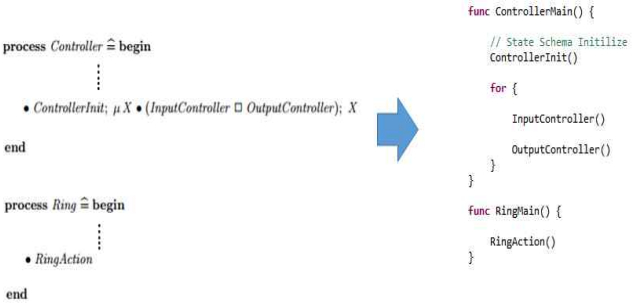


그림 5 프로세스 액션 변환 전략

(3) 채널

Go 에서는 채널 역시 전역 변수처럼 선언이 된다. 하지만 일반 전역 변수와는 다르게 make(chan TYPE)과 같은 방법으로 선언된다. 아래의 그림에서 4 개의 채널(input, output, write, read)이 선언되고 있다. input, output 은 각각 정수형 타입의 채널로 make 키워드를 이용하여 선언된다. write 와 read 채널은 복합 타입으로써 새로운 타입 oneToMaxringbyNat 을 정의하여 선언한다.

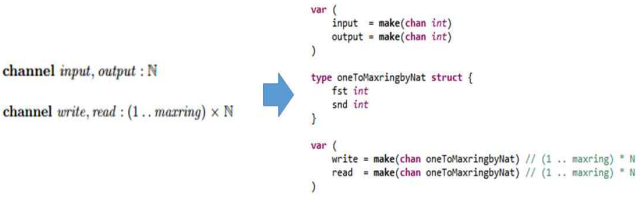


그림 6 채널 변환 전략

(4) 상태

프로세스의 상태는 전역 변수의 집합으로 변환된다. 각 상태 변수들은 해당 전역 변수로 도출되며 선언된 타입은 해당 타입으로 변환된다. 타입 변환에 대해서는 아래의 Translation Rules 에서 언급한다. 아래의 그림에서 Controller 의 상태는 size, ringsize, cache, top, bot 으로 구성된다. 이는 우측 상단의 5 개의 전역 변수로 그대로 도출 되며 Ring 의 상태 변수인 ring 역시 그대로 ring 전역 변수로 도출된다.

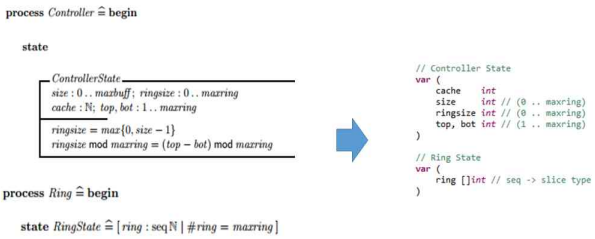


그림 7 상태 변환 전략

(5) 불변속성
상태스키마 하단에 정의되어 있는 불변속성은 각각의 고루틴으로 정의된다. ControllerState 상태스키마의 불변속성은 2 가지로써 각각 ControllerInvariant1Check(), ControllerInvariant2Check()의 함수로써 나타나며 이는 메인함수에서 go 키워드를 이용하여 동시적으로 동작하며 불변속성이 위배되는지를 확인한다. RingState 상태스키마 역시 #ring = maxring 이라는 불변속성이 존재하며 이는 RingInvariant1Check()로 변환된다. 불변속성의 내용은 이 함수 내의 조건문에 들어가며 조건이 만족하지 않을 경우, 즉 불변속성이 위배될 경우 오류 메시지를 출력한다. RingInvariant1Check() 역시 메인 함수 내의 고루틴으로 동시적으로 동작하며 확인한다. 불변속성은 goSched()함수를 사용하여 다른 함수들의 실행 후 항시 확인하며 불변속성 확인 시 다른 고루틴이 끼어드는 것을 막는다

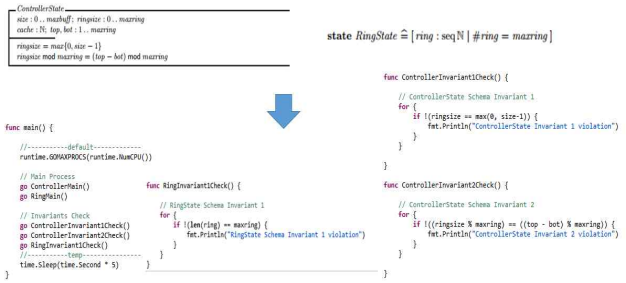


그림 8 불변속성 변환 전략

(6) 오퍼레이션, 액션
오퍼레이션 스키마와 액션은 각각 함수로써 정의 된다. 오퍼레이션 스키마의 전조건(precondition)은 함수 내의 if 구문의 조건문으로 후조건(postcondition)은 조건문이 만족했을 경우 수행되는 코드로 변환된다. 액션 역시 & 앞에 붙은 조건문은 함수 내의 if 구문의 조건문으로, & 뒤의 구문은 조건문 내의 수행되는 코드로 변환된다. 아래의 그림에서 StoreInputController 오퍼레이션의 전조건인 0 < size < maxbuff 는 함수 StoreInputController() 의 if 문의 조건문으로 변환되었으며 후조건은 if 문 내의 수행문으로 변환되었다. InputController 액션에서 & 구문 앞의 조건문 size < maxbuff 은 if 문의 조건문으로 변환되었으며 이를 만족할 시 뒤의 구문을 if 문의 수행문에 들어간다. 이 수행문을 살펴보면 input?x 즉 input 채널에서 x 값을 받아오는 구문은 x := <- input 으로 변환하였으며 그 후의 2 개의 & 앞의 전조건은 각각 if 조건문 그 후의 구문은 각각의 조건문 내의 수행문으로 변환되었다.

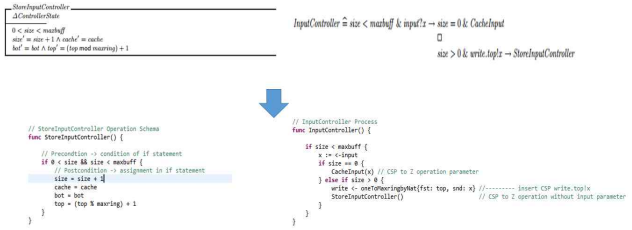


그림 9 오퍼레이션, 액션 변환 전략

3.2 변환 규칙 소개

변환 규칙은 변환 함수들로써 정의된다. 변환 함수는 Circus 언어의 전체 또는 일부분을 입력 받아 문법적으로 판단하여 해당 Go 코드를 생성 또는 해당하는 다른 변환 함수에 적용한다.

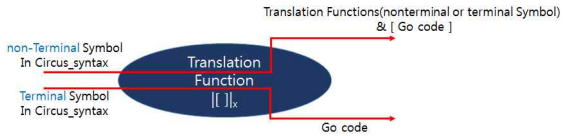


그림 10 변환 함수 소개

표 1 변환 함수 예시

Program ::= AC | B **A ::= a** **B ::= b** **C ::= c**	
[[Program]] Program → If Program = AC then **[[AC]]** AC ElseIf Program = B then **[[B]]** B	**[[AC]]** AC → **[[A]]** A & **[[C]]** C

|[A]|$_A$ → **|[a]|$_a$**	**|[B]|$_B$** → **|[b]|$_b$**	**|[C]|$_C$** → **|[c]|$_c$**
|[a]|$_a$ → ***Program code a***	**|[b]|$_b$** → ***Program code b***	**|[c]|$_c$** → ***Program code c***

표 1 은 명세 언어 문법과 변환 규칙의 예시를 보여준다. Program 은 전체 프로그램으로써 AC 또는 B 가 될 수 있다. 이 때 변환 함수에 Program 이 입력으로 들어오면 |[ ]|$_{Program}$ 함수에 적용한다. |[Program]|$_{Program}$ 적용된 결과는 Program 이 AC 일 경우 |[ ]|$_{AC}$ 함수에 입력으로 AC 를 전달하고 B 일 경우 |[ ]|$_B$ 함수에 입력으로 B 를 전달한다.
표 2 에서 입력 받은 전체 명세 Program 은 AC 로써 AC 함수에 입력 값 AC 를 전달한다. AC 함수는 A 부분을 A 함수에 C 부분은 C 함수에 전달하며 "&" 코드를 생성한다. A 함수와 C 함수는 다시 a 함수, c 함수에 입력 값 a, c 를 전달하고 a 함수는 템플릿 코드인 "Program code"와 입력 값 a 를 토대로 "Program code a"라는 코드를 생성해 낸다. c 함수 역시 같은 동작을 수행하여 전체적으로 "Program code a & Program code c"라는 코드를 도출해 낸다.
즉 변환 함수는 문법 중 non-Terminal 심볼(표 1 의 Program, AC, A, B, C)을 입력 받으면 해당하는 변환 함수에 적용하거나 코드의 일부분(AC 함수의 '&')을 생성하며 Terminal 심볼(표 1 의 a, b, c)을 입력 받으면 해당하는 코드를 생성한다.

표 2 변환 함수 적용 예시

Program = AC
→ |[Program]|$_{Program}$
→ |[AC]|$_{AC}$
→ |[A]|$_A$ & |[C]|$_C$
→ |[a]|$_a$ & |[C]|$_C$
→ Program code a & |[C]|$_C$
→ *Program code a* & |[c]|$_c$
→ *Program code a* & Program code c

본 연구에서는 Circus 문법 일부분에 대한 약 100 여개 이상의 변환 함수들이 정의되어 있다. 아래에서는 이 중 대표적인 함수들을 소개한다.

(1) 채널 관련 변환 함수
Circus 에서는 채널을 이용하여 각 프로세스들 간의 통신이 이루어 진다. 이는 Go lang 에서의 채널로 변환된다. 둘 이상의 채널 선언은 CircusPar()를 통해 분할 되며 이는 각 해당 타입 선언으로 변환되며 크기는 임의로 10 으로 선언된다. 만약 Circus 에서 타입이 선언되어 있지 않다면 크기가 없는 채널, 즉 동기화 채널로 선언된다. Go lang 에서는 이를 임의의 타입 T 에 크기가 없는 채널로 선언하면 동기화 채널로 인식한다.

그림 11 채널 관련 변환 함수들

(2) 프로세스 관련 변환 함수
Circus 에서 선언된 프로세스들은 Go lang 에서 Main() 함수 내에 고루틴으로 동작한다. 고루틴은 순차적이 아니라 독립적으로 동작하는 함수를 의미한다. 이는 go lang 에서 함수명 앞에 go 키워드를 사용하여 나타낸다. 프로세스는 메인 함수에서 호출되며 이는 각각의 해당 이름의 함수로써 정의된다.

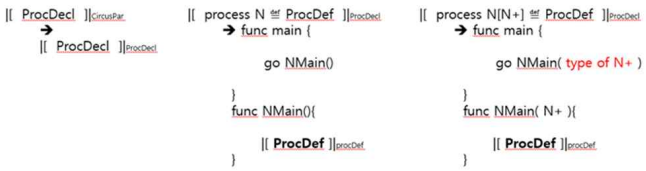


그림 12 프로세스 관련 변환 함수들

(3) 상태 선언 관련 변환 함수
Circus 에서 스키마는 전체 시스템의 상태를 나타내는 상태 스키마와 시스템의 전이를 나타내는 오퍼레이션 스키마로 나뉜다. 상태 스키마는 또한 시스템의 상태를 구성하는 상태 변수들과 이들의 불변 속성을 나타낸다. 상태 스키마 내에 선언되는 상태 변수들은 Go lang 에서 var 키워드를 이용하여 선언되는 전역 변수로 변환된다. 또한 타입은 다음과 같이 매핑 되어 변환된다.
o Z, N, Number1 ... Number2 → int

o seq Z, seq N, seq Number1 ... Number2
→ []int{}
o N X ... X N → struct
기본적으로 정수, 자연수, 범위 타입은 int 타입으로 변환되며 이에 대한 시퀀스 타입은 슬라이스 타입으로 변환된다. 또한 카티션 곱으로 이루어진 순서쌍 타입은 구조체로 선언된다. 또한 정수 외 타입들은 해당 변수들이 범위를 넘어서는지 확인하는 고루틴을 생성한다. 이 고루틴은 해당 변수_타입() 의 이름을 가지게 된다.

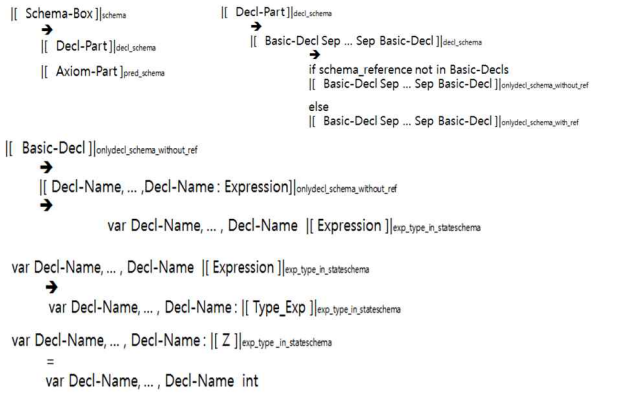


그림 13 상태 선언 관련 변환 함수들

(4) 오퍼레이션 선언 관련 변환 함수
오퍼레이션 스키마는 하나의 함수로써 변환된다. 오퍼레이션 스키마 내의 선언 부분은 상태 스키마 선언 부분과 비슷하게 동작한다. 하지만 상태 스키마의 상태 변수들은 전역 변수들로 선언되지만 오퍼레이션 스키마 내의 변수들은 해당 함수 내의 지역 변수, 입력 및 출력으로 선언된다. 변수명에서 ?가 붙으면 입력, !은 출력 그 외에는 지역 변수로 나타내진다. 또한 지역 변수, 입력, 출력에 대해 각각 변환되는 위치가 다르다. 입력과 같은 경우는 함수의 괄호 안 시그니처로 들어가며 지역변수는 바디 내에 선언되고 출력은 함수 정의 뒤 return 키워드 뒤에 나타나게 된다. 지역 변수에 대한 타입 변환은 상태 스키마에서의 타입 변환과 동일하게 이루어 진다. 하지만 지역 변수 타입들에 대한 범위 위반 확인 작업은 새로운 고루틴이 아닌 해당 함수 내에서 이루어 진다는 차이가 있다.

그림 14 오퍼레이션 관련 변환 함수들

(5) 불변 속성 및 오퍼레이션 술어 관련 변환 함수
상태 스키마의 불변 속성은 메인 함수 내의 하나의 고루틴을 생성한다.
이 고루틴 내에는 해당 속성을 조건으로 하여 속성을 위배하는 지를 독립적으로 확인한다.
오퍼레이션의 술어부는 오퍼레이션의 전조건(precondition)과 후조건(postcondition)으로 나뉜다. 이들은 각각 해당 함수의 수행 조건문과 조건문 내의 수행문으로 변환된다.

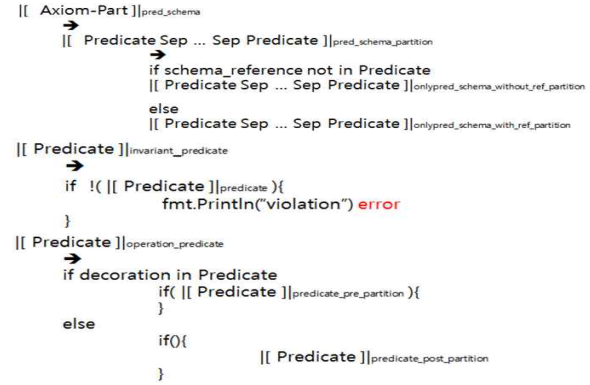


그림 15 불변 속성 및 오퍼레이션 술어 관련 변환 함수들

(6) 액션 관련 변환 함수
Circus 에서 액션은 전체 시스템의 제어를 의미한다. 즉 각 오퍼레이션의 순서 및 수행 조건들을 나타낸다. 채널을 통한 입력 및 출력은 <-와 -> 키워드로 변환되며 수행 조건인 Pred&Action 은 Pred 는 조건문으로 Action 은 조건이 만족할 경우 수행되는 수행문으로 변환된다. 순차적 수행을 나타내는 ; 는 순서대로 해당 오퍼레이션이 수행되며 재귀 호출을 나타내는 u N. Action ; N 은 for 문을 이용하여 반복적으로 Action 이 수

행되도록 한다. 비결정적으로 동작하는 Action n Action 은 랜덤함수를 이용하여 랜덤하게 함수를 수행토록 한다.

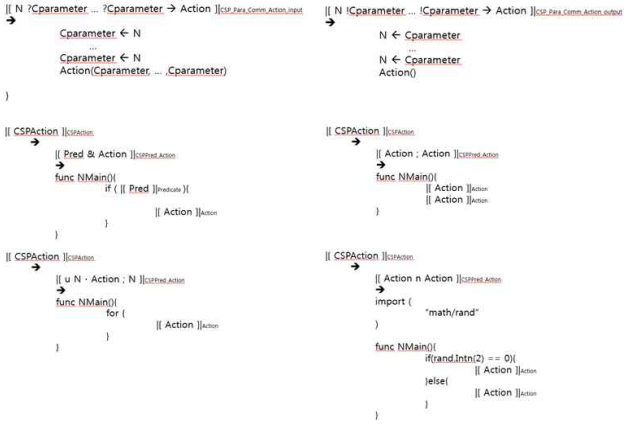


그림 16 액션 관련 변환 함수들

4. 사례 연구

4.1 사례 : Accumulator

사례 연구의 대상은 누산기이다. 누산기는 정수 타입의 값을 저장하고 이 값을 하나씩 늘려나가는 아주 단순한 시스템이다.

$$
\begin{aligned}
&\textbf{channel } out : \mathbb{N} \\
&\textbf{process } Accumulator1 \ \widehat{=} \\
&\textbf{begin} \\
&\quad \textbf{state} \\
&\quad State \ \widehat{=}\ [v : \mathbb{N}] \\
&\quad AddOp \ \widehat{=}\ [\Delta State \mid v' = v + 1] \\
&\quad \bullet\ (\mu X \bullet out!v \rightarrow AddOp\,;X) \\
&\textbf{end}
\end{aligned}
$$

그림 17 누산기 Circus 명세

누산기는 자연수 타입의 값을 저장하는 변수 v 를 가지며 이 값을 하나씩 늘려나가는 오퍼레이션 AddOp 가 존재한다. 또한 현재의 값을 출력하는 out 채널이 있다. 이 시스템의 액션은 현재의 값을 out 채널에 출력하고 AddOp 를 수행 후 재귀적으로 이를 반복한다.

4.2 변환 과정

본 절에서는 변환 과정을 순차적으로 보여준다. 본 절에서 표의 왼쪽 부분은 전체 명세 중 변환되는 대상인 명세 일부분을 나타낸다. 오른쪽 상단 부분은 정의된 변환 함수를 오른쪽 중간 부분은 정의된 변환함수에 해당 명세가 적용되는 과정 및 결과를 맨 하단 부분은 이로써 도출된 Go 코드를 보여준다. 변환 과정 중 초기에 발생하는 채널, 채널셋, 타입선언, 프로세스 구분 과정은 생략한다.

표 3 변환 과정 1

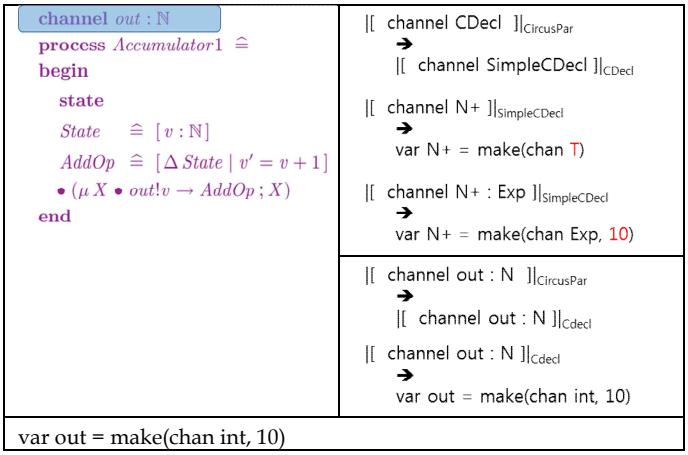


표 3 은 채널 선언 부분의 변환 과정을 보여준다. 채널 선언 변환 함수인 SimpleCDecl 함수는 입력값 “channel out : N”을 입력 받아 정의된 “var N+ = make(chan Exp, 10)에서 out 과 N+가 N 이 Exp 와 매핑되어 “var out = make(chan int, 10)”이라는 코드가 도출된다. 여기서 N 타입은 Exp 로 도출되고 Exp 는 다시 타입 함수에 적용되어 int 를 도출한다. 타입 도출 과정은 지금은 생략하며 이후 변환 과정에서 다시 언급한다.

표 4 변환 과정 2

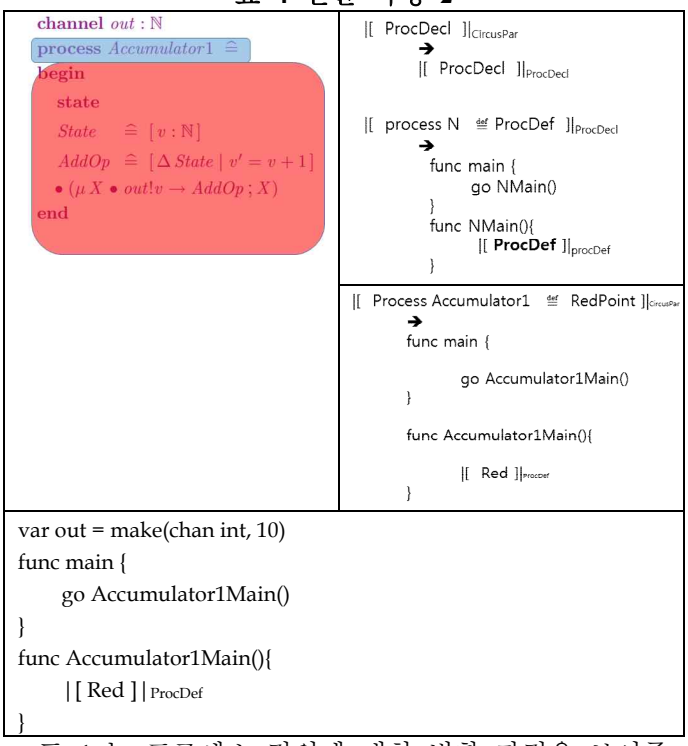
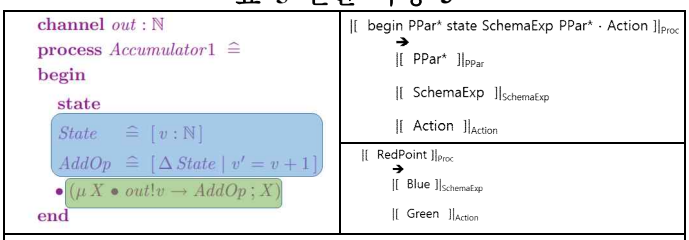


표 4 는 프로세스 정의에 대한 변환 과정을 보여준다. Process 이름인 N 에 해당하는 Accumulator1 에 대한 Accumulaotr1Main 함수를 생성하고 이를 메인

함수에서 호출한다. 프로세스 내용인 붉은 색 부분은 다시 ProcDef 함수에 적용된다. 이후 ProcDef 함수는 붉은 색 부분을 다시 Proc 함수에 전달한다.

표 5 변환 과정 3




```
var out = make(chan int, 10)
func main {
    go Accumulator1Main()
}
func Accumulator1Main(){
    |[  Blue ]|SchemaExp
    |[  Green  ]|Action
}
```

프로세스 정의는 프로세스 내에 사용할 타입 및 공리 선언, 오퍼레이터 정의를 하는 패러그래프(PPar), 상태 및 오퍼레이션 스키마(SchemaExp), 액션(Action)으로 구성된다. 표 5 는 이들을 구분하여 각각 PPar, SchemaExp, Action 함수에 적용한다. 본 예시에서는 패러그래프는 존재하지 않음으로 이 부분은 PPar 함수에 전달하지 않고 스키마에 해당하는 파란색 부분을 SchemaExp 함수에 액션에 해당하는 초록색 부분을 Action 함수에 전달한다.

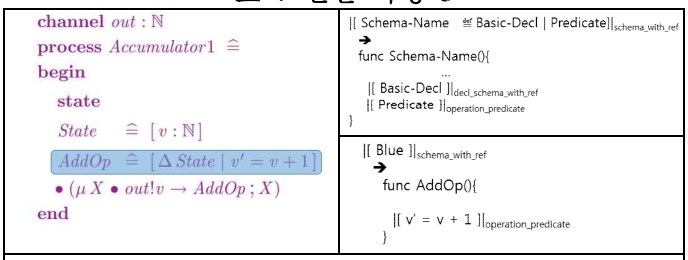
표 6 변환 과정 4


```
var out = make(chan int, 10)
var v   int
```

```
func main {
    go Accumulator1Main()
    go v_Type()
}
func Accumulator1Main(){
    |[  Green  ]|Action
}
func v_Type() {
    if( v < 0) {
        fmt.print( "v error" )
    }
}
```

표 6 은 상태 선언에 대한 변환 과정을 보여준다. Decl_Name 에 해당하는 v 는 onlydecl_schema_without_ref 함수에 적용되어 "var v"코드를 도출하고 뒤의 타입 표현인 N 은 exp_type_in_stateschema 함수에 적용된다. 입력값 N 을 통해 "int"가 도출되고 변수 v 가 0 보다 작은 값을 가지는 지를 모니터링하는 v_Type 고루틴을 생성하여 v 의 값이 0 보다 작은 값을 가지는 지 동시적으로 확인한다. 만약 상태 스키마에 불변속성이 존재한다면 이와 같은 방법으로 고루틴을 생성한다.

표 7 변환 과정 5




```
var out = make(chan int, 10)
var v   int
func main {
    go Accumulator1Main()
    go v_Type()
}
func AddOp(){
    |[ v'  = v + 1 ]|operation_predicate
}
func Accumulator1Main(){
    |[  Green  ]|Action
}
func v_Type() {
    if( v < 0) {
        fmt.print( "v error" )
    }
}
```

표 7 은 오퍼레이션 스키마의 변환 과정을 보여준다. Decl_schema_with_ref 함수는 오퍼레이션의 선언부(로컬 함수, 입력 및 출력) 부분을 변환하는 데 사용된다. 본 예제에서는 선언부가 없으므로 이는 무시된다. Predicate 부분인 v′ = v + 1 부분은 다시

operation_predicate 함수로 전달된다. operation_predicate 함수는 입력을 그대로 predicate 함수에 전달한다.

표 8 변환 과정 6

<table>
<tr>
<td>channel $out : \mathbb{N}$
process $Accumulator1 \triangleq$
begin
state
$State \triangleq [v : \mathbb{N}]$
$AddOp \triangleq [\Delta State \mid v' = v + 1]$
$\bullet (\mu X \bullet out!v \rightarrow AddOp ; X)$
end</td>
<td>|[Predicate]|predicate
➔
|[Expression Rel Expression]|predicate
|[Expression = Expression]|predicate
➔
|[Expression]|expression = |[Expression]|expression
|[Expression1 + Expression2]|predicate
➔
|[Expression1]|expression + |[Expression2]|expression
|[Var-Name]|expression
➔
Var-Name

|[v' = v + 1]|predicate
➔
|[v']|expression = |[v + 1]|expression
|[v']|expression
➔
v
|[v + 1]|expression
➔
|[v]|expression + |[1]|expression
|[v]|expression
➔
v</td>
</tr>
<tr>
<td colspan="2">var out = make(chan int, 10)
var v int
func main {
go Accumulator1Main()
go v_Type()
}
func AddOp(){
v = v + 1
}
func Accumulator1Main(){
|[GreenPoint]|Action
}
func v_Type() {
if(v < 0) {
fmt.print(“v error”)
}
}</td>
</tr>
</table>

표 8 은 v′ = v + 1 부분의 변환 과정을 보여준다. v′ = (v + 1)으로 우선 predicate 함수에 적용되고 v + 1 은 다시 재귀적으로 적용되어 go 코드인 “=”과 “+”를 도출한다. 각각 나뉘어진 v′, v, 1 은 expression 함수에 전달되어 v, v, 1 을 도출하여 “v = v + 1”이라는 코드를 최종적으로 도출한다.

표 9 변환 과정 7

<table>
<tr>
<td>channel $out : \mathbb{N}$
process $Accumulator1 \triangleq$
begin
state
$State \triangleq [v : \mathbb{N}]$
$AddOp \triangleq [\Delta State \mid v' = v + 1]$
$\bullet (\mu X \bullet out!v \rightarrow AddOp ; X)$
end</td>
<td>|[CSPAction]|CSPAction
➔
|[u N · Action ; N]|CSPPred_Action
➔
func NMain(){
for {
|[Action]|Action
}
}

|[GreenPoint]|CSPAction
➔
|[u X· out!v → AddOp ; X]|CSPPred_Action
➔
func Accumulator1Main(){
for{
|[out!v → AddOp]|Action
}
}</td>
</tr>
<tr>
<td colspan="2">ar out = make(chan int, 10)
var v int
func main {
go Accumulator1Main()
go v_Type()
}
func AddOp(){
v = v + 1
}
func Accumulator1Main(){
for{
|[out!v → AddOp]|Action
}
}
func v_Type() {
if(v < 0) {
fmt.print(“v error”)
}
}</td>
</tr>
</table>

표 9 는 액션의 변환 과정을 보여준다. u 구문은 Action 을 수행하고 다시 u 구문을 수행함을 의미한다. 즉 Action 구문을 반복적으로 수행하게 된다. CSPPred_ Action 함수는 u(반복)구문을 입력받으면 메인 함수 내에 for 문을 도출하고 해당 Action 을 반복적으로 수행토록 한다. 해당 Action 인 out!v → AddOp 는 다시 Action 함수로 전달된다.

표 10 변환 과정 8

<table>
<tr>
<td>channel $out : \mathbb{N}$
process $Accumulator1 \triangleq$
begin
state
$State \triangleq [v : \mathbb{N}]$
$AddOp \triangleq [\Delta State \mid v' = v + 1]$
$\bullet (\mu X \bullet out!v \rightarrow AddOp ; X)$
end</td>
<td>|[
N ICparameter ... ICparameter → Action
]|CSP_Para_Comm_Action_output
➔
N ← Cparameter
...
N ← Cparameter
Action()

|[out!v → AddOp]|Action
➔
out ← v
AddOp()</td>
</tr>
<tr>
<td colspan="2">var out = make(chan int, 10)</td>
</tr>
</table>

```
var v   int
func main {
     go Accumulator1Main()
     go v_Type()
}
func AddOp(){
     v = v + 1
}
func Accumulator1Main(){
    for{
         out ← v
         AddOp()
    }
}
func v_Type() {
      if( v < 0) {
         fmt.print(“v error”)
      }
  }
```

표 10 은 메인 액션 내부의 통신 후 오퍼레이션 수행에 대한 변환 과정을 보여주고 있다. N!Cparameter 는 N 채널로 Cparameter 의 값을 보내는 것을 의미한다. out 은 N, v 는 Cparameter, AddOp 는 Action 에 매핑되어 “out ← v; AddOp()” 코드를 도출한다.

4.3 변환 결과

4.2 절의 변환 과정을 거쳐 그림 18 의 코드를 도출한다. 도출된 코드 외에 모듈 선언부분(package), 다른 모듈 호출부분(import), 코어 환경변수 세팅 함수인 GOMAXPROCS, time 함수는 기본적으로 생성된다.

```
package main

import (
    "fmt"
    "runtime"
//  "math/rand"
    "time"
)

var out = make(chan int, 10)

var v int

func v_Type(){

    if( v < 0 ) {
        fmt.Print( "V error")
    }
}
```

```
func AddOp(){
    v = v + 1
}

func Accumulator1Main(){

    for{
        out <- v
        AddOp()
    }
}

func main(){

    runtime.GOMAXPROCS(runtime.NumCPU())

    go Accumulator1Main()
    go v_Type()


    time.Sleep(time.Second * 10)
}
```

그림 18 누산기 변환 결과(Go 코드)

5. 결 론

Refinement Calculus 를 적용한 방법론에서는 개발할 시스템의 추상적 모델을 우선적으로 수립하고 이 모델을 바탕으로 정제해 나아가며 최종적으로 동작하는 코드를 도출한다. 모델을 정제하고 정제된 모델을 코드로 변환하는 데에 개발자가 수동으로 함으로써 개발자의 의도가 들어가고 또한 잘못된 정보 및 실수가 삽입될 가능성이 있다. 따라서 이 단점들을 제거하고자 이를 자동적으로 수행하는 연구들이 진행되고 있다. 본 연구에서는 정형 명세 언어 Circus 로 작성된 명세를 바탕으로 자동적으로 Golang 코드를 도출하는 연구를 수행하였다. 이를 위해 자동 변환을 위한 변환기(Translator)를 제안하였으며 변환 전략 및 규칙들을 설명하였다. 생성된 Golang 코드는 모니터링 코드의 항시 검사를 통해 속성 위배 여부를 확인한다. 코드의 신뢰성을 향상시킬 수 있으며 속성 위배 탐지 시 설계 또는 코드의 오류를 찾아낼 수 있을 것이다. 본 논문에서는 코드 생성의 정확성을 검증하기 보단 계속적인 속성 확인을 통해 생성된 코드가 올바름을 확인한다. 현재까지 정의된 변환 규칙들은 Circus 명세 언어의 일부분이며 향후 연구로써 나머지 규칙들을 정의할 필요가 있으며 이를 지원하는 툴 개발이 이루어져야 할 것이다.

참고문헌

[1] Edmund M. Clarke, Jeannette M. Wing: “Formal methods: state of the art and future directions”, ACM Computing Surveys(CSUR), 1996

[2] Carroll Morgan, Ken Robinson: “Specification Statements and Refinement”, Formal Approaches to Computing and Information Technology (FACIT), pp23-46, 1992

[3] Meyer, Bertrand: “Design by Contract”, Technical Report TR-EI-12/CO, Interactive Software Engineering Inc., 1986

[4] Angela Freitas and Ana Cavalcanti: “Automatic Translation from Circus to Java”, FM 2006: Formal Methods Lecture Notes in Computer Science Volume 4085, pp 115-130, 2006

[5] Atelier B, http://www.atelierb.eu/

[6] Xiaoli Liu and Rongquiang Fan: “A lightweight framework for code generation from B formal specification” ETCS2010 Volume 1, pp 133-136, 2010

[7] Rajiv Murali and Andrew Ireland, “E-SPARK: Automated generation of provably correct code from formally verified designs”, Electronic Communications of the EASST, vol 53, pp 1-15, 2012

[8] Jim Woodcock, Ana Cavalcanti: “A concurrent language for refinement”, Proceedings of the 5th Irish conference on Formal Methods, pp 93-115, 2001

[9] Meyerseon and Jeff: The Go Programming Language, IEEE Software Volumn 31, Issue 5, 2014

동시성을 포함한 액티비티 다이어그램 기반 테스트 시나리오 생성 기법

백승찬 [1], 최효린 [1], 이병정 [1], 이정원 [2]

[1] 서울시립대학교 컴퓨터과학부
서울 동대문구 전농동 163(전농동 90)
{bsch0111, yoinoichr2015, bjlee}@uos.ac.kr

[2] 아주대학교 전자공학과
경기도 수원시 영통구 월드컵로 206
jungwony@ajou.ac.kr

요약: 시스템의 요구사항이 복잡해지고 다양성을 띠게 되면서 명세 기반 테스트의 테스트 설계 자동화는 상당히 어려워졌다. 이러한 문제는 테스트 설계 과정에서 요구사항을 정형화하는 방향으로 해결되어 왔는데, 그 방법으로 모델 기반 테스트가 주로 사용되고 있다. 본 논문은 모델 기반 테스트의 테스트 시나리오 생성 방법에 주안점을 두고, 경로 폭발 문제(path explosion)에 대해 이를 회피하거나 개선할 수 있는 방법으로 동시성 경로의 노드의 쌍을 기반으로 한 새로운 탐색 기법에 대해서 소개한다. 사례 연구에서는 동시성 작업을 포함하는 액티비티 다이어그램을 제시하고, 새로운 탐색 기법이 만든 테스트 시나리오가 얼마나 효과적인지 기존의 방법과 비교해 본다. 본 연구 방법은 테스터가 경로 폭발 문제를 피하면서 반복과 동시성 경로에 대한 결함을 예상할 수 있는 효율적인 테스트 시나리오를 생성할 것으로 기대한다.

핵심어: 모델 기반 테스트, 테스트 시나리오 생성, 동시성 경로, 액티비티 다이어그램

1. 서론

소프트웨어 테스트를 효과적으로 수행하기 위해선 먼저 테스트 설계 단계를 거쳐야 한다. 이 단계에선 시스템의 요구사항을 분석하고 구현된 시스템에 대한 검증(verification)이 필요하고[1], 복잡한 시스템의 경우 테스터가 주체적으로 테스트 설계를 수행하기엔 많은 노력과 비용, 그리고 불완전한 테스트 케이스 등의 어려움이 있다. 따라서 효과적인 테스트를 위한 정형화된 테스트 설계 방법이 필요하다.

테스트 설계를 효과적으로 수행할 수 있는 방법으로 모델 기반 테스트(model-based test)가 소개되고 있다. 요구사항 분석 단계에서 소프트웨어의 행동(behavior)를 고려하면서 주요한 상태 정보의 내용에 기반한 효율적인 테스트 케이스를 만든다.

본 논문은 시스템의 행동을 표현하기에 용이한 액티비티 다이어그램(activity diagram)의 동시성 작업에 중점을 두는 새로운 탐색 기법을 제안한다.

2. 액티비티 다이어그램 분석 방법

액티비티 다이어그램이 표현할 수 있는 상황을 반복, 동시성을 중심으로 4 가지 형태로 구분하고, 이를 탐색하는 방법을 제안한다.

첫째로, 일반 경로(normal path)는 액티비티 다이어그램의 행동이 일련의 순서로 이루어진 형태를 말한다. 초기 상태부터 조건까지 혹은 조건부터 조건까지 하나의 경로로 보고 경로를 탐색하기 때문에 경로의 수에 상관없이 한 가지로 탐색한다.

다음은 시스템의 복잡성을 높이는 주요 원인인 반복을 3 가지 형태로 나누었다. 일반적인 조건과 순서로 동시성을 포함하지 않는 반복으로 이루어진 반복 경로(loop path)와 일정 시간 내에 동시에 진행되는 작업을 표현하는 동시성 경로(concurrency path), 반복 경로 속에 동시성 경로를 포함한 복합 경로(complex path) 3 가지이다.

반복 경로의 탐색 방법은 수행문의 결과가 조건문으로 잘 반영되는지 판단하기 위해 반복문의 형태에 따라 반복의 횟수를 지정하여 탐색한다. Do-while 반복의 경우에는 한 번, While-Do 반복의 경우에는 두 번 반복을 수행하면서 테스트 경로를 탐색한다.

동시성 경로는 반복경로와는 다르게 활동 간 순서가 존재하지 않다. 순서가 정해져 있지 않기 때문에 다양한 경로를 가질 수 있고 이는 경로 폭발 문제(path explosion)로 이어진다. 그러나 실제로 동시성을 포함한 프로그램에서 나타나는 오류의 부분적인 순서에서 대부분 일어나기 때문에 본 논문에서는 동시성 경로가 표현하는 활동의 쌍에 기반해서 테스트 경로를 탐색한다.

그림 1 에서 표현한 복합 경로의 경우엔 테스트 경로 폭발 문제를 야기시키는 원인 두 가지가 모두 존

재하고 있기 때문에 적절한 기준을 두어서 탐색해야 한다. 그래서 앞서 반복 경로의 두 가지 반복 유형과 동시성 경로의 쌍을 모두 고려하면서 탐색한다.

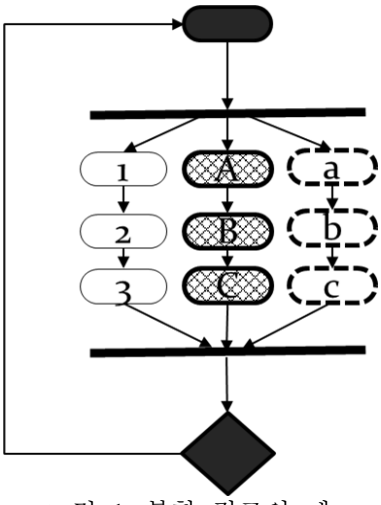


그림 1. 복합 경로의 예

3. 제어 흐름 그래프

기존의 제어 흐름 그래프 탐색 기법은 반복과 동시 작업에 대한 경로를 탐색할 수 없다. 그래서 본 논문에서는 4 가지의 경로 특성에 따라 다른 탐색기법을 적용하고 액티비티 다이어그램의 전체 흐름을 용이하게 확인하기 위해서 표 1 과 같이 7 가지 노드로 표현하는 제어 흐름 그래프를 제안한다.

표 1. 제어 흐름 그래프에 사용되는 도형

Path Type	Node Form
Decision Node	D
Normal Path	N
Loop Path	L
Concurrent Path	C
Complex Path	X
Start Node	●

4. 평가

본 논문에서 제안한 방법을 토대로 동시성을 가진 전자상거래 시스템의 기능 요구사항을 표현한 액티비티 다이어그램의 테스트 경로를 생성해보고 다른 논문에서 제안한 커버리지 기준으로 생성된 테스트 경로의 수를 표 1 과 같이 비교 하였다.

표 1. 기존의 경로 탐색 방법과 비교

Coverage criterion	Basis path coverage	Simple path coverage	Activity path coverage	Our paper coverage
Total Paths	2	1120	1660	126
Concurrent Path(s)	1(BFS)	560	560	42

5. 결론

본 논문은 모델 기반 테스트를 수행할 때 액티비티 다이어그램을 통해 테스트 시나리오를 생성하는 방법을 제안한다. 논문의 테스트 시나리오를 생성하는 방법은 동시성과 반복에 기반하여 액티비티 다이어그램이 표현할 수 있는 형태를 4 가지로 제안하고 각 형태에 따라 다른 탐색 기법과 제안한 탐색기법을 효과적으로 적용하기 위해서 수정된 제어 흐름 그래프를 소개한다. 이를 통해 테스터는 동시성 경로에 대한 테스트 시나리오를 생성할 때 경로 폭발 문제를 발생시키지 않고 일반적인 공통로를 다루지 않는 테스트 시나리오를 생성할 수 있다.

그러나 액티비티 다이어그램의 반복과 관련된 부분은 아직 위험을 가지고 있고 논리적으로 실행 불가능한(infeasible) 경로에 대해서 제거하는 기술이 필요하고 만들어진 테스트 시나리오에 적합한 테스트 데이터를 뽑아 테스트 케이스를 생성하는 기법이 필요하다. 향 후에는 본 논문의 테스트 시나리오 생성 기법을 적용한 테스트 케이스 자동 생성 방법을 모색하고, 이를 지원하는 도구를 제공할 계획이다.

Acknowledgement

이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보・컴퓨팅기술개발사업(NRF-2014M3C4A7030504)과 서울시의 재원으로 수행한 서울시 창조전문인력 양성사업(CAC15106)의 지원을 받아 수행된 연구임.

참고문헌

[1] G. J. Myers, C. Sandler, and T. Badgett, The Art of Software Testing, John Wiley & Sons, pp. 41-84, 2011.

[2] Lu. Shan, P. Soyeon, S. Eunsoo, and Z. Yuanyuan, "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics," ACM Sigplan Notices, vol. 43, no.3, pp. 329-339, 2008.

코드 중복 없이 안드로이드 컴포넌트에 예외 처리를 추가하는 클래스 설계 방법에 관한 연구

김현순 이승휘 이화중 최광훈

연세대학교 컴퓨터정보통신공학부
강원도 원주시 흥업면 연세대길 1
{coli,sunkus711,hj_tl22,kwanghoon.choi}@yonsei.ac.kr

창병모

숙명여자대학교 컴퓨터과학과
서울 용산구 청파로 47 길 100
chang@sookmyung.ac.kr

요약: 안드로이드 앱의 취약점으로 비정상 종료되는 경우 발생하는 예외를 처리하기 위해 안드로이드 컴포넌트 구조에 적합한 예외처리 방법이 필요하다. 안드로이드 컴포넌트 기반 예외처리 방법에 관한 이전 연구에서 해결책을 제시한 바가 있으나 유사한 컴포넌트 클래스들이 동일한 코드를 중복해서 갖는 문제가 있었다. 이 논문은 중복 코드 없이 안드로이드 컴포넌트에 예외처리 기능을 추가한 클래스를 설계하는 방법을 제안한다.

핵심어: 안드로이드, 예외, 클래스 설계, 디자인 패턴

1. 서론

안드로이드는 모바일 디바이스를 위한 구글의 오픈소스 플랫폼이다. 안드로이드 앱은 안드로이드 플랫폼 API 를 사용하는 자바 프로그램이다. 네 가지 유형의 안드로이드 컴포넌트, 액티비티, 서비스, 브로드캐스트리시버, 컨텐트프로바이더로 구성된다.

안드로이드 앱의 취약점으로 빈번하게 비정상 종료되는 사례에 대응하고자 안드로이드 컴포넌트 구조에 적합한 예외 처리 방법에 관한 연구[1]가 있었다. 이 연구에서 안드로이드 컴포넌트 클래스를 확장하여 안드로이드 플랫폼과 앱의 경계에 방어벽을 만들고, 앱에서 발생한 예외가 플랫폼으로 전달되어 비정상 종료되는 것을 막는 방법을 제안하였다.

그림 1 에서 안드로이드 액티비티 컴포넌트 클래스(Activity, ListActivity, PreferenceActivity)에 예외 기능을 추가한 새로운 컴포넌트 클래스(ExceptionActivity, ExceptionListActivity, ExceptionPreferenceActivity)간의 클래스 다이어그램을 보여준다. 기존 방식에서 Activity, ListActivity, PreferenceActivity 클래스를 상속받아 앱의 액티비티를 만들지만, 안드로이드 예외를 고려한 액티비티를 사용하는 방식에서는 Exception-으로 시작하는 해당하는 클래스를 상속받아 액티비티를 만든다.

하지만 안드로이드 예외 클래스 라이브러리[1]에서 동일한 유형의 컴포넌트 클래스들에서 앱과 플랫폼 사이의 방어벽에 해당하는 코드를 제대로 공유하지 못하고 각 클래스마다 방어벽 코드를 중복해서 포함시켜야 하는 문제가 있다.

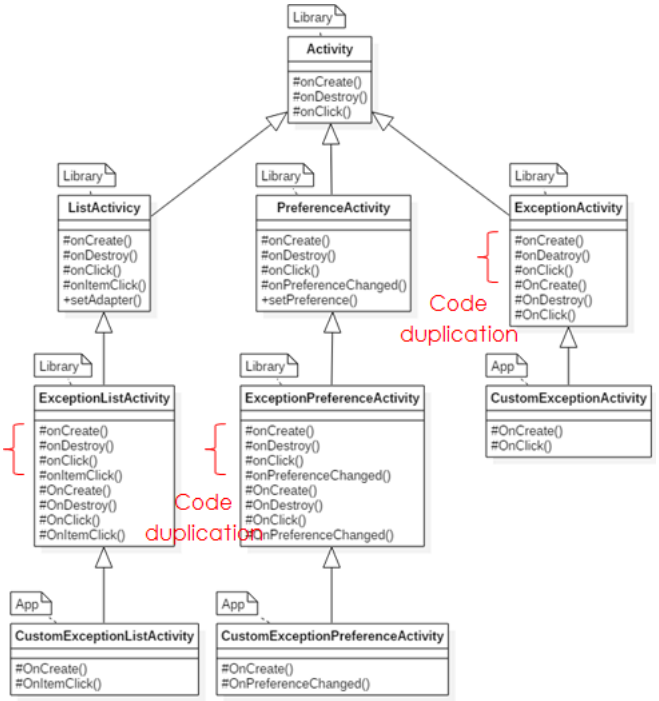


그림 1 클래스 다이어그램

코드 중복 문제가 발생한 이유는 기존 안드로이드 컴포넌트 클래스들의 상속 관계가 예외를 추가한 컴포넌트 클래스들 사이의 상속 관계로 유지되지 않는 방법으로 설계했기 때문이다. 즉, Activity 클래스를 상속받아 예외 기능을 추가해서 ExceptionActivity 클래스를 정의하고, Activity 클래스의 자식 클래스 ListActivity 를 상속받아 예외 기능을 추가하여 ExceptionListActivity 클래스를 정의했지만[1], 새로 정의한 두 예외 클래스는 서로 상속 관계가 아니다. 따라서 ExceptionActivity 에 포함된 방어벽 코드를 ExceptionListActivity 에서 공유하지 못하고 중복해서 코드를 포함시켰다. 이와 유사한 상황이 PreferenceActivity 뿐만 아니라 Activity 의 모든 자식 클래스에 대해서도 벌어진다.

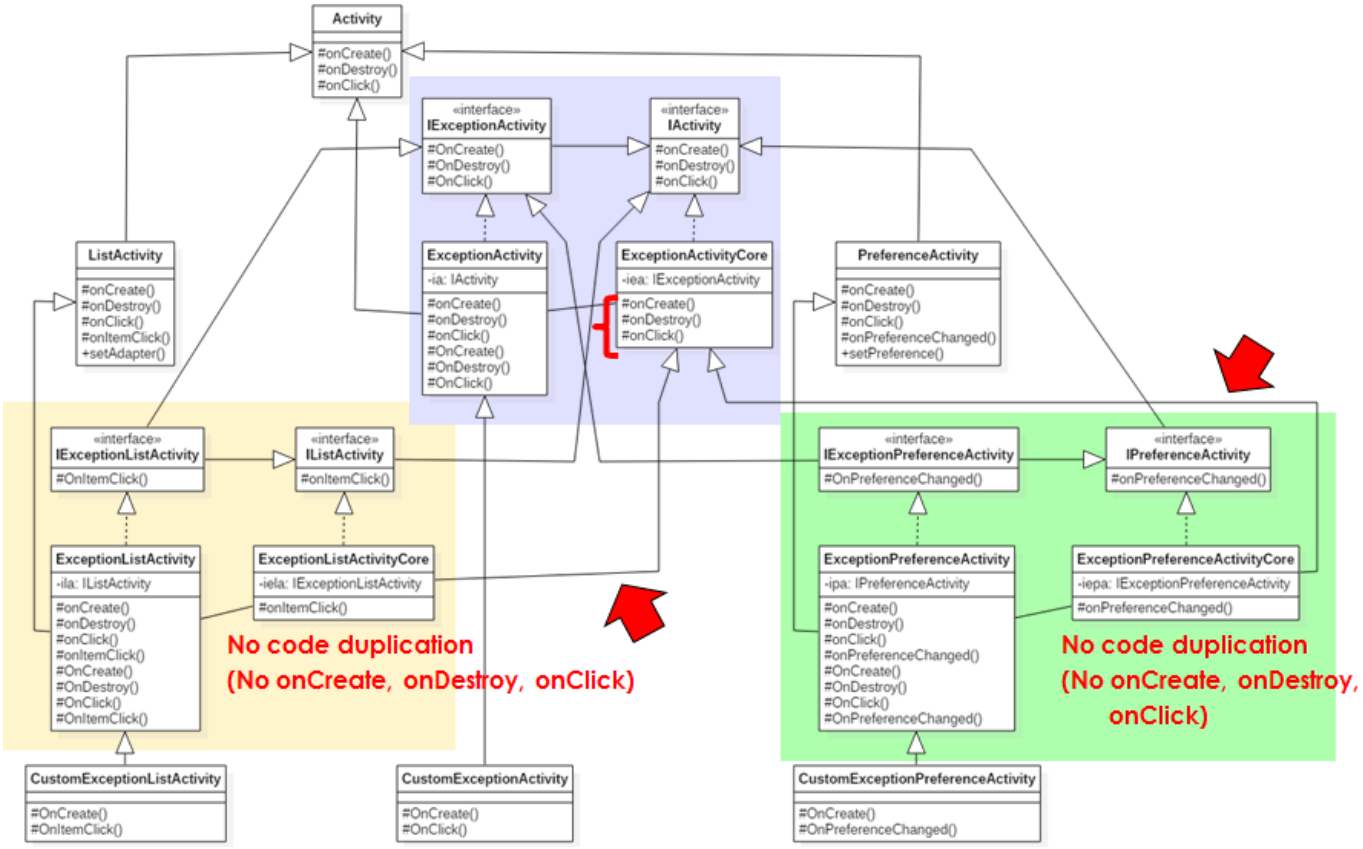


그림 2 새로운 방법으로 설계한 클래스 다이어그램

이 논문은 안드로이드 예외 클래스에서 앞에서 설명한 중복된 코드가 없는 설계 방법을 제안한다.

2. 안드로이드 컴포넌트 예외 기능을 공유하는 클래스 설계 방법

앞에서 설명한 코드 중복 문제를 해결하면서 안드로이드 컴포넌트에 예외 기능을 확장하는 클래스 설계 방법을 제안한다.

- 안드로이드 컴포넌트 클래스를 예외 처리 기능을 구현한 핵심(Core) 클래스와 이를 사용하는 껍데기(Shell) 클래스로 나눈다.
- 두 클래스를 서로 연관 관계(association)로 구성하여 각 클래스의 객체들을 합하면 하나의 객체처럼 동작하도록 구성한다.
- 기존 클래스를 상속받아 껍데기 클래스를 정의하고, 핵심 클래스들끼리 서로 상속 관계를 구성한다. 이로써 코드 중복 문제를 해결한다.

그림 2 는 이 아이디어에 따라 설계한, 예외 기능이 추가된 안드로이드 클래스 다이어그램이다. ExceptionActivityCore 클래스에 정의된 예외 처리 기능을 두 개의 하위 예외 (핵심) 클래스들이 그대로 상속 받아서 코드 중복 문제를 해결하였다. 자세한 구성은 확장 버전의 논문[2]을 참고한다.

3. 결론

이 논문에서 안드로이드 컴포넌트에 예외처리를 확장한 클래스를 설계하는 새로운 방법을 제안하여 기존 방법의 코드 중복 문제를 해결하였다. 기존의 예외 클래스[1]를 제안한 방법의 예외 클래스로 대체하면 기존 앱을 수정하지 않고 사용할 수 있다[2].

참고문헌

[1] K. Choi, B-M Chang, A Lightweight Approach to Component-level Exception Mechanism for Robust Android Apps, Computer Languages, Systems, and Structures, Vol.44, Part C, p.283-298, December 2015.

[2] 김현순, 이승휘, 이화중, 최광훈, 창병모, 코드 중복 없이 안드로이드 컴포넌트에 예외 처리를 추가하는 클래스 설계 방법에 관한 연구, 2016 년 1 월. http://mobilesw.yonsei.ac.kr/paper/KCSE2016.pdf

--

이 논문은 2014 년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No.NRF-2014R1A1A2053446).

레거시 소프트웨어의 품질 개선을 위한 역방향 Tactic 분석 중심의 아키텍처 평가

류동국
소프트웨어공학
엑스퍼트그룹
ryudwig@swexpertgroup.com

김재선
소프트웨어공학
엑스퍼트그룹
jskim@swexpertgroup.com

김진태
소프트웨어공학
엑스퍼트그룹
jtkim@swexpertgroup.com

요약: 본 논문은 레거시 소프트웨어에서 발생된 비기능 품질 저하 문제를 분석하기 위해 ATAM (Architecture Trade-off Analysis Method)을 응용하여 적용한다. 이 때 레거시 소프트웨어의 개발 산출물 미흡함으로 인해 ATAM 적용이 어려운 문제는 "정적 분석 기반의 역공학"과 "역방향 Tactic 분석"기법을 제안하여 해결한다. 본 연구 기법은 실제 레거시 소프트웨어의 품질 저하 문제 발생 건으로 인한 아키텍처 평가 컨설팅 사례를 중심으로 소개한다.

핵심어: 레거시 소프트웨어(Legacy Software), 소프트웨어 아키텍처 평가(Software Architecture Evaluation), 정적 분석(Static Analysis), 역공학(Reverse Engineering), ATAM(Architecture Trade-off Analysis Method)

1. 연구 배경

소프트웨어 제품의 배포 이후 실 설계 당시 예상치 못한 다양한 비기능적 품질 저하 문제들이 발생 가능하다. 이들 문제들의 원인을 정확히 파악하고 개선하기 위해서 현행 소프트웨어의 아키텍처를 평가하는 것이 필요하다. 아키텍처 평가 방법으로 ATAM (Architecture Tradeoff Analysis Method)[2]이 많이 사용되고 있으나, 레거시 소프트웨어를 대상으로 적용 시 어려움이 발생 가능하다. ATAM 은 아키텍처 문서 등이 사전에 준비되지 않으면 그 수행이 어려운 방법이기 때문이다. 또한, 우리나라의 많은 소프트웨어 기업들은 개발 시 체계적인 아키텍처 설계 과정을 수행하지 않음으로 인해, 사후 아키텍처의 평가는 더 큰 어려움을 겪게 된다.

2. 관련 연구

2.1 ATAM(Architecture Trade-off Analysis Method)

본 기법은 SEI(Software Engineering Institute)가 1998 년 제안한 연구 기법으로서 소프트웨어의 상세 구현 전에 아키텍처 설계가 비기능 품질 요구사항을 달성 가능한지를 검증하기 위한 방법이다[2]. 아키텍처가 품질 속성을 만족시키는지 평가하고, 품질속성들 간의 절충 관계(Trade-off)를 분석한다.

2.2 정적 분석 기반의 역공학

정적 분석(Static Analysis)은 프로그램 실행 없이 소스 코드만으로 소프트웨어를 분석하는 기법이다[5]. 특히, 역공학은, 대상 소프트웨어를 기반으로 높은 수준의 추상화 모델을 만들기 위해 분석하는 기술을 말한다[3]. 코드 내 선언되어 있는 모듈 목록, 각 모듈 간 상호 호출/참조 관계, 모듈의 크기/분포 등 소프트웨어를 상위 수준에서 추상화 한다.

3. 레거시 소프트웨어의 아키텍처 평가 방안

본 연구의 제안 기법은 "정적분석 기반의 역공학", "역방향 Tactic 분석", "ATAM" 3 가지의 approach 로 구성 된다. 이들 3 가지 approach 간의 흐름으로서 구성된 전체 기법은 아래 그림과 같다. 이를 "역방향 Tactic 분석 중심의 아키텍처 평가"라고 한다.

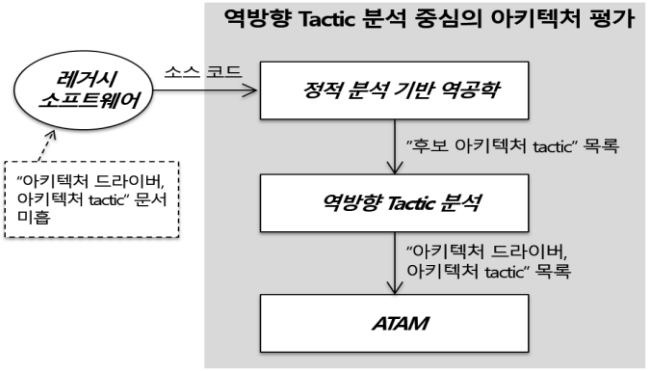


그림 1 본 연구 제안 기법의 구성요소

첫째, "정적 분석 기반의 역공학"을 통해 레거시 소프트웨어의 소스 코드를 분석하여 추상화 모델을 만든다. 정적 분석 도구를 이용한 자동 분석과 코드 리뷰를 통한 매뉴얼 분석을 통해서 레거시 소프트웨어 내에서 발견되는 다양한 동적/정적 구조들(모듈의 목록, 명칭, 개수, 참조/호출 관계, 메서드 호출 시퀀스 등)을 추출한다. 이를 통해 레거시 소프트웨어의 구축 시 적용했을 것으로 예상되는 아키텍처 tactic 목록, 즉 "후보 아키텍처 tactic 목록"을 추출한다.

둘째, "역방향 tactic 분석"은 "후보 아키텍처 tactic 목록"을 기초로 수행된다. "후보 아키텍처 tactic 목

록" 안에서 개발 당시 의도적으로 채용했던 tactic 이라면 달성하고자 했던 비 기능 요구사항이 있을 것이다. 이는 해당 tactic 이 실제로 소프트웨어 구축 시 적용되었다는 것을 입증할 수 있는 근거를 제공할 수 있다. 레거시 소프트웨어의 개발 관계자들과의 인터뷰를 통해 tactic 별 관련 비 기능 요구사항을 도출한다. 인터뷰 시 레거시 소프트웨어 분석가는 해당 tactic 의 내용, 해당 tactic 이 일반적으로 달성하고자 하는 품질 속성 등에 대해서 설명을 해주어야 한다. 이를 통해 개발 관계자들이 개발 당시의 관련 요구사항들을 쉽게 도출하는 데에 도움을 줄 수 있다. 후보 아키텍처 tactic 목록 중 관련 비 기능 요구사항과 연관되지 않는 tactic 들은 제거되고, 남아 있는 tactic 목록이 최종 목록이 된다. 도출된 비 기능 요구사항들 간 우선 순위 분석 작업 후에 결과적으로, "아키텍처 드라이버 목록"과 "아키텍처 tactic 목록"이 도출된다.

셋째, "역방향 tactic 분석" 작업의 결과 산출물은 바로 ATAM 수행 시 입력 정보로 이용하게 된다.

결과적으로 ATAM 적용 시 필요한 입력 정보인 "아키텍처 드라이버, 아키텍처 tactic 목록"이 미흡한 레거시 소프트웨어를 대상으로 하여, 아키텍처 평가를 수행할 수 있도록 한다. 평가의 최종 결과로는 레거시 소프트웨어의 비 기능 품질 저하 문제의 원인을 확인할 수 있게 된다.

4. 역방향 tactic 분석 중심의 아키텍처 평가 적용 사례

본 논문 기업은 소프트웨어공학 컨설팅 기업으로서 고객 A 사의 특정 솔루션 제품에 대한 아키텍처 평가를 의뢰 받아 수행한 실제 사례를 중심으로 제안 기법을 소개하고자 한다. 사례에 적용된 연구 기법은 아래 그림과 같이 크게 5 단계의 절차들로 수행되었다.

실제 적용 사례를 통해, 의뢰 기업이 찾고자 하는 성능 및 가용성 측면에 음(-)의 영향을 미치는 다양한 tactic 을 정확히 예측할 수 있었다. 이를 기준으로 성능, 가용성 문제점으로 최종 보고 되었다. 아키텍처 평가 이후 의뢰 기업은 이를 최종 확인하기 위해 레거시 소프트웨어에 대해 다양한 도구를 활용한 동적 성능 테스트와 프로파일링을 수행하였으며, 아키텍처 평가에서 예측된 대부분을, 실제 동작 시 확인할 수 있었다.

결국, 그 동안 정보의 부족으로, 블랙박스로 취급되어온 레거시 소프트웨어의 아키텍처의 주요 부분들이 재건되고 아키텍처 차원의 논리적, 정량적 평가가 가능해졌다. 이번 아키텍처 평가를 통해, 다양한 품질속성에 대한 해결방안 및 개선점을 확인하고 지속 또는 개선해 나갈 수 있는 것이다.

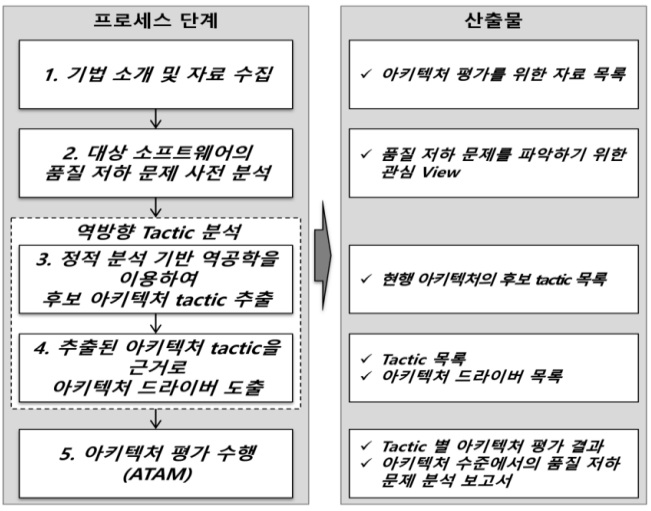


그림 2 역방향 tactic 분석 중심의 아키텍처 평가 절차와 각 단계 별 산출물

5. 결론

본 연구에서 제안하는 기법은 첫째, 명시적 아키텍처가 없거나 현행화 되지 않은 레거시 소프트웨어의 아키텍처 추출 및 분석 방법을 제시한다. 둘째, 추출된 레거시 소프트웨어의 아키텍처에 대한 평가 방법을 제시한다. 셋째, 레거시 소프트웨어의 품질저하 문제 해결을 위한 아키텍처 분석 및 평가 적용 방법을 제시한다.

현행 소프트웨어로부터 아키텍처 Tactic 식별과 Tactic 의 driver 추출 작업의 결과물은 아키텍처 평가자의 역량에 따라 그 정확성에 편차가 있을 수 있다. 이를 해결하기 위해 상세한 프로세스를 정의하는 보완 연구가 필요할 것이다.

참고문헌

[1] Clements, Paul C. Software architecture in practice. Diss. Software Engineering Institute, 2002.

[2] Kazman, Rick, et al. "The architecture tradeoff analysis method." Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings. Fourth IEEE International Conference on. IEEE, 1998.

[3] Chikofsky, Elliot J., and James H. Cross. "Reverse engineering and design recovery: A taxonomy." Software, IEEE 7.1 (1990): 13-17.

[4] Cousot, Patrick, and Radhia Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints." Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. ACM, 1977.

[5] Nielson, Flemming, Hanne R. Nielson, and Chris Hankin. Principles of program analysis. Springer, 1999.

논리적 UI 모델의 MVC 패턴으로의 매핑[1]

김세화, 최기봉, 이승엽

한국외국어대학교 정보통신공학과
경기도 용인시 처인구 모현면 외대로 80
ksaehwa@hufs.ac.kr, chnaru@naver.com, yup@mojisoft.co.kr

요약: 본 논문에서는 논리적 UI(User Interface, 사용자 인터페이스) 모델을 MVC (Model, View, Controller) 아키텍처 패턴으로 어떻게 매핑할 수 있는지에 대해 제시한다. 논리적 UI 모델은 PELUM(Pattern and Event Logical User Interface Modeling, 패턴 및 이벤트 기반 논리적 사용자 인터페이스 모델링)을 기반으로 한다. PELUM 은 UI 중심적인 내장형 시스템을 UI 모델링을 통해 효율적으로 개발하기 위한 방법이다. PELUM 은 논리적 UI 모델을 통해 자동으로 코드를 생성해주는 도구를 제공하고 있으나 사용자가 이렇게 생성된 코드를 맞춤형으로 개선하기 위해서는 논리적 UI 모델의 MVC 패턴으로의 매핑에 대한 이해가 필요하다. 본 논문에서 제시한 결과는 또한 MVC 패턴에 대한 이해도를 높여 UI 중심적 소프트웨어의 재사용 가능성과 유지 보수성을 극대화 하는데 활용될 수 있다.

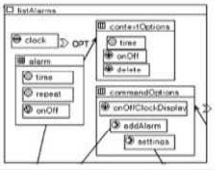
핵심어: 사용자 인터페이스 (UI), 모델 기반 개발, MVC 패턴, 논리적 UI 모델링, 코드 자동 생성.

1. 서론

지능형 자동차, 스마트 폰, 스마트 TV 등 내장형 기기들이 고도화됨에 따라 이들에 탑재될 내장형 소프트웨어를 여러 기기에 적용시키며 이를 재사용할 필요성이 급격히 커지고 있다. 또한 이러한 내장형 소프트웨어가 UI(User Interface) 중심적인 형태로 고도화됨에 따라 다양한 내장형 기기에 적용 시 이를 재사용하기 보다는 다시 새로 개발하는 경우가 대부분이다. 이러한 어려움을 해결하기 위하여 우리는 [1]에서 PELUM(Pattern and Event based Logical User Interface Modeling)을 제안 하였다. PELUM 은 논리적 사용자 인터페이스 모델 (Logical UI Model, LUM)을 포함하여 계층적인 내장형 소프트웨어 아키텍처를 제공할 뿐만 아니라, LUM 을 설계하는 모델링 도구[2]와 LUM 을 기반으로 HTML5 기반 응용 소프트웨어(모바일 웹 앱)를 자동으로 생성해 주는 도구[3]를 포함한다.

UI 중심적 소프트웨어를 효율적으로 개발하기 위한 관련 도구로서는 Balsamiq [4], InVision [5], Appery [6], mBizamaker [7] 등이 많이 사용되고 있다. Balsamiq 과 InVision 은 UI mock-up 도구로서 널리 사용되는 UI 컨트롤 위짓들을 Drag & Drop 으로 빠른 시간에 구성할 수 있게 도와주는 도구이다. Appery 와 mBizmaker 는 이에 더해 화면 전환에 대해 명확히 기술하여 사용자가 시연 할 수 있도록 동작 가능한 앱을 생성해준다. 그러나 사용자에게 직접 코드를 제공하지 않아 유연성이 크게 떨어지는 단점이 있다. PELUM 은 특정 플랫폼에 국한된 UI 컨트롤 위짓이 아니라 추상화된 논리적 UI 메타 모델을 통해 논리적 UI 를 구성하게 한다. 이를 통해 다양한 기기와 다양한 플랫폼에 유연하게 매핑될 수 있는 재사용 가능한 UI 모델을 관리할 수 있게 한다. 또한 이미 정의된 패턴이나 새롭게 정의된 패턴을 활용하여 자동으로 동작 가능한 코드를 생성하고 이를 사용자가 직접 수정할 수 있다.

표 1. PELUM 의 4 계층 아키텍처.

PELUM 계층적 아키텍처	Example (Alarm)
Graphical Resource Model (GRM)	
UI Controls and Layout Model (CLM)	
Logical UI Model (LUM)	
Programming Interface Model (PIM)	

[1] 이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2013R1A1A3006819).

본 논문에서는 PELUM 의 논리적 UI 모델인 LUM 이 MVC(Model, View, Controller) 아키텍처 패턴[8]에 어떻게 매핑될 수 있는지에 대해 제시한다. 본 논문에서 제시한 연구 결과는 PELUM 모델링 및 모바일 웹 앱 자동 생성 도구로부터 생성된 구현을 사용자가 직접 코드 수정을 통해 맞춤형 제작을 하고자 할 때 도움이 될 수 있다. 또한 MVC 패턴에 대한 이해도를 높여 UI 중심적 소프트웨어의 재사용 가능성과 유지 보수성을 극대화 하는데 활용될 수 있다.

본 논문의 나머지는 다음과 같이 구성되어 있다. 2 장에서는 본 논문의 배경이 되는 PELUM 과 MVC 패턴에 대해 설명한다. 3 장에서는 PELUM 의 LUM 및 기타 계층 요소가 MVC 아키텍처 패턴에 어떻게 매핑되는지 논한다. 4 장에서는 사례 연구를, 5 장에서 결론을 맺는다.

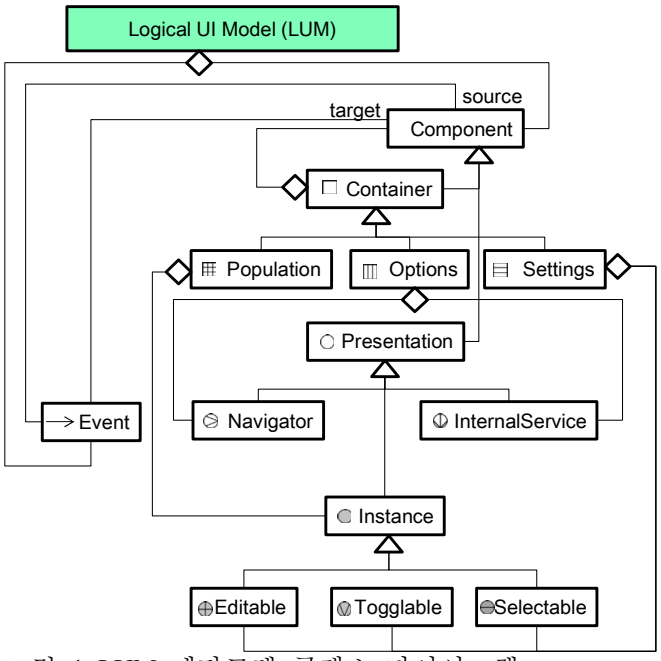


그림 1. LUM 메타모델 클래스 다이어그램.

2. 연구 배경

2.1 PELUM (Pattern and Event Based Logical UI Modeling)

PELUM 은 표 1 과 같이 4 개의 계층적 모델을 기반으로 하고 있다. 첫째, PIM(Programming Interface Model)은 Android API(Application Programming Interface)나 iOS API 와 같이 특정 플랫폼에 따라 API 로 주어지거나 사용자가 제공하는 데이터 클래스나 기타 하부 라이브러리를 포함하는 계층이다. 둘째, LUM(Logical UI Model)은 추상적인 UI 요소들을 이벤트와 함께 모델링 하는 계층이다. 셋째, CLM(UI Control and Layout Model)은 UI 요소들이 버튼이나 체크 박스 등 특정 UI control 위젯으로 매핑되면서 특정 레이아웃에서 어디에 위치 할 것인지 결정하는 계층이다. 마지막으로 GRM(Graphical Resource Model)은 각 UI 요소에 그래픽 자원을 제공하는 계층이다.

그림 1 은 LUM 메타모델에 대한 클래스 다이어그램을 보여준다. 그림 1 에서와 같이 LUM 은 크게 Component 와 Event 로 구성되며, Component 의 종류로는 Container 와 Presentation 이 있다. Container 는 Container 나 Presentation 을 포함한다. Event 는 두 Component 를 source 와 target 의 관계로 연관짓는다. Container 의 일종으로서 Population, Options, Settings 가 있으며, Presentation 의 일종으로 Navigator, InternalService, Instance 가 있다. Instance 의 일종으로서 다시 Editable, Selectable, Togglable 이 있다. Population 은 Instance 로 구성되며, Options 는 Navigator 나 InternalService 로 구성된다. 또한 Settings 는 Editable, Togglable, 또는 Selectable 로 구성된다.

표 2. PELUM 논리적 UI 모델(LUM)의 구성 요소.

PELUM 모델링 요소	설명
Container	임의의 LUM 컴포넌트를 포함하는 일반적인 포함자
Population	PIM의 특정 클래스의 인스턴스들의 집합을 나타내는
Options	선택적인 서비스를 포함한
Settings	특정 구성사항을 설정하기 위한
Presentation	텍스트나 이미지를 표현하기 위한 표현자
Navigator	다른 컴포넌트(나)를 활성화시키는
Internal Service	PIM API에 연결된 내부서비스를 제공하기 위한
Instance	PIM에 속한 변수의 값을 나타내는
Editable	편집 가능한
Togglable	즉각적으로 설정을 키고 끌 수 있는
Selectable	미리 정의된 값들 중 특정 값을 고를 수 있는
Event	컴포넌트(나) 간의 방향성 있는 연관. 화살표의 시작 컴포넌트가 해당 이벤트를 받으면 도착 컴포넌트가 활성화된다.

표 2 는 각 LUM 메타모델 구성 요소들에 대한 간략한 설명을 보여준다. 먼저, Container 는 임의의 LUM 메타모델을 포함하는 일반적인 포함자이며 Population 은 PIM 의 한 클래스의 인스턴스들의 집합을 나타내는 Container 다. Options 는 모바일 웹 응용프로그램에서 선택적인 서비스를 포함하는 Container 이고 Settings 는 특정 구성사항을 설정하기 위한 Container 이다. 그 다음, 비-개체 표현자(Presentation)가 있다. Presentation 은 텍스트나 이미지를 표현하기 위한 표현자이다. 그리고 Navigator 는 다른 LUM 컴포넌트(Container 나 Presentation) 를 활성화 시키는 Presentation 이며. InternalService 는 PIM 의 API 에 통상 연결되어 있는 내부서비스

를 제공하기 위한 Presentation 이다. Instance 는 PIM 의 객체의 개체변수를 나타내는 Presentation 이다. Editable 은 편집 가능한 Instance 이며, Togglable 은 즉각적으로 설정을 켜고 끌 수 있는 Instance 이다. Selectable 은 미리 정의된 값들 중 특정 값을 고를 수 있는 Instance 이다. 마지막으로 Event→는 컴포넌트(Container□나 Presentation○)들 간을 연결하는 방향성 있는 연관으로서, 화살표의 시작 컴포넌트가 해당 이벤트를 받으면 도착 컴포넌트가 활성화되는 것을 모델링한다.

2.2 MVC(Model, View, Controller) 아키텍처 패턴

MVC 아키텍처 패턴은 그림 2 와 같이 표현된다. 그림 2 (a), (b) 모두 UI 소프트웨어의 아키텍처에서 데이터를 포함하는 모델 부분, 사용자에게 보이는 뷰 부분, 이 둘 모델과 뷰를 제어하는 컨트롤러 부분이 명확하게 분리된 형태로 소프트웨어가 구현되어야 한다는 것이 핵심이다. 그림 2 (a)는 전통적인 형태로서, 모델과 뷰를 컨트롤러가 제어하고, 컨트롤러가 사용자나 외부 디바이스, 혹은 외부 시스템으로부터 이벤트를 받고, 모델이 뷰에게 모델의 변화를 알리는 방식이다.

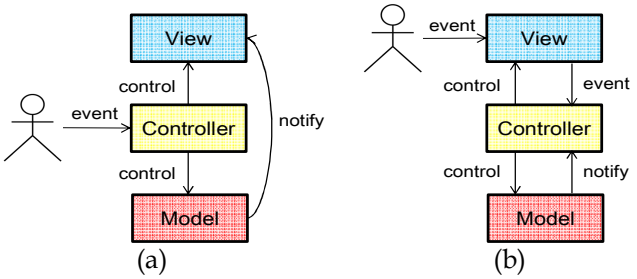


그림 2. MVC 패턴: (a) 전통적인 형태, (b) MVP(Model, View, Presenter)라고 지칭되는 MVC 패턴의 변형; Controller 가 Presenter 에 해당한다.

그림 2 (b)는 Microsoft 나 Google Chrome, Apple iOS 등 현재 상업적인 영역에서 널리 채택되는 MVC 패턴의 형태로서 Controller 를 Presenter 라고 지칭하기도 하여 MVP 패턴이라고도 지칭된다[9]. 여기에서는 모델이 직접 뷰에게 자신의 상태 변화를 알리지 않고 컨트롤러에게만 알린다. 또한 이벤트를 컨트롤러가 직접 받는 것이 아니라 뷰가 받아서 이를 컨트롤러에게 전달한다. 이 아키텍처의 장점은 view 와 model 이 완전히 분리되고, 이벤트를 뷰가 직접 받음으로써, 뷰를 최상위 계층으로 하고, 컨트롤러가 중간 계층, 모델이 최하위 계층이 되는 계층적인 아키텍처에 가깝게 된다는 점이다. 본 논문에서도 이러한 그림 2 (b)의 MVC 패턴에 대해서 초점을 맞춘다.

3. PELUM 의 MVC 패턴으로의 매핑

그림 3 은 PELUM 의 계층 구조 및 논리적 UI 모델 (LUM)의 각 구성 요소가 MVC 패턴으로 어떻게 매핑되는지를 보여준다. 표 1 에서와 같이 PELUM 의 계층적 아키텍처는 그래픽 리소스 모델 (GRM), UI 컨트롤 및 레이아웃 모델 (CLM), 논리적 UI 모델 (LUM), 프로그래밍 인터페이스 모델(PIM)로 이루어진다. 그림 3 에서와 같이 GRM 과 CLM 은 뷰에, PIM 은 모델에 완전히 매핑된다. 반면 LUM 은 뷰, 컨트롤러, 모델 모두에 매핑된다. MVC 패턴 구성 요소 중 컨트롤러는 전적으로 LUM 에 매핑되는 것은 주목할 만 하다.

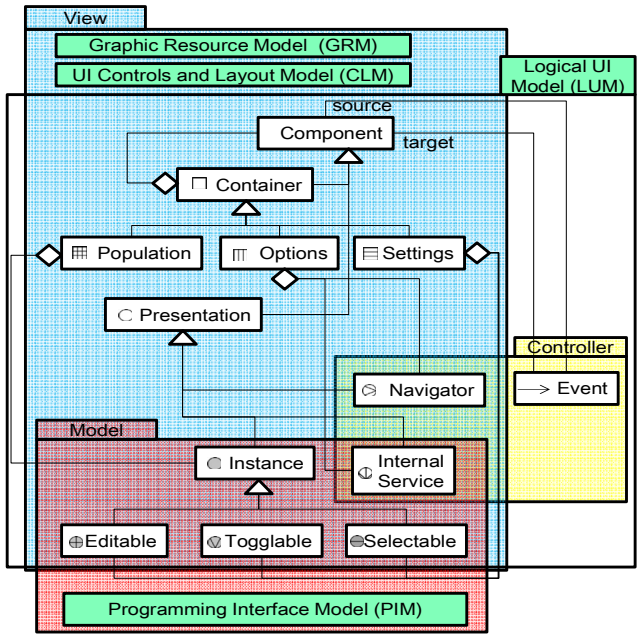


그림 3. PELUM 의 MVC 패턴으로의 매핑.

표 2 의 LUM 의 구성 요소는 그림 3 에서와 같이 Event 를 제외하고 모두 뷰에 매핑된다. Event 와 Navigator 와 Internal Service 는 컨트롤러에 매핑된다. Event 는 Container 나 Presentation 간의 활성화를 제어하며, Navigator 는 명시적으로 Event 의 source 로 연관되어야 하는 요소이다. InternalService 는 다른 뷰 Component 와의 연관 없이 이벤트를 받아 모델에 영향을 끼칠 수 있는 요소로서, LUM 상 Event 가 source 로서 모델링되지 않는다. 한편 Instance 와 이를 상속하는 Editable, Togglable, Selectable 은 모두 모델에 매핑된다. 한편 InternalService 또한 모델에 매핑된다. Instance 및 이를 상속하는 요소들은 모델의 개체 변수 또는 이들의 값을 얻는 get 메소드에 대응하며, InternalService 는 set 메소드나 모델의 변화를 일으키는 여러 메소드들에 대응한다.

이와 같이 모든 LUM 구성 요소와 PELUM 아키

텍처 요소가 MVC 패턴 구성 요소에 완전히 매핑되며, PELUM 의 계층적 아키텍처가 MVC 패턴에 매핑되면서 그대로 유지되는 것을 볼 수 있다. 이는 MVC 패턴에서 뷰가 최상층 계층, 모델이 최하층 계층, 컨트롤러가 중간 계층이 되는 것을 보여준다. 이는 그림 2 (b)의 MVP 패턴과 그 궤를 같이 한다.

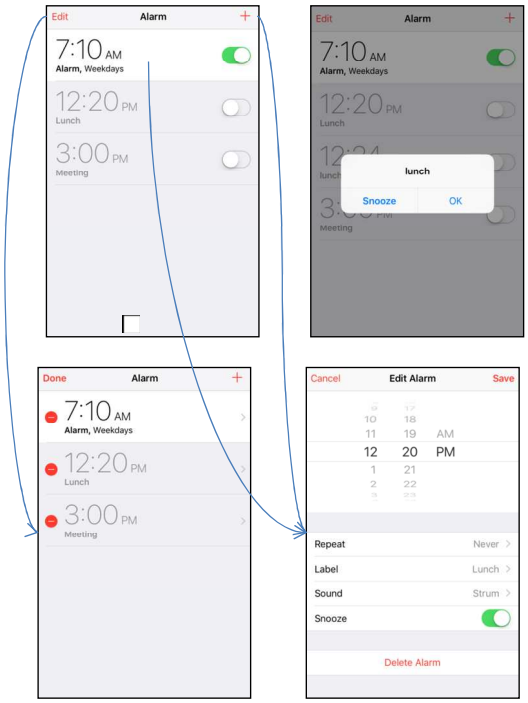


그림 4. 사례 연구: 스마트 폰 용 Alarm 앱의 UI mock-up 모델.

4. 사례 연구

사례 연구로서 스마트폰에 탑재되는 Alarm 앱에 대해 논한다. 그림 4 는 Alarm 앱의 UI mock-up 모델을 보여준다. UI mock-up 모델은 Balsamiq[4]과 이나 InVision[5]과 같은 도구를 통해 뷰 구성 요소와 화면 전환을 나타낸 모델이다. 그림 5 에서 UI mock-up 모델의 뷰 구성 요소가 LUM 의 구성 요소로 어떻게 매핑되는지를 표기하였다.

그림 6 은 그림 4 의 Alarm 앱이 LUM 으로 어떻게 모델링되는지를 보여준다. 그림 6 의 LUM 에서는 그림 4 의 UI mock-up 모델에서는 모델링되지 않는 다음과 같은 것들이 모델링된다.

- MVC 의 뷰 구성 요소
  - Options: 이 Container 안의 구성요소들은 MVC 의 컨트롤러인 Navigator 나 MVC 의 컨트롤러와 모델을 겸한 InternalService 등이다. (예) alarmList, alarmNotification, edit 의 menu.
  - Population: 이 Container 안의 구성요소들은 MVC 의 모델 구성 요소에서 get 메소드 호출 혹은 개체 변수의 값을 얻어서 값이 표현되는 것으로, Instance 등이다. (예) alarmList 의 alarm.
  - Settings: 이 Container 안의 구성요소들은 MVC 의 모델 부분에 대응하는 Editable, Togglable, Selectable 등이다. 이는 이 컨테이너의 source 가 되는 Population 의 내부 구성요소 Instance 들을 편집 가능한 형태로 확장한 것들이다. 통상 Settings 는 이를 source 가 되는 Population 의 내부 구성 요소보다 더 많은 구성 요소를 포함한다. (예) edit 에 포함된 sound, snooze 는 alarm 에는 없는 구성요소이다. 이는 LUM 모델로부터 MVC 의 모델 부분의 클래스 코드를 자동 생성할 수 있게 하는데 필수적인 모델링 요소이다.
  - Presentation: 이는 MVC 의 뷰에만 대응하는 것으로, MVC 의 모델에 대응하는 요소가 없음을 나타낸다. (예) alarmList 의 menu 의 alarmTitle, edit 의 menu 의 editTitle.

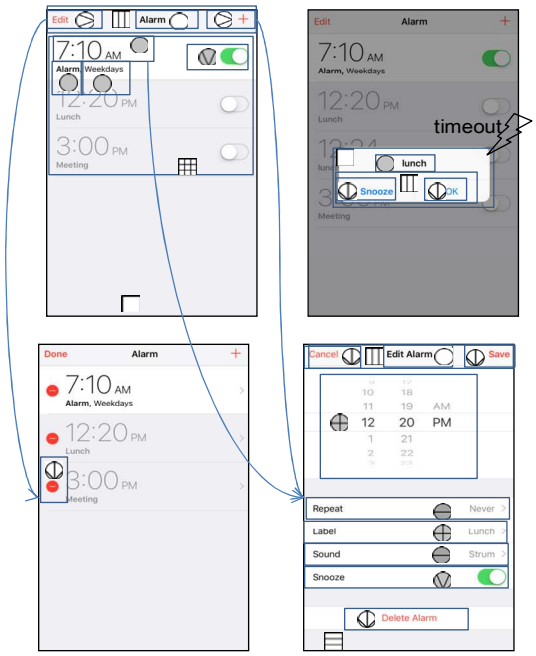


그림 5. 그림 4 에서의 각 뷰 구성요소에 대한 LUM 요소로의 매핑.

- MVC 의 컨트롤러 구성 요소
  - Event→: 이는 MVC 의 뷰 부분에 매핑되지 않는 LUM 의 유일한 구성 요소로서, MVC 의 컨트롤러에 매핑되어 뷰 구성 요소의 활성화를 표현한다. (예) timeout
- MVC 의 뷰 및 컨트롤러 구성 요소
  - Navigator: 이는 MVC 의 컨트롤러에 매핑되며서 뷰 부분도 있는 것을 표현한다.

Event→에 반드시 source 로서 연관되어야 한다. (예) alarmList 의 menu 의 edit

- MVC 의 모델 구성 요소
  - Instance : 이 Presentation은 MVC 의 모델 부분에 해당 개체 변수가 있음을 나타낸다. (예) alarmList 의 alarm 에 포함된 time
  - Editable, Togglable, Selectable: 이들은 편집 가능한 Instance 임을 나타낸다. (예) edit 에 포함된 sound.
- MVC 의 뷰, 컨트롤러, 및 모델 구성 요소
  - InternalService: 이는 MVC 의 뷰, 컨트롤러, 모델의 모든 구성요소에 해당한다. 컨트롤러로서 사용자의 입력을 받아들일 수 있으나 뷰 활성화와 직접 연관이 있을 필요는 없어서 Event→와 연관이 필수적이지 않다. MVC 의 모델 부분으로서는 MVC set 메소드나 기타 모델에 영향을 미치는 메소드가 연관되어 있음을 나타낸다. (예) alarmNotification 의 menu 의 snooze.

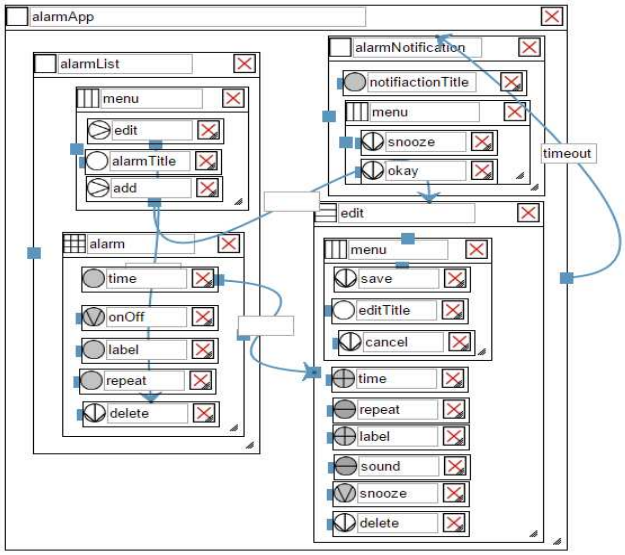


그림 6. 그림 4 의 Alarm App 에 대한 LUM. http://pelum.hufs.ac.kr 에서 제공되는 논리적 UI 모델링 도구에서 모델링 됨.

5. 결 론

본 논문에서는 PELUM(Pattern and Event Logical User Interface Modeling, 패턴 및 이벤트 기반 논리적 사용자 인터페이스 모델링) 아키텍처가 MVC (Model, View, Controller) 아키텍처 패턴에 어떻게 매핑 관계에 대해 제시하였다. PELUM 은 UI 중심적 내장형 소프트웨어를 다수의 기기에 적응하기 쉽게 하기 위하여 논리적 UI 모델(LUM)을 통해 소프트웨어를 개발하는 방법 및 그 도구를 포괄한다. PELUM 은 다른 UI mock-up 도구나 코드 자동 생성 도구와 달리 논리적 UI 모델을 통해 보다 유연성 있는 UI 중심적 내장형 소프트웨어 개발 방법을 제공한다.

PELUM 은 GRM, CLM, LUM, PIM 의 4 개의 계층으로 이루어지는데, 본 논문에서 GRM, CLM 이 뷰에, PIM 이 모델에, LUM 이 뷰, 컨트롤러, 모델을 아우름을 보였다. 또한 MVC 패턴의 각 구성요소에 어떻게 매핑되는지를 보임으로써, MVC 패턴을 보다 명확히 이해할 수 있는 기초를 제시하였다. 특히 뷰의 어느 구성 요소들이 모델이나 컨트롤러로 매핑될 수 있으며, 어떤 구성 요소는 온전히 컨트롤러로 매핑될 수 있는지를 보였다. 이러한 결과는 MVC 패턴에 대한 이해도를 높여 UI 중심적 소프트웨어의 재사용 가능성과 유지 보수성을 극대화 하는데 활용될 수 있다. 또한 PELUM 도구로부터 자동 생성된 코드를 사용자 맞춤형으로 수정할 때 활용될 수 있다.

참고문헌

[1] S. Kim, Pattern and Event Based Logical UI Modeling for Multi-Device Embedded Applications, Proceedings of International Conference on Convergence and Hybrid Information Technology, 2011.

[2] S. Kim, Graphical Modeling Environment for Logical User Interfaces Based on Eclipse GMF, Journal of Information Industrial Engineering, 2011.

[3] 최기봉, 김세화, 클라우드 기반 UI 모델링 및 모바일 웹 앱 자동 생성 도구, Korea Computer Congress (KCC), 2015.

[4] Balsamiq, https://www.balsamiq.com/

[5] InVision, http://www.invisionapp.com/

[6] Appery, https: // www.appery.io/

[7] mBizMaker, http://www.mbizmaker.com/

[8] Glenn E. Krasner, Stephen T. Pope, A cookbook for using the model–view controller user interface paradigm in Smalltalk-80. The JOT (SIGS Publications), Aug–Sep 1988.

[9] Microsoft Developer Network, The Model-View-Presenter (MVP) Pattern, https://msdn.microsoft.com/en-us/library/ff649571.aspx

소프트웨어의 계층 구조 파악을 위한 계층적 K-means 알고리즘

이선로, 이찬근

중앙대학교 컴퓨터공학부
서울 동작구 흑석로 84
ssunno; cglee@cau.ac.kr

요약: 소프트웨어 아키텍처의 모듈-뷰 복원에 이용되는 클러스터링 기법은 대개 평면적인 결과를 도출하며 계층적인 결과를 얻는 방법은 그 수가 적어 소프트웨어의 계층 구조를 파악하는데 제약이 따른다. 본 논문에서는 소프트웨어의 계층 구조를 파악하고 아키텍처 모듈-뷰 복원을 돕기 위해 K-means 알고리즘을 계층적으로 적용하는 방법을 제시한다. K-means 알고리즘의 계층적 적용을 통해 소프트웨어의 계층 구조를 다양한 방향에서 관찰할 수 있으며 소프트웨어의 모듈-뷰 복원에 활용할 수 있다.

핵심어: 계층적 클러스터링, K-means 알고리즘, 토픽 벡터

1. 연구배경

소프트웨어 아키텍처는 소프트웨어의 구조를 파악하고 관리하는데 중요한 지표로써 활용되고 있다. 소프트웨어가 진화함에 따라 문서가 진화 과정을 제대로 반영하지 못하면 소프트웨어의 아키텍처를 이해하는 것이 점점 어려워지게 된다. 이러한 문제를 해결하기 위해 리파지토리 또는 소스코드를 분석하여 아키텍처를 복원하는데 대부분의 소프트웨어 아키텍처 복원 과정은 클러스터링 기법을 활용하여 데이터를 분석하고 결과를 도출하여 복원 자료로써 활용하게 된다. 복원에 적용되는 클러스터링 기법과 목적에 따라 노드간 응집도나 결합도 등 객체지향 측면에서의 소프트웨어 구조를 파악하거나 아키텍처의 모듈-뷰를 복원하고 있다.

소프트웨어 아키텍처 복원에 사용되는 클러스터링 기법에는 ACDC, Bunch 등[1] 다양한 기법이 존재하지만 다수가 평면적인 결과를 도출하기 때문에 소프트웨어의 계층 구조를 파악하는데 어려움이 있고 계층적인 결과를 도출하는 방법은 그 수가 적어 다양한 관점에서 살펴보기 힘들다. 본 연구에서는 소프트웨어의 계층 구조를 파악하기 위해 평면적인 결과를 도출하는 K-means[2] 클러스터링 알고리즘을 계층적으로 수행하면서 결과를 도출하고 소프트웨어의 계층 구조를 복원하는 방법을 제안한다.

2. 기본 개념

2.1 K-means 알고리즘

K-means 알고리즘은 데이터를 여러 개의 그룹으로 나누는 클러스터링 기법이다. N 개의 노드가 주어졌을 때 K(K<=N)개의 그룹으로 노드를 분할하며 노드의 유사도(similarity)나 거리(distance)가 분할의 기준이 되고 그룹의 중심점(centroid)과 그룹 내 노드들의 거리(또는 비유사도 등)가 최소가 되는 그룹을 찾는 것을 목표로 한다. 그림 1 은 K-means 알고리즘을 2 차원 공간에서 수행한 결과이다. 4 개의 초기 중심점이 K-means 알고리즘을 통해 클러스터의 중심부로 옮겨지고 노드들이 가까운 클러스터에 배속된 것을 확인할 수 있다.

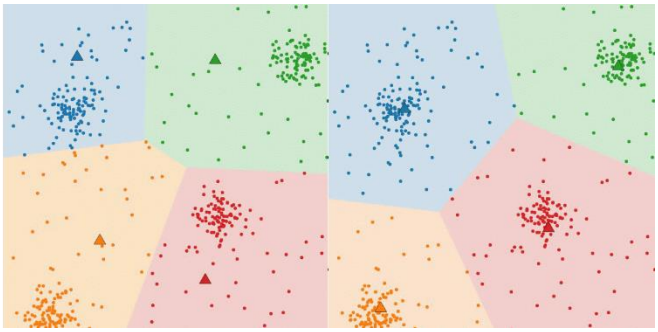


그림 1. K-means 알고리즘 수행 전(좌)과 후(우)

표준 알고리즘은 다음과 같다. 클러스터 C_i 의 중심을 $\boldsymbol{\mu_i}$, C_i 에 속하는 점의 집합을 S_i 라고 할 때, 분산 $V = \sum_{i=1}^{k} \sum_{i \in S_i} |\boldsymbol{x_j} - \boldsymbol{\mu_i}|^2$ 이고 알고리즘에서는 V 를 최소로 하는 집합 S_i 를 찾는다. 데이터를 K 개의 클러스터로 묶기 위해 각 클러스터의 초기 중심점 $\boldsymbol{\mu_i}$를

선택한다. 그 다음 초기 중심점을 시작으로 클러스터 설정, 클러스터 중심점 재 조정의 과정을 반복해 수행한다. 클러스터 설정 단계는 각 노드와 $\boldsymbol{\mu_i}$ 까지의 거리(또는 유사도)를 계산해 가장 가까운 클러스터에 노드를 배속한다. 그 다음 각 클러스터에 속한 노드들의 중심을 계산해 클러스터의 새로운 $\boldsymbol{\mu_i}$로 설정한다. 이 과정을 노드가 소속된 클러스터가 바뀌지 않거나 중심점의 위치가 변하지 않는 상태로 수렴하게 될 때까지 반복하여 클러스터링을 수행한다.

2.2 토픽 벡터

본 연구에서 사용된 토픽 벡터는 소프트웨어의 소스코드에서 지정된 개수의 토픽 분포를 추출하고 소스코드들 간 토픽의 일치성을 수치화한 것이다. 소프트웨어 아키텍처에 따라 소스코드 파일은 소속된 패키지의 개발 컨셉트를 따르게 되며 코드를 이루고 있는 단어나 주석들이 해당 패키지의 주제를 반영하고 있기 때문에 이를 이용해 구문적 일치성을 측정하면 소스코드 간의 연관성이 얼마나 높은지 파악할 수 있다.

소스코드에서 토픽 분포를 추출하기 위한 방법으로 LDA (Latent Dirichlet Allocation)[3]를 이용한다. LDA 는 문서에서 토픽이 나타날 확률과 단어가 토픽에 포함되는 확률이 디리클레 사전확률(Dirichlet Prior)을 따르도록 설정된 확률 모형이다. 문서에서 특정 토픽에 속하는 단어가 많이 등장하면 문서가 해당 토픽과 연관성이 있는 것으로 판단하며 해당 기법을 소스코드 내 변수, 클래스, 주석, 메서드 등 구문적 정보에 적용하여 소스코드 간 토픽의 일치성을 추출해낸다. 개발자가 명명하는 위의 단어들은 소스코드의 기능과 관련된 어휘로 구성되기 때문에 토픽 추출에서 연관된 기능을 가진 소스코드는 같은 토픽에서 공통적으로 높은 수치를 기록하게 된다.

토픽 벡터를 추출하기 위해 여기에서는 소프트웨어의 구문적 품질 측정 기법[4]에서 사용된 토픽 추출 방법을 이용했다. 토픽 추출에 사용된 도구인 *MALLET*[1]은 머신 러닝 도구로 문서를 LDA 분석하는 기능을 제공한다. 본 논문에서 토픽 분포를 얻기 위해 사용한 설정은 다음과 같다.

- $|T| = 100$ (토픽 개수)
- $\alpha = 0.25$ (문서 당 토픽 추출 매개변수)
- $\beta = 0.1$ (토픽 당 단어 추출 매개변수)
- iteration = 1000 (반복 횟수)

위의 설정을 바탕으로 문서에서 $|T|$개의 토픽을 추출하며 이를 수치화 하여 토픽 벡터로 나타낼 수 있다.

2.3 K-means 알고리즘의 계층적 적용

K-means 알고리즘은 평면적인 클러스터링 결과를 도출한다. 알고리즘이 비교적 간단하며 결과를 도출하는 속도가 빠르지만 데이터가 계층 구조를 가지는 경우 클러스터링 결과에서 관련 정보가 누락될 수 있다.

K-means 알고리즘을 계층적으로 적용하기 위한 단계는 다음과 같다. 클러스터링에 적용될 최대 계층 레벨 N 을 입력하는데 이 값은 어느 정도의 레벨까지 클러스터링을 수행할 지 결정하게 된다. 첫 단계에서 입력된 데이터를 대상으로 클러스터링을 수행하여 K 개의 클러스터 그룹으로 나누고 클러스터 각각에 대해 클러스터링을 다시 적용하여 두 번째 단계의 클러스터 그룹을 얻는다. 이 과정을 최대 N-depth 까지 반복하여 각 단계에서의 결과를 트리 형태로 구성해 계층적 클러스터링 결과를 얻는다.

그림 2. 계층적 클러스터링 적용 과정

K-means 알고리즘에서는 클러스터링을 수행하기 전에 데이터를 몇 개의 클러스터로 나눌지 정해야 한다. 평면적인 클러스터링의 경우 하나의 K 만 선정하면 되기 때문에 값을 입력받는 것이 가능하지만 계층적 클러스터링 과정에서 데이터가 어떻게 나누어질지 모르기 때문에 모든 클러스터에 대해 사용자에게 K 를 입력받는 방식은 사용하기 어렵다. 본 연구에서는 각 단계에서의 입력 데이터의 개수를 n_i 라고 할 때 클러스터 개수 K_i를 다음과 같이 계산한다.

$$K_i \approx \sqrt{n_i/2}$$

위의 수식에서 n<5 인 경우 K 가 1 이 되기 때문에 계층 레벨 N 을 과도하게 높게 설정하더라도 의미가 없다. 그림 2 에서 레벨 2 에서 작은 크기의 클러스터는 레벨 3 에서 더 이상 클러스터링을 수행하지 않는 것을 볼 수 있는데 작은 클러스터를 더 작은 단위로 나누면 클러스터에 노드가 하나만 배속되거나 의미

[1] http://mallet.cs.umass.edu

없는 단위로 분리될 수 있으므로 더 작게 나누는 것은 의미가 없다.

각 단계에서 클러스터링을 수행하려면 K 개의 초기 중심점 μ_i를 선택해야 한다. 초기화 기법에는 알려진 몇 가지의 방법이 존재하며 본 연구에서는 Forgy 알고리즘[5]을 사용해서 μ_i를 선택한다. Forgy 알고리즘은 데이터 집합에서 임의로 K 개의 데이터를 선택해 각 클러스터의 μ_i로 선택한다. 이 방법은 데이터 순서에 독립적이고 무작위로 선택하기 때문에 초기 클러스터가 분산되어 배정된다.

3. 실험 결과

3.1 실험 결과 분석

이 실험에서는 Hadoop 0.19.0 을 파일 수준에서 토픽 벡터를 추출해 데이터로 사용한다. 100 개의 토픽에 대해 각각의 파일을 분석하여 얻은 토픽 벡터를 이용하여 클러스터링을 수행하고 그림 3 과 같은 트리 형태의 계층적 클러스터링 결과를 얻었다.

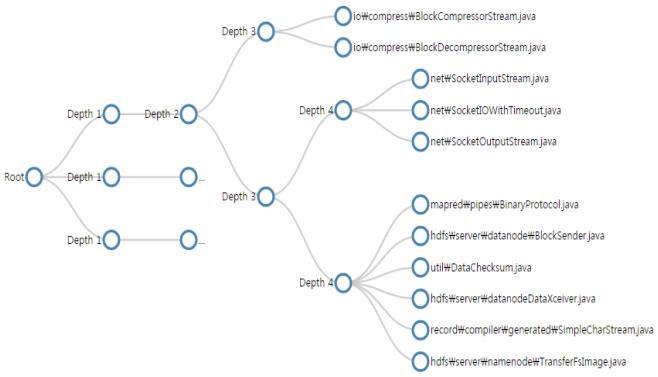


그림 3. Hadoop 0.19.0 의 계층적 클러스터링 결과(부분)

실험에 사용된 hadoop 0.19.0 의 토픽 벡터 매트릭스는 604 개의 노드로 이루어져 있다. 표 1 은 최대 4 단계까지 클러스터링을 수행한 결과의 클러스터 리스트의 일부이다.

표 1. 계층적 클러스터링 결과(부분)

인덱스	레벨	노드 수	중심점 수	상위그룹
0	0	604	17	Root
1	1	20	3	0
2	2	6	2	1
3	3	4	0	2
4	3	2	0	2
5	2	9	2	1
6	3	7	2	5
7	4	4	0	6
8	4	3	0	6

토픽 벡터 전체가 속한 초기 노드로부터 클러스터링이 진행되면 하위 노드에서 분할된 노드가 각각 다시 클러스터링 되는 것을 볼 수 있다. 클러스터 단계 N 을 높게 설정하면 같은 패키지 수준에 있는 파일들이라도 그 수가 분할 되기에 충분한 경우 연관된 기능을 가진 파일들이 하위 클러스터로 나뉘게 되며 예시가 그림 4 에 나타나 있다. N=4 로 설정하고 클러스터링을 수행한 결과 io 패키지에 속한 파일들이 단계 4 에서 분할 된 것을 볼 수 있다. 이와 반대로 클러스터 단계를 적정량보다 줄이면 두 개 이상의 패키지가 같은 클러스터로 배속되는 경우도 발생한다.

N 을 무한대로 설정하는 경우 클러스터 그룹에서 여러 패키지가 섞여있는 것을 하위 클러스터에서 그룹화 할 수 있지만, 계층 레벨이 많아져서 전체적인 구조를 파악하기는 점점 어려워진다. 실험에서는 레벨 3 이상에서 노드 수가 큰 차이가 없는 것으로 볼 때 N 을 3 보다 작거나 같은 수준으로 설정하는 것이 가장 적절하다고 판단할 수 있다.

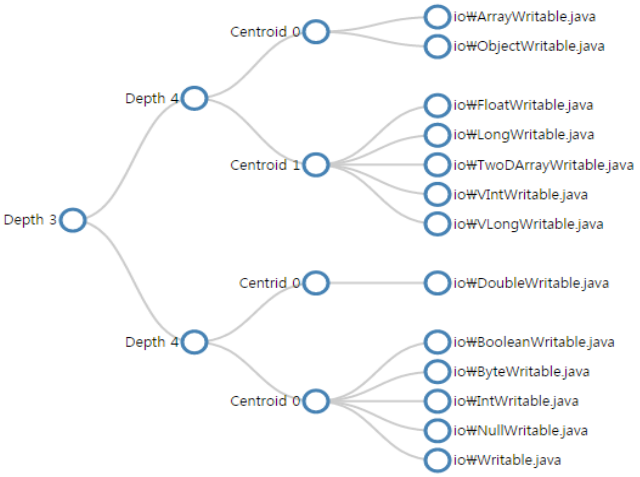


그림 4. 레벨이 높게 설정된 계층적 클러스터링

일부 노드는 클러스터에 혼자 배속되기도 하는데 소프트웨어에서 광범위하게 사용되는 노드가 클러스터링에서 원래의 패키지 그룹에 속하는 것이 아니라 다른 곳에 배속된 경우로 볼 수 있다. 클러스터링 결과에서 전혀 다른 패키지의 파일이 같은 클러스터에 소속되어 있거나 클러스터 내 일부 노드가 다른 것들과 관련 없는 것처럼 보이는 경우가 있는데 이것은 해당 파일이 원래 속해있는 패키지와 연관성이 크지 않거나 토픽 벡터에서 토픽 유사도가 높은 다른 파일과의 연관성이 더 크기 때문인 것으로 해석할 수 있으며 그림 5 는 실제로 다른 패키지에 있지만 비슷한 기능을 하는 노드가 함께 클러스터링 된 부분의 그래프이다. BlockLocation.java 와 LocatedBlock.java 는 서로 다른 패키지에 속해있지만 토픽

벡터가 유사한 값을 갖고 있어 같은 클러스터로 묶이게 되었다.

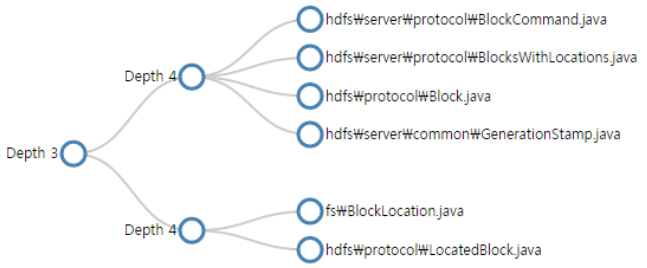


그림 5. 유사한 기능으로 묶인 클러스터

3.2 평가

K-means 알고리즘의 계층적 적용은 빠른 결과수렴과 비교적 간단한 구현방식 등 K-means 알고리즘의 장점을 그대로 갖지만 다음의 몇 가지 단점 또한 반영된다.

K-means 알고리즘의 결과는 클러스터 개수인 K에 민감하다. 데이터의 실제 클러스터 개수와 설정된 K 가 다른 경우 클러스터 하나에 두 개 이상의 그룹이 묶이거나 한 그룹이 여러 개의 클러스터로 나누어지는 일이 발생하기도 한다. 이 실험에서는 K 를 클러스터링 대상 노드의 개수에 따라 선정하게 되는데 토픽 벡터나 파일 구조 등 실제 데이터의 상태를 반영하지 않고 있기 때문에 적절한 K를 선택하지 못하는 경우가 많다. 그림 6 은 표준 K-means 알고리즘에서 클러스터 개수가 실제 데이터의 클러스터 수와 일치하지 않는 경우를 예시로 보여주며 그림과 같이 클러스터링에서 사용된 클러스터 개수가 실제와 다른 경우 실제 클러스터들이 한 곳에 몰리거나 빈 경우가 나타나기도 한다.

클러스터 중심점 초기화 기법으로 Forgy 알고리즘을 사용하는데 무작위 선출 방식이기 때문에 같은 데이터라도 서로 다른 결과가 나타날 수 있다. 실험에 사용된 데이터는 토픽의 수가 많고 노드의 개수가 비교적 적은 편이기 때문에 초기 중심점의 위치에 따라서 배속되는 클러스터가 달라질 수 있다. 토픽 수치가 높은 노드는 중심점이 다르더라도 결과에서 같은 클러스터로 수렴하게 되는 경우가 많지만 유사도가 충분히 높지 않거나 여러 토픽에 대해 유사도가 높은 노드는 매번 다른 클러스터에 수렴하게 되는 문제가 있다. 토픽 유사도가 충분히 높더라도 초기 중심점의 위치에 따라 결과가 지역 최적값에 수렴할 수도 있다. 이는 K-means 알고리즘에서 발생하던 문제이며 데이터가 한쪽 클러스터에 몰리게 되고 다른 쪽은 소수의 데이터가 의미 없이 묶여있는 경우를 볼 수 있다.

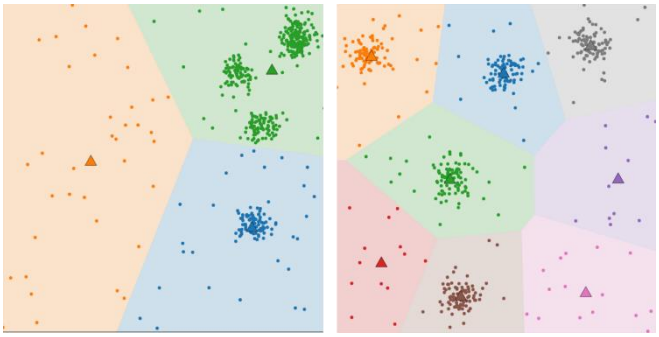


그림 6. 클러스터 개수가 실제 데이터와 다른 경우의 예시

Junit-4.10 과 apache-ant-1.8.4 에서 토픽 벡터를 추출하여 실험한 결과에서도 Hadoop 0.19.0 과 비슷한 결과를 얻을 수 있었다. 그림 7 은 Junit-4.10 에서 토픽 벡터를 추출해 클러스터링을 적용한 결과의 일부를

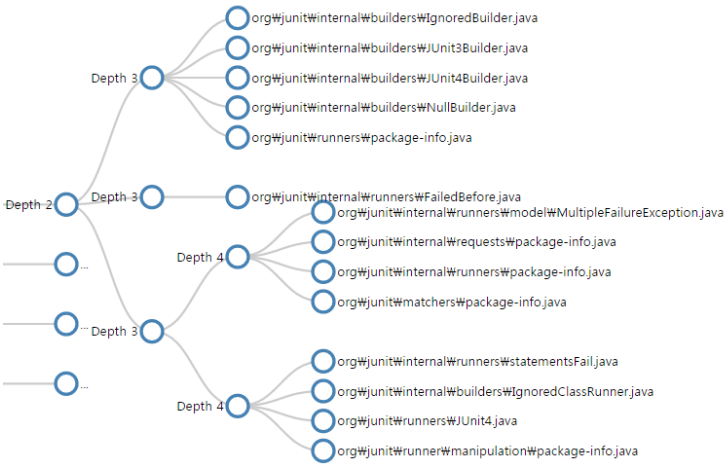


그림 7. Junit-4.10 의 계층적 클러스터링 결과(일부)

트리 형태로 나타낸 것이며 builder 관련 파일들이 잘 클러스터링 된 반면 여러 종류의 파일이 한 클러스터에 묶이는 경우가 발생하기도 한다. 클러스터링 수준은 Hadoop 0.19.0 의 결과와 비슷하게 나타나며 apache-ant-1.8.4 에서도 비슷한 수준의 결과를 확인할 수 있었다.

3.3 향후 연구계획

클러스터링 결과가 3.2 절에서 언급한 문제로 인해 노드가 엉뚱한 곳에 배속되거나 계층 구조를 제대로 반영하지 못하는 경우가 발생한다. 이를 해결하기 위한 방법으로 계층적 적용의 각 단계에서 클러스터의 개수 K를 선정하는 방법에 대한 알려진 기존의 방법론을 적용해 볼 수 있다. 클러스터링 기법에서 적절한 결과를 찾기 위해 클러스터 개수 K 를 늘려가며 결과를 모니터링하고 가장 좋은 결과를 도출하는 K 를 선택할 수 있다. 이 방법은 기존의 K-means 알고리즘에서 쉽게 적용이 가능하지만 계층적 적용에서

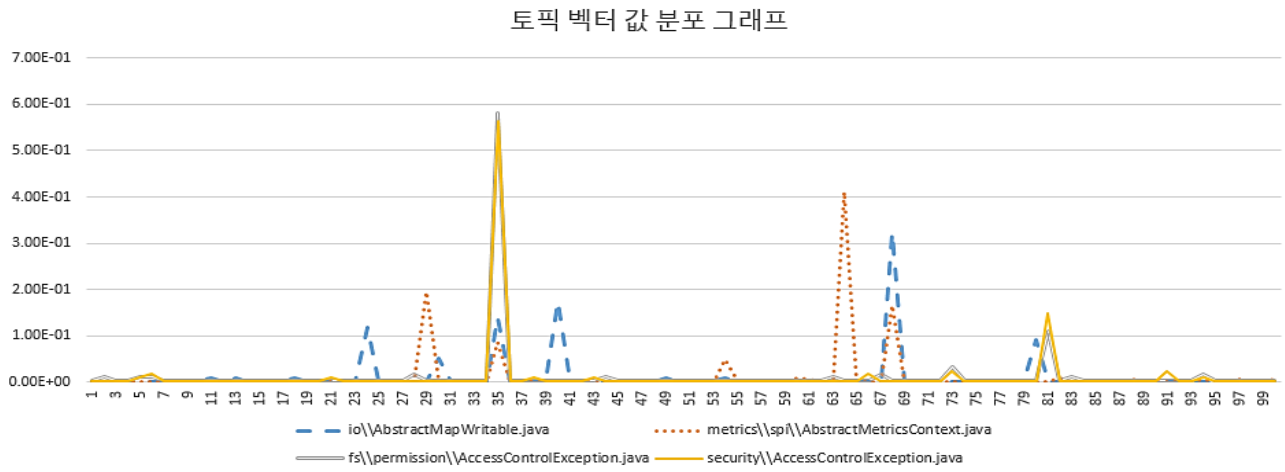


그림 8. 토픽 벡터 값 분포 그래프

는 선정해야 하는 K 의 개수가 많고 모든 단계를 모니터링 하는 것은 어렵기 때문에 클러스터링 결과가 적절한지 판단하는 기준을 세워 이 방법을 이용할 수 있다.

클러스터 중심점을 초기화 하는 기법에서 Forgy 는 매번 다른 중심점을 선택하기 때문에 결과에 오차를 발생시킬 수 있다. 본 연구에서 사용한 토픽 벡터는 노드의 토픽 유사도 정보를 가지고 있으며 특정 토픽의 유사도가 높을수록 클러스터의 중심에 가깝게 배치될 수 있다. 그림 8 은 토픽 벡터의 일부를 그래프로 나타낸 것인데 벡터가 몇 개의 특정 토픽에서만 높은 수치를 나타내고 있는 것을 볼 수 있다. 이처럼 클러스터링에 사용되는 데이터의 특성을 이용해 클러스터를 초기화 하는 방법을 사용한다면 결과를 보다 안정적으로 도출해 낼 수 있을 것이다.

본 논문에서의 향후 연구 방향으로는 위에서 언급한 개선 방안을 적용해 계층적 K-means 알고리즘의 정확도와 안정성을 개선하는 것을 목표로 하고 있으며 계층적 구조를 복원하는 기존의 클러스터링 기법과 비교하여 정확도를 향상시키고 알고리즘의 특성을 파악할 계획이다.

4. 결 론

본 연구에서는 소프트웨어의 계층 구조 파악을 위해 K-means 클러스터링 알고리즘을 계층적으로 적용하고 결과를 분석했다. 토픽 벡터에서 높은 연관성을 보이는 경우 클러스터링 결과에서도 비슷한 양상을 보이며 파일 수준에서 작성된 토픽 벡터를 이용했기 때문에 결과는 소프트웨어의 패키지 구조를 반영하는 계층 구조를 형성하는 것을 볼 수 있다. 클러스터링 결과에서 노드가 연관성 없는 곳에 배정되는 문제는 평가에서 지적한 대로 K-means 알고리즘을 수행하는 데 필요한 클러스터의 개수와 초기 클러스터 설정을 보정하여 정확도를 높일 수 있을 것이다. 이와 같은 문제를 개선하여 클러스터링 결과의 신뢰도가 충분한 수준이 되면 K-means 알고리즘의 계층적 적용 방법이 소프트웨어의 계층 구조를 표현하는 새로운 방안이 될 수 있다.

참고문헌

[1] J. Garcia, “A Comparative Analysis of Software Architecture Recovery Techniques”, Proc. of Automated Software Engineering(ASE), 2013

[2] J. A. Hartigan, M. A. Wong, “Algorithm AS 136: A K-Means Clustering Algorithm”, Journal of the Royal Statistical Society, Series C, pp. 100-108, JSTOR 2346830, 1979

[3] David M. Blei, Andrew Y. Ng, Michael I. Jordan, “Latent dirichlet allocation”, the Journal of machine Learning research, pp. 993-1022

[4] Ki-Seong Lee, Chan-Gun Lee, “Semantic Quality of Source Code Artifacts and its Application to Software Architecture Recovery”, Journal of Systems and Software, 2015

[5] M. R. AnderBerg, Cluster Analysis for Applications, Academic Press, 1973

공중전투기동 내장형 훈련 S/W 아키텍처 설계

장영찬, 지철규, 오지현, 김천영, 홍영석
국방과학연구소

목 차

서 론 (1/3)

◆ 공중전투기동훈련장비 (ACMI, Air Combat Maneuvering Instrumentation)

- 기동하는 전투기의 속도, 방향, 자세 등의 비행자료와 가상 무장발사/피해효과를 지상의 시현장비에서 실시간으로 시현하는 공중전투기동훈련 및 지상폭격훈련장비

◆ ACMI의 구성

1. AIS Pod(Airborne Instrumentation Subsystem) : 항공기 장착용 Pod
2. GRS(Ground Relay Station) : 지상 중계소
3. CCR(Central Control Room) : 중앙 통제실
4. DDS(Display and Debriefing Station) : 시현 강평실

AIS Pod

서 론 (2/3)

◆ ACMI 체계 운용개념도

서 론 (3/3)

◆ 현재 ACMI의 문제점

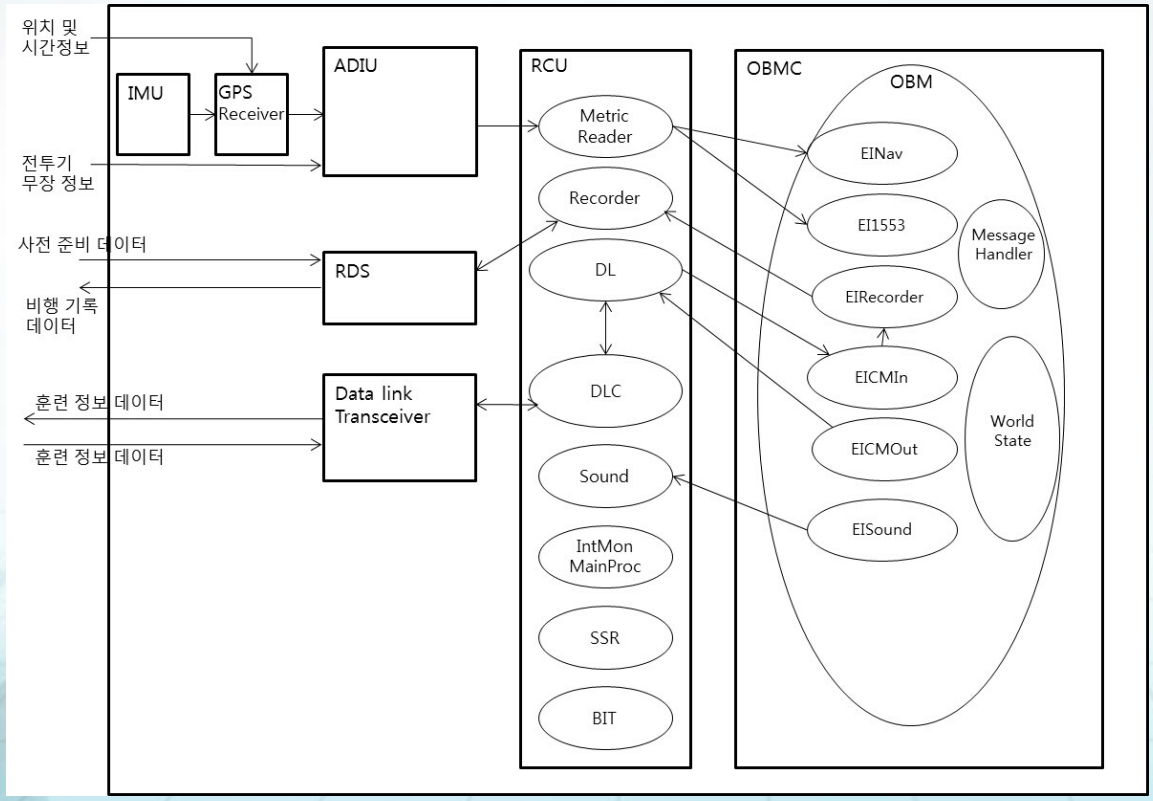
- 항공기 외부 장착 Pod
  - 유지보수 비용 증가
  - 항공전자장비의 중복
- 다양한 전술 훈련의 어려움
  - 참여 항공기 대수의 제한
  - 제한적 공대지 임무 수행
- 실기동급 단독 운용
  - 유류비, 유지보수비 등 예산 문제
  - 기상 악화로 인한 훈련 제한
  - 지형 및 환경적 제한

LVC 연동
전투기동훈련장비 및
내장형 훈련 S/W 필요성
제기

공중전투기동 훈련 S/W 분석(1/6)

◆ 국내 AIS-Pod 시스템 아키텍처

- AIS Pod는 ACMI의 핵심장비로서, 시공간위치정보(TSPI) 및 가상 무장발사 및 피해효과를 생성, 처리하여 다른 항공기의 AIS Pod 및 지상의 GRS에 데이터링크를 통하여 데이터를 전송함

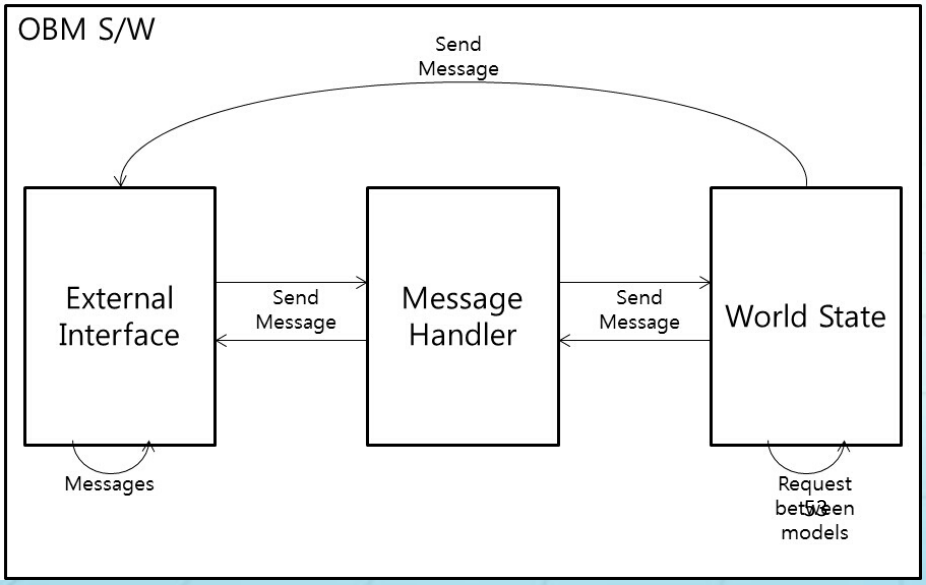


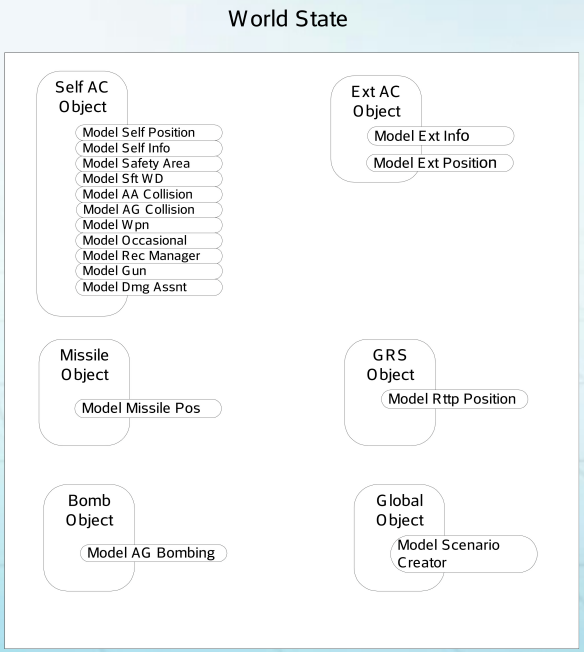
AIS Pod 시스템 아키텍처

공중전투기동 훈련 S/W 분석(2/6)

◆ OBM(Object Manager) S/W 분석

- OBM은 Pod의 심장 역할
- OBM의 구성
  - External Interface : OBM과 다른 component를 연결하는 외부 인터페이스
  - World State :
    - 다양한 객체(Object)들로 구성됨
    - 각각의 객체는 다양한 모델(model)들로 구성하며, 이 모델들이 종합적으로 기능 하여 **객체의 행위나 기능 모의**
  - Message Handler : External Interface들과 객체의 모델로부터 메시지 (데이터)를 수집하고 처리함

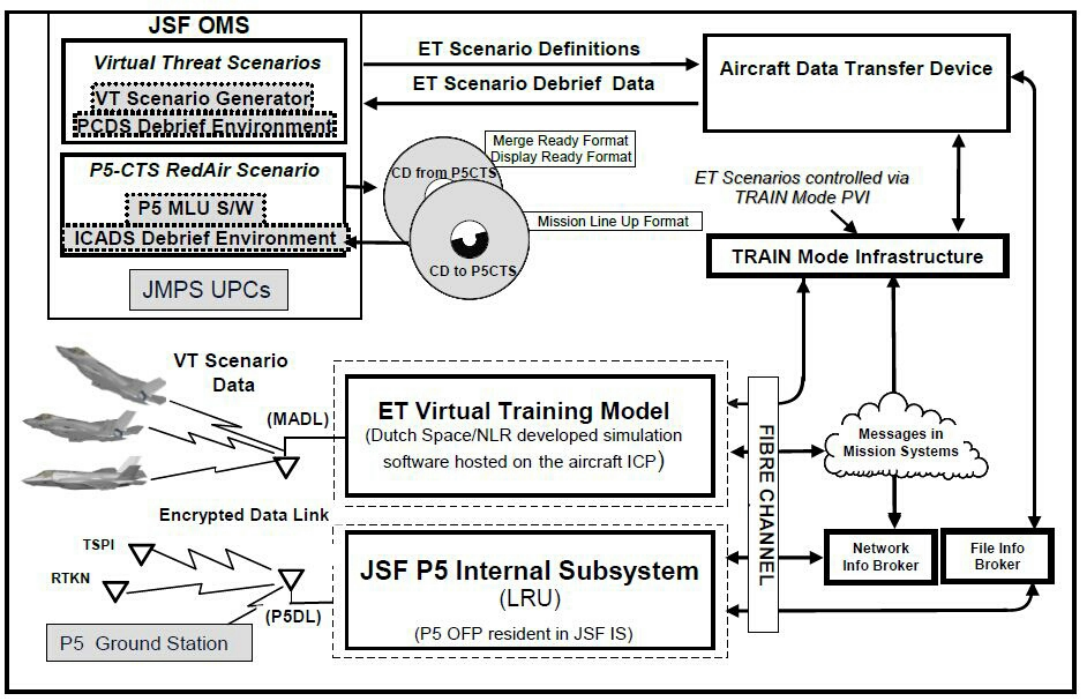




공중전투기동 훈련 S/W 분석(3/6)

◆ 국외 공중전투기동 훈련 S/W 분석

- F-35 Embedded Training System
  - 네덜란드 NLR에서 개발한 E-CATS(Embedded Combat Aircraft Training System) S/W 기반의 가상 전투(Virtual Training) 기능과 Cubic사가 개발한 P5-CTS(Combat Training System)이라 일컫는 ACMI 기능을 통합

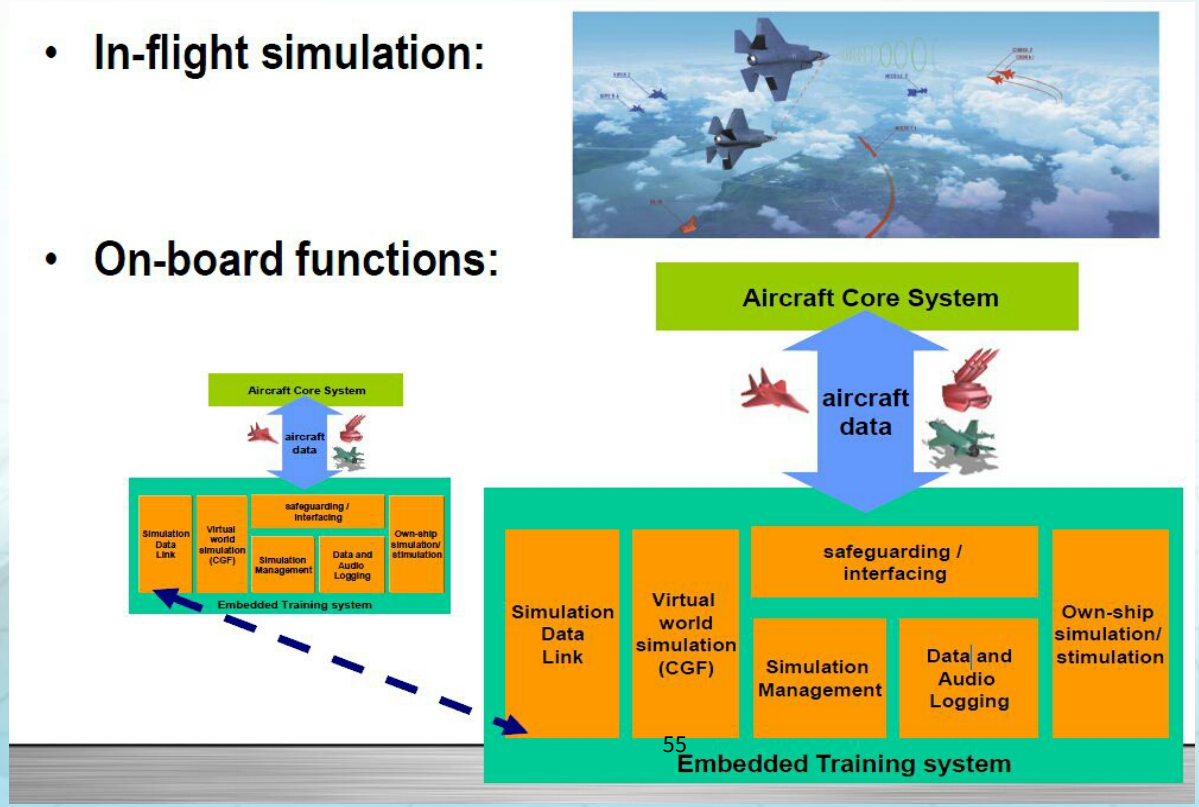


공중전투기동 훈련 S/W 분석(4/6)

◆ 국외 공중전투기동 훈련 S/W 분석

- F-35 Embedded Training System (계속)
  - E-CATS : VT는 최대 4대의 가상 전투기와 10대의 가상 지상위협을 CGF(Computer Generated Force)로 제공함

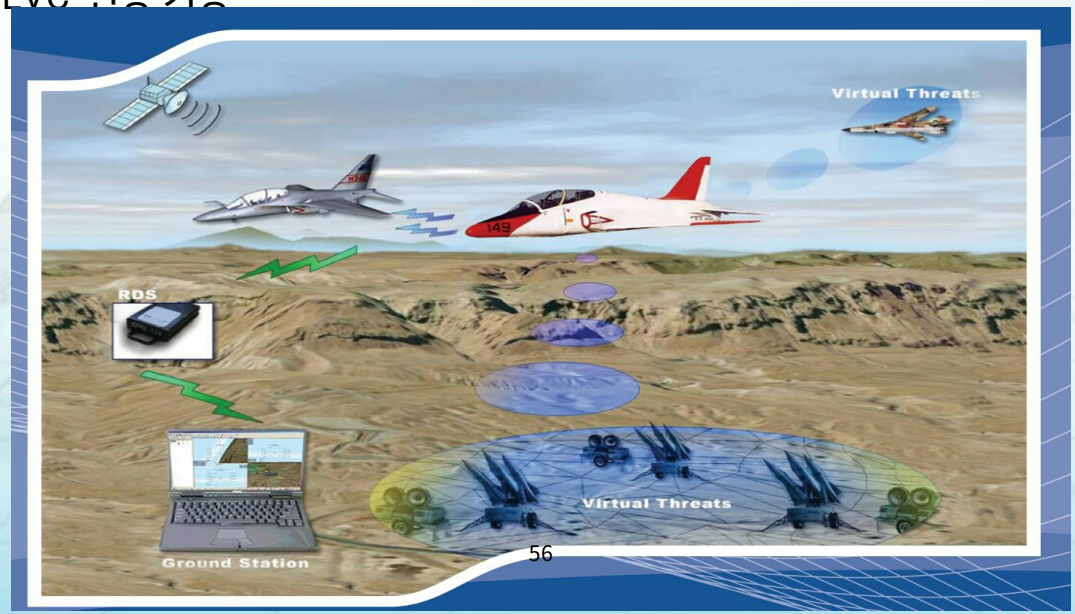
    -> On-board 기반 LVC 연동 훈련



공중전투기동 훈련 S/W 분석(5/6)

◆ 국외 공중전투기동 훈련 S/W 분석

- 이스라엘 EVA(Embedded Virtual Avionics)
  - 이스라엘 Elbit 사는 내장형 훈련 시스템인 EVA(Embedded Virtual Avionics)를 개발하였으며, 훈련기에 설치 시 5세대 전투기로 모의가 가능함
  - 주요 기능
    - 가상 항전장비, 무장, 센서, 합성 전장환경 제공
    - 내장형 ACMI와 디브리핑 기능을 포함
    - 다수의 전투기가 참여하는 분산훈련에서의 데이터링크 기능
    - LVC 연동 기능



공중전투기동 훈련 S/W 분석(6/6)

◆ 현 운용 공중전투기동 훈련 S/W 문제점 및 개선점 도출

- 국내 공중전투기동 훈련 S/W는 LVC 연동 기능 없음
- LVC 객체 정보 시현 등 항공기와 Pod 간 인터페이스 필요
- RWR, Chaff/Flare등 전자전 모의 기능 필요

공중전투기동 내장형 훈련 S/W 아키텍처 설계(1/5)

◆ 주요 요구사항

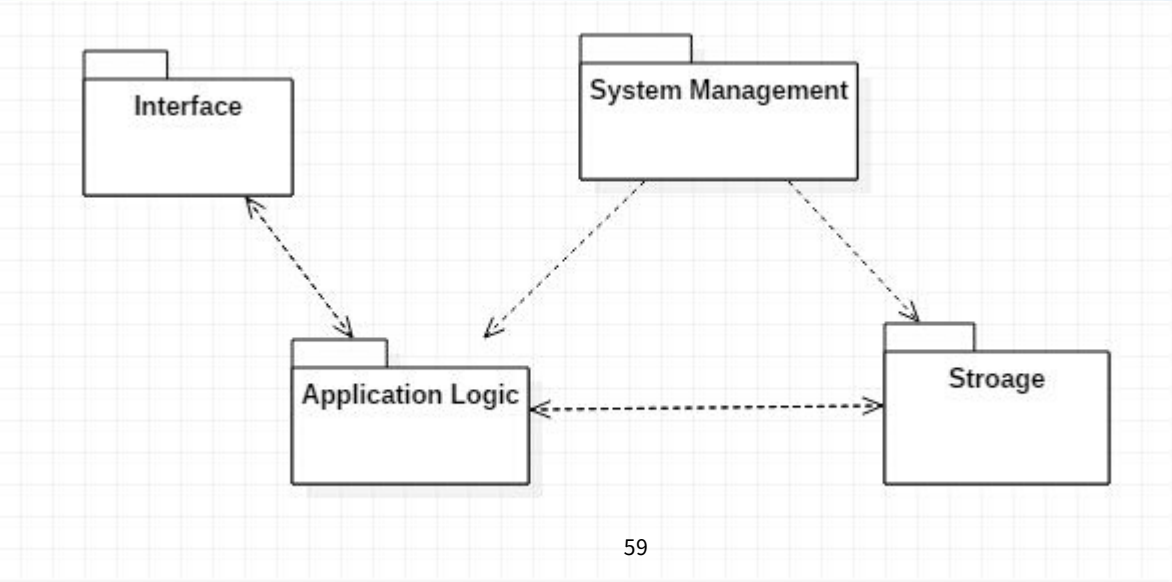
- 항공기 외부 장착 Pod
  - 항공기 OFP 수정의 제한으로 인한 외부 장착 Pod 기반 내장형 훈련 S/W 개발
- 무장 모의
  - 공대공/공대지 무장모의, 피해효과 등 모의
- LVC 객체 연동모의
  - 데이터링크 기반 실기동급/가상급/구성급 객체 연동모의
- 레이더/전자전 모의
  - 공대공/공대지 레이더, RWR, Chaff/Flare 모의
- 훈련 통제 및 제어
  - 훈련 시작/종료 제어, 훈련 임무자료 로드, 훈련 데이터 저장, 위협상황 경고
- 시스템 관리
  - BIT, 시스템 초기화, 시스템 상태 관리, 실시간 프레임 관리
- 인터페이스
  - 항공기 무장 발사 Discrete 신호, MIL-STD-1553B, 항법 데이터, 데이터링크, 이더넷 통신(항공기/Pod 간 인터페이스)
- 데이터 저장소
  - Removable Data Storage

공중전투기동 내장형 훈련 S/W 아키텍처 설계(2/5)

◆ 패키지 다이어그램 작성

- 패키지 분할
  - 인터페이스
  - 시스템 관리
  - Application Logic : 무장모의, LVC 객체 연동모의, 레이더/전자전 모의, 훈련 통제 및 제어
  - 데이터 저장소

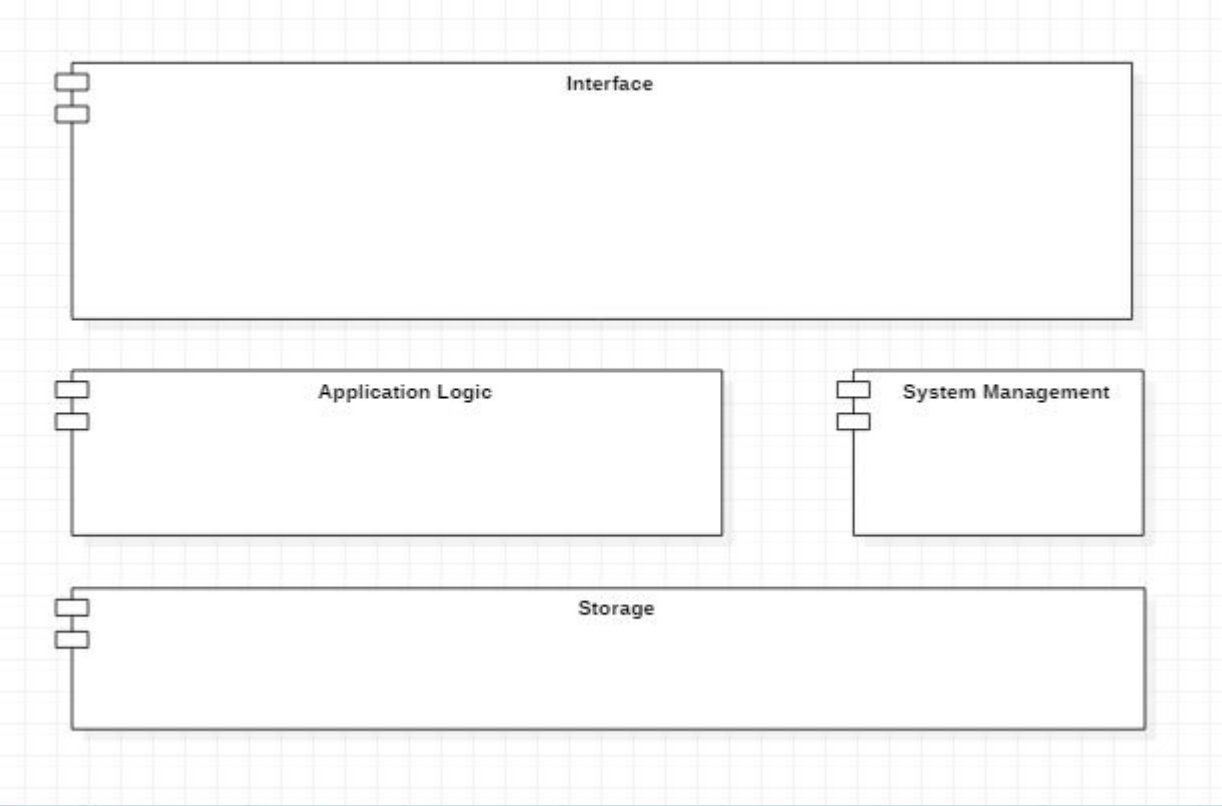
- 의존관계 설정



공중전투기동 내장형 훈련 S/W 아키텍처 설계(3/5)

◆ 소프트웨어 아키텍처 패턴 적용

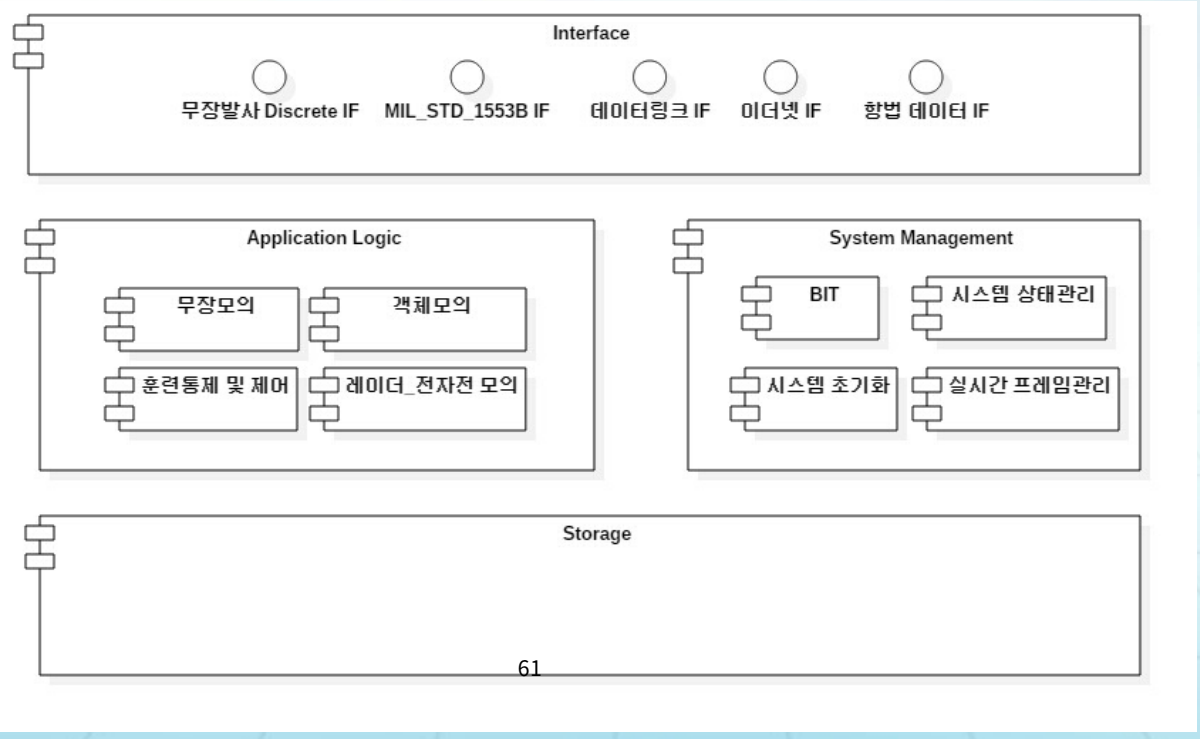
- 계층 패턴(Layered Pattern) 적용
  - Component Level 0 수준 구성 : 인터페이스, 시스템 관리, Application Logic, 데이터 저장소



공중전투기동 내장형 훈련 S/W 아키텍처 설계(4/5)

◆ 소프트웨어 아키텍처 패턴 적용

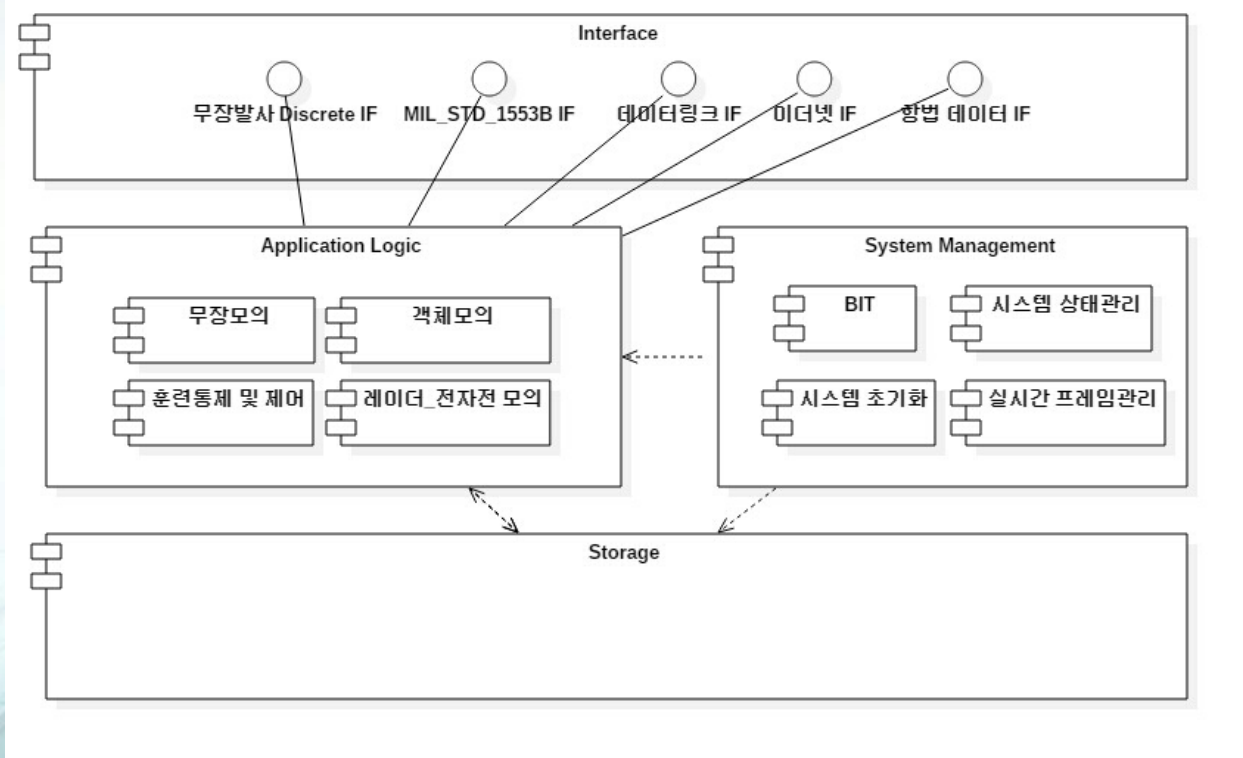
- 계층 패턴(Layered Pattern) 적용
  - Component Level 1 수준 구성
    - 인터페이스 : 무장 발사 Discrete 인터페이스, MIL-STD-1553B 인터페이스, 항법 데이터, 데이터링크 인터페이스, 이더넷 인터페이스
    - 시스템 관리 : BIT, 시스템 초기화, 시스템 상태 관리, 실시간 프레임 관리
    - Application Logic : 무장모의, LVC 연동 객체모의, 레이더/전자전 모의, 훈련 통제 및 제어



공중전투기동 내장형 훈련 S/W 아키텍처 설계(5/5)

◆ 소프트웨어 아키텍처 패턴 적용

- 계층 패턴(Layered Pattern) 적용
  - Component 관계 정의



결 론

- 공중전투기동훈련 S/W의 분석 내용을 기술하였고,
  현재 개발 중인 LVC 연동 공중전투기동 내장형 훈련 S/W의 아키텍처 설계내용을 기술하였음

- 향후에는 본 아키텍처를 보완 및 발전하여 기본설계 및 상세설계가 진행될 예정임

자동차 원격 제어의 신뢰성을 높이기 위한 시스템 구조 및 동작 절차 제안

이윤성[1], 권영채[2], 이성주[3], 이율희[3], 이석원[1]

아주대학교 소프트웨어융합학과[1], 아주대학교 미디어학과[2] , 아주대학교 사이버보안학과[3]
경기도 수원시 영통구 월드컵로 206

meetbaba@ajou.ac.kr, yyyoungk@ajou.ac.kr, tjdwn715@ajou.ac.kr , y1006@ajou.ac.kr,leesw@ajou.ac.kr

요약: 본 연구에서 제안하는 시스템 구조는 자동차 원격 제어 시 발생할 수 있는 보안문제를 해결하기 위한 논리적, 물리적 시스템 구조이다. 본 시스템 구조를 통해서 무선통신시에 메시지를 해킹 당하더라도 자동차에 접근하기 위한 주요정보와 사용자 정보가 노출되지 않게 하여 보안성을 높일 수 있다. 또한 모바일과 자동차 이외의 추가적인 인증 기기가 필요하지 않아 기존의 시스템에 비하면 사용성 또한 높일 수 있다.

1. 서론

1.1 연구배경

구글 자율주행자동차가 공개되고 난 뒤, 전세계적으로 자동차의 발전 방향은 자율주행 자동차, Connected Car 의 개념으로 암묵적으로 통일 되었다. 완전한 자율 주행 자동차, 완전한 Connected Car 로 발전하기 위한 중간단계로써 중요하게 여겨지는 부분이 바로 자동차 원격 제어이다. 원격으로 제어 가능한 기능의 범위를 확대하기 위한 신뢰성, 안전성을 확보하고 이를 위한 인프라를 갖추는 것이 완전한 자율주행차로 가는 길의 주요한 교량이 될 것으로 예상된다. 현재 각 제조사 마다 자동차 원격 제어 서비스를 제한적으로 시행하고 있다.[1][2]

그 중에 자동차 원격 제어 기능을 발달에 있어서 가장 큰 걸림돌이 되는 것은 바로 보안 문제이다. 기본적으로 무선 통신을 통해 이루어지는 기능에는 항상 해킹의 위험이 도사리고 있기 때문이다. 이를 해결하기 위해 현재 사용되고 있는 서비스는 본인 인증을 위한 복잡한 절차를 필수적으로 거쳐야 한다.

1.2 문제 정의

모바일을 이용한 자동차 원격 제어를 위해서는 사용자의 정보와 제어하는 자동차의 정보가 외부에 노출되지 않도록 하는 보안문제가 가장 큰 이슈이다. 이를 위해서 기존 시스템은 개인정보를 추가로 입력하거나, 추가적인 도구를 통해서 인증한 뒤에야 원격 제어 어플리케이션을 사용 할 수 있다. 이는 보안성을 일부 높일 수는 있지만 오히려 사용자의 사용성을 악화시키는 부작용이 있다.

자동차 원격제어를 위한 시스템을 단순히 보안 기술 관점을 넘어서 구조적으로 개선하기 위해 접근한다면 보안성을 높이는 동시에 사용성 또한 개선할 수 있을 것이다.

1.3 제안 방법

본 연구에서 제안하는 시스템 구조는 자동차원격 제어 시 발생할 수 있는 보안문제를 해결하기 위한 논리적, 물리적 시스템 구조이다. 시스템은 물리적으로 모바일, 서버, 중계기, Infotainment 시스템, ECU 5 가지로 구분된다. 이들 사이에는 교환되는 정보가 다른데, 모바일과 서버 사이에는 사용자의 ID, Vehicle Code, 모바일의 Mac 주소 3 가지 데이터가 공유된다. 이후 서버에서는 해당 Vehicle Code 에 해당하는 자동차의 IP 주소를 찾아 해당 주소로 인증요청, 혹은 원격제어 명령을 전달한다. 이때 중요한 점은, 사용자(end user)에게 자동차의 IP 정보가 전혀 노출되지 않는다는 점이다. 이는 메시지가 해킹을 당한다 하여도 어떤 차량의 정보인지를 원천적으로 불가능하게 함으로써 원격제어 시 메시지해킹으로 인해 발생할 수 있는 보안문제를 논리적, 물리적으로 원천 차단할 수 있는 구조이다. 본 시스템 구조를 사용하게 되었을 때, 기존 시스템에 존재하는 추가적인 인증절차가 필요 없어 이용자 스스로 차량 소유주임을 인증하고 해제할 수 있다는 점이 사용성 또한 높일 수 있는 장점이 된다.

2. 관련 제품 및 선행 연구

2.1 관련 제품

2.1.1 현대자동차 BlueLink

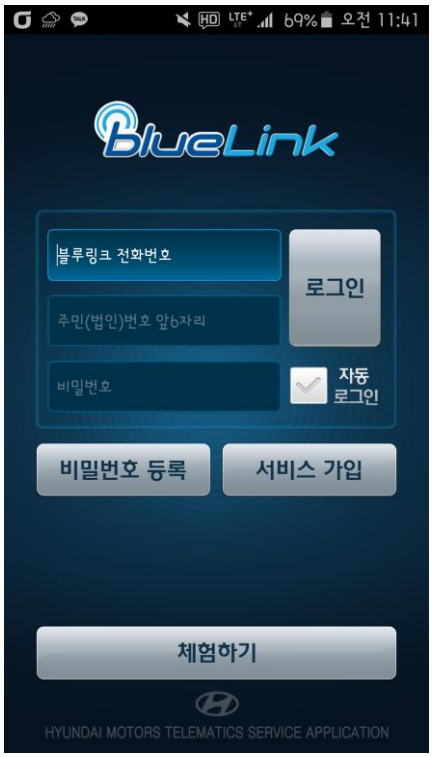


그림 1. 블루링크 어플리케이션 화면

현대 자동차에서는 원격 시동, 온도 조절 등의 원격 제어 기능과 더불어 텔레메틱스를 이용하는 서비스를 이용할 수 있는 도구를 블루링크(BlueLink)라고 한다. 원격 제어도 이 블루링크를 통하여 이루어지는데, 기본적으로 통신사 결합형 상품 기반이다. 따라서 자동차 원격 제어를 위해서는 사전에 자신의 휴대폰 번호를 통해서 본인 인증 및 사용자 인증을 따로 진행하여야 한다.[1]

또한 보는 바와 같이 서비스를 이용하기 위해서는 주민등록번호와 같은 개인 정보를 필수적으로 입력하여야 하기 때문에, 보안 이슈 외에도 개인정보 보호 문제가 발생할 소지가 있다.

2.1.2 BMW 의 Connected Drive

자동차 원격 제어 서비스의 해외 대표 모델은 BMW 의 Connected Drive 서비스라고 할 수 있다. Connected Drive 는 스마트 폰과 차량에 설치된 어플리케이션을 기반으로 동작하는데 이 또한 인증과정이 필요하다. 인증을 위해서는 BMW 웹 포탈에 회원가입을 통해서 차량과 통신할 스마트폰을 직접 등록하여야 한다. 이렇게 등록된 계정에 자동차의 고유번호(이하 Vehicle Code)를 계정에 추가하면 가입자 식별 정보가 들어있는 칩인 SIM 카드를 통해 보안코드가 차로 보내진다. 사용자가 이 번호를 확인하여 포탈에 입력하여 인증하면 인증 절차가 끝이 난다. 이 때부터 스마트폰에서 요청하는 정보는 BMW 서버를 거쳐서 각 앱에 전달된다. [4]

위 그림 2 를 보면 단순히 e-mail 과 비밀번호로 블루링크와 비교하면 개인정보 유출의 위험은 적다고 볼 수있다. 하지만 휴대폰과 차 이외에도 웹 포탈과 SIM 칩을 이용하여 차 인증이 이루어지기 때문에 그 과정에서 사용자의 노력이 추가로 필요하다는 단점은 여전히 존재한다.

그림 2. Connected Drive 어플리케이션 화면

2.2 선행 연구

자동차의 원격 제어의 보안 문제는 특별히 따로 연구되었다기 보다는 기존의 인터넷 보안 이슈와 동일하게 여겨지는 경향이 있다. 이는 1:1 무선 통신에서의 관점에서만 논리적으로 해결하려는 노력이지만, 자동차 원격 제어를 위한 전체 시스템의 구조와 데이터의 흐름에 따라서 보안 안전성을 강화할 수 있는 방법에 관한 연구는 이루어지지 않고 있다.

이와는 별개로 메시지 전달 포맷을 어떻게 하여 신뢰성을 높일 것인가의 주제의 연구 분야가 있다.[3] 하지만 이는 제품이 서비스화 되었을 때 다수의 유저가 존재하는 상황에서 제조사가 회원을 구별하고 관리하는 서비스 운영 차원의 시각이 결여 되어 있기 때문에 원격 제어를 위한 인프라까지 포함된 전체 시스템 관점에서 안전성을 높일 수 있는지에 대해서는 보장할 수 없다.

3 제안 기법

3.1 중계기(Intermediate Device)

자동차 원격 제어를 위해서는 차량의 시동이 걸려 있지 않은 상태에서도 무선신호를 수신할 장치가 필요하다. 또한 원격제어 명령이 올바른 사용자로부터의 올바른 명령인지 확인한 뒤 Infotainment 시스템으로 명령을 전달할 필요 또한 있다.

본 연구에서는 이 목적을 위하여 차내에 이러한 목적만을 위한 중계기가 탑재되어야 한다고 제안한다. 제조사의 지정된 서버로부터 push 방식으로 무선통신 메시지를 수신하도록 되어있어서, 메시지를 보낸 사용자가 인증된 사용자인지만 확인하여 메시지의 명령을 Infotainment 시스템에 전달하는 기능만 한다면 24 시간 내내 켜져 있더라도 차량 내 배터리 소모에 큰 무리가 없도록 할 수 있다.

이를 통해서 원격 제어를 위한 무선 통신 명령이 항상 자동차에서 수신 될 수 있게 하고, 또한 구조적으로 Infotainment 시스템에 전달되는 메시지가 한 번 필터링 되기 때문에, 차량 내 안전에 영향을 미칠 수 있는 Infotainment 의 보안을 한층 강화할 수 있다.

3.2 물리적 구조

본 연구에서 제안하는 시스템 구조는 그림 3 과 같이 모바일, 서버, 중계기, Infotainment 시스템, ECU 로 이루어진 구조로 사용자는 모바일 어플리케이션을 통해서 자동차 인증을 하고 원격제어 명령을 내린다.

서버는 각 제조사에서 관리하는 서버로서, 자동차의 고유번호(Vehicle Code)와 그에 해당하는 자동차의 IP(인터넷 주소)정보를 가지고 있다. 이미 인증된 사용자라면 그에 해당하는 사용자의 정보(User ID, Mac 주소)를 매칭하여 데이터베이스화 하고 있다. 중계기는 위에서 언급한 바와 같이, 서버로부터 오는 메시지를 수신하고 명령의 전달자 및 명령내용을 확인한 뒤 Infotainment 시스템에 메시지를 전달한다. 이 메시지의 내용에 따라 Infotainment 시스템의 어플리케이션을 동작시키거나 에어컨, 배터리와 같은 일부 센서 및 액츄에이터를 동작시키기 위해 ECU 를 컨트롤 한다.

모바일과 서버, 중계기 사이에는 무선통신이 사용되고 중계기과 Infotainment 시스템 그리고 ECU 사이에는 물리적 연결을 기반으로 한 하드웨어 신호 또는 CAN 통신 등의 방법이 사용된다.

3.3 논리적 구조 및 절차

3.3.1 1 단계 : 모바일→서버

User ID, User Mac, Vehicle Code 과 요청 내용을 전달한다. 이 정보 중 User ID 와 Vehicle Code 는 사용자에게 입력을 받는다. 사용자는 자신의 자동차의 IP 주소를 모르고도 자동차의 고유번호만으로 자동차를 원격제어하기 위한 명령을 전달할 수 있다. 추후, 중계기로부터 모바일로 전달하기 위한 메시지 또한 마찬가지 방법으로 IP 주소의 노출 없이 진행될 수 있다.

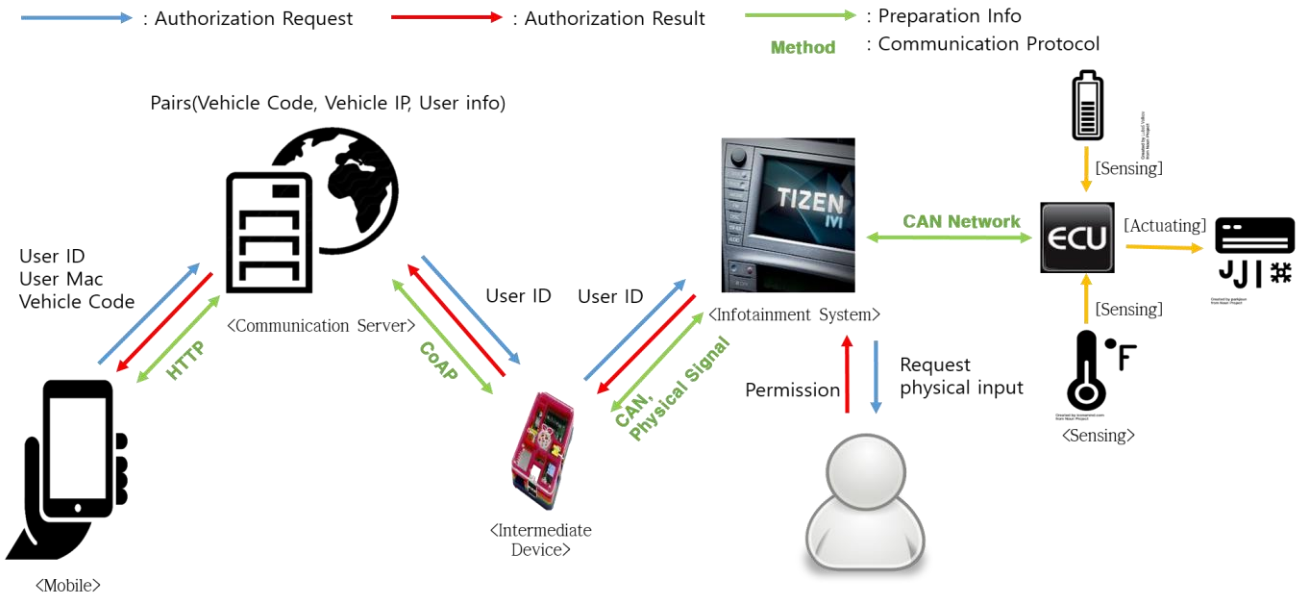


그림 3. 시스템 구조도

3.3.2 2 단계: 서버→중계기

서버에서는 Vehicle Code 를 통해 어떤 IP 주소로 메시지를 전달할지 이미 사전에 제조사에서 저장해 놓은 테이블을 통해서 확인할 수 있다. 만약 이미 인증된 사용자가 등록되어있는 Vehicle Code 라면, 1 단계에서 받은 User Mac 주소를 확인하여 서버 차원에서 명령을 차단하고 요청 거부메시지를 모바일 클라이언트에게 전달할 수 있다. 이는 구조적으로 보안을 한 단계 강화시킬 수 있다.

3.3.3 3 단계 : 중계기→Infotainment 시스템

Infotainment 시스템이 꺼져있다면, 하드웨어 신호를 통해서 Infotainment 시스템을 켤 수 있다. 인증과정에서는 모바일에서 요청된 사용자에 대한 정보(User ID)를 함께 보낸다. 인증 된 이후에는, 중계기에서 명령을 전달한 사용자와, 메시지를 필터링 한 후에 Infotainment 시스템에 전달함으로써 자동차의 편의 및 안전에 영향을 미칠 수 있는 Infotainment 시스템의 보안을 더욱 강화할 수 있다.

3.3.4 4 단계 : Infotainment 시스템 → ECU

이 단계에서는 Infotainment 시스템에서 사용하기 위한 정보를 ECU 에서 읽은 센서 값을 가져오거나, Infotainment 에 전달된 액츄에이터 동작 명령을 ECU 로 전달하게 된다. 이 과정에서는 자동차 전자제어 소프트웨어 표준인 AUTOSAR 의 DCM 을 이용하는 등의 방법을 통해서 소프트웨어 구조적으로 안전성을 한 단계 높일 수 있다.[4]

소프트웨어 설계 시, Infotainment 시스템과 ECU 와 교환하는 정보에 대해서 논리적으로 자동차 안전에 영향을 미치는 핵심 제어기에 대한 접근을 원천 차단한다면 차량의 원격제어 안전성과 신뢰도를 더욱 높일 수 있다.

3.3.5 인증 해지

자동차의 핵심 조작 권한은 소유주, 운전주에게 있다. 휴대폰과의 인증 절차 없이 자동차 내에서 수동으로 인증 해지가 가능하게 한다면, 자동차 구입 초기, 사용자 미숙지 또는 사용자의 인증된 모바일 기기 분실 등으로 인해 다른 사용자가 자동차와 이미 인증을 맺은 경우에 대한 우려도 해소할 수 있다.

4 구현 및 검증

본 연구에서 제안한 구조의 시스템이 실제로 구현 가능한 지, 구조적 신뢰성을 확보할 수 있는 지에 대해 검증하기 위해 프로토타입 수준의 시스템을 설계, 구현 하였다.

구현을 통해 본 시스템 구조의 구현 가능성을 검증하였고, 위에서 언급한 구조적 신뢰성을 확보 할 수 있었다. 이에 더해, 각 무선통신 마다 메시지 내용을 AES 등의 암호화 기법으로 암호화 하였을 때에도 구조에서 오는 문제 없이 보안 수준을 높일 수 있음을 확인하였다.[5][6]

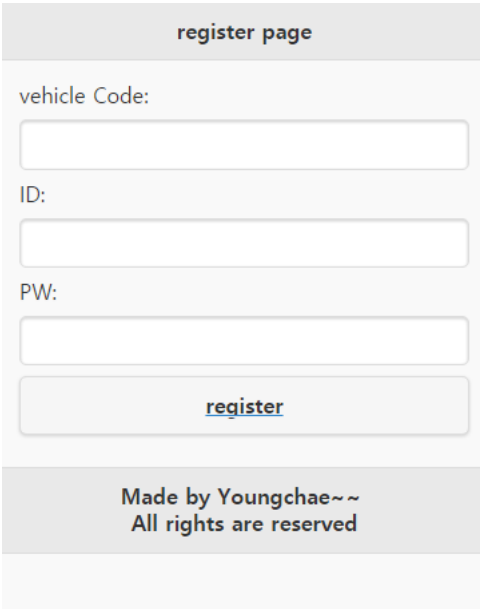


그림 4. 모바일 어플리케이션 구현 화면

5. 결 론

본 연구에서는 자동차 원격 제어 시스템의 보안성과 사용성을 높이기 위한 시스템 구조와 시스템 동작 절차를 제안하였다. 제안한 시스템 구조와 방법을 통해서 자동차 고유 번호와 짝을 이루는 자동차의 IP 를 외부에 노출시키지 않고 사용자가 자동차를 원격 제어하는 메시지의 전달이 가능하다. 또한, 기존 시스템이 사용하고 있던 추가적인 통신사 인증, 제조사 구매인증 등의 방법 없이 모바일과 자동차 기기만으로 사용자 인증 및 해지가 가능하다.

이를 통해 기존에 존재하는 시스템에 비교하여 구조적으로, 논리적으로 보안성을 높일 수 있고 모바일과 자동차 이외에 추가적인 인증수단을 없앰으로써 사용자의 사용성 또한 높일 수 있다.

향후 연구 과제로는 본 논문에서 제시한 소프트웨어 구조, 그리고 향상 가능한 신뢰성과 사용성에 대해서 측정 가능한 방법을 통하여 검증하고, 기존 시스템 및 다른 연구들과 비교 검증하는 연구가 필요

하다.

ACKNOWLEDGEMENT

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 실전적 SW 교육(SW 중심대학)지원사업의 연구결과로 수행되었음(R7115-15-1005)

참고문헌

[1] 현대 자동차 블루링크 서비스, http://bluelink.hyundai.com

[2] BMW, BMW_ConDrive_HowTo_Guide_Registrierung.pdf, 2013

[3] 최용운, 휴대 전화망 기반의 원격 차량 진단 및 제어 서비스에 관한 연구, 대구대학교 대학원, 2015.

[4] 이정환, 성기순, 오성민, 신재욱, LTE 기반 V2I 환경에서의 차량 진단 구조에 관한 연구, 한국 통신학회 학술대회 논문집, 580-581, 2015.

[5] 동영상 시연 링크, https://drive.google.com/open?id=0B3ty8WHnfFf3U0pEZHpEQ0ZMWjQ

[6] 설계 및 구현 문서 링크, https://drive.google.com/open?id=0B_D68Wa2f2HMOVRsX21oZl9HSnc

Using Problem Frames and Goal Modeling techniques to determine Variability in Smart Grid RTP systems

Meetushi

Ajou University, Computer Engineering Department
Suwon-si Gyeonggi-do, South Korea
meetushi@ajou.ac.kr

Seok-Won Lee

Ajou University, Department of Software Convergence
Suwon-si Gyeonggi-do, South Korea
leesw@ajou.ac.kr

ABSTRACT: Variability management is an important aspect that determines to what extent complexities can be handled in the system. This paper proposes to analyze variability in smart grid RTP system using i* goal modeling, problem frames, use case maps and live sequence charts. The integration helps to analyze goals of the system along with system contexts. The scenarios associated with the context are extended using use case maps and LSCs which assists in analyzing changes in contexts and the corresponding changes in requirements in detail. The goal of this approach is to make problem frames use case maps and LSCs even more alluring to software developers, who are generally comfortable with the ideas of scenario-based modeling.

KEYWORDS: Smart Grid, i* Goal model, Problem Frames, Use Case maps, Live Sequence Charts.

1. Introduction

Smart grid is an intelligent power system that integrates advanced control, communications, and sensing technologies into the power grid [8]. In smart grid, the electricity service providers rely on demand response programs to motivate customers to shift their loads from on-peak to off-peak periods [1]. Real-time pricing (RTP) is considered as a very direct and efficient approach for DR [1]. With RTP, the service provider broadcasts electricity prices on a rolling basis, i.e., the price for a given time duration (e.g., an hour) is decided and broadcasted before the commencement of the period (e.g., 15 minutes beforehand).

Residential buildings have been one of the large sectors in terms of consuming energy [19]. HVAC systems are the main target for energy management and load reduction because they constitute a significant part of the annual total energy consumption in the world [18, 19]. Programmable Thermostats (PTs) are widely used to automatically control HVAC devices to conserve energy and provide indoor thermal comfort [20]. PCTs help households to save on high electricity prices [22]. In fact, the user can preset a price threshold, then manually enter different offset values associated with different RTP rates (On-peak, Mid-peak, off-peak) [23].This feature potentially enables consumers to participate in DR events that will have a considerable impact on peak load reduction [21].

From the service provider's viewpoint, furnishing pricing updates at regular intervals will enable better load shaping and thus achieve better coordination of inconsistent supply and demand. From the consumer's viewpoint, RTP will provide new ways to reduce rates, provided that smart usage decisions are made.

But so far not much effort has been made to review the requirements and context changes of the RTP system and analyze them in detail to understand the loopholes and increase system robustness.

In this paper we propose a framework based on goal models and problem frames using use case map and live sequence charts to capture the behaviour of the RTP systems in various contexts. The main contribution of the work is to synergize the goal modeling, problem framework, use case maps and live sequence charts to assist in requirements elicitation process. We capture variability by using goal models and problem frames and model the variability using use case maps which are transformed into live sequence charts.

The remainder of the paper is organized as follows. Section 2 introduces the background of the framework. Section 3 describes the related work. Section 4 introduces the framework, Section 5 explains the location of framework in RE technology Map. Section 6 Introduces the case study of smart grids and section 7 explains in detail the application of the framework on RTP systems of Smart Grid. Finally, Section 8 concludes the paper.

2. Background

In this section we briefly introduce goal modeling, problem frames, use case maps and live sequence charts which form background of our framework.

2.1 Goal Modeling

Goal-oriented modeling techniques in requirements engineering provide a powerful mechanism for requirements projection. Goal models are mostly represented in tree-like

structures that define the intentions of different stakeholders at different levels of abstraction [7].Goal modeling also serves as a means of linking high-level strategic goals to low-level systems requirements [4].

Figure 1 shows simple i* Goal model. As seen in the figure the main goal is to organise a party at home which has sub goals invite friends and order food. If the friends are invited by email it helps in the achievement of softgoal faster response where inviting friends by letter hurts the soft goal. The goal order food depends on the number of friends invited.

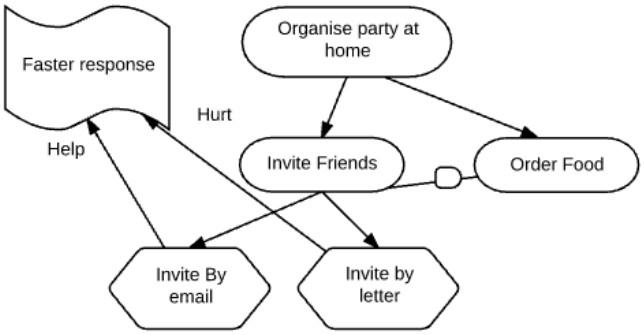


Figure. 1 A simple example of goal model structure

However, despite the application of goal-oriented modeling techniques to model business goals and strategy they have a number of shortcomings. First, they tend to be incompetent in describing problem context [5]. Second, the goal models grow quickly, acting as a threat to manageability and traceability [3]. Third, goals are hierarchical, hence, sometimes it becomes quite complex to determine where a goal is situated in the hierarchy and how it relates to the problem context to know what is needed.

2.2 Problem Frames

Problem frames [2] are one of the RE approaches, involved in the early life-cycle of software engineering. A problem frame consists of machine, problem domains, interfaces between them, and requirement. As shown in Figure 2 ATM is the machine where the domain card reader is constrained by the requirement validate the card.

Figure. 2 Problem Frame Diagram

Problem Frames serve as a powerful tool for describing and decomposing problem contexts. But they are weaker at associating requirements to one another when projecting from problem context towards the machine which is required for problem decomposition of complex systems. Also there is no way to connect the requirements to the goals or to identify new goals.

2.3 Use Case Maps

Use Case Maps are a scenario based software engineering technique for describing causal relationships between responsibilities of one or more use cases[24][25]. They provide the ability to model dynamic systems where scenarios and structures may change at run-time. UCMs also assist in the analysis and preparation of high level design.

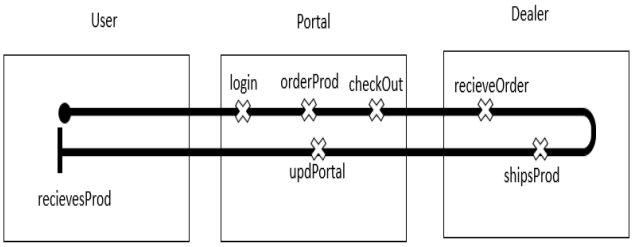


Figure. 3 Use Case Map

Figure 3 shows a use case map for a scenario where a user orders a product from a portal.

2.4 Live Sequence Charts

Live Sequence Charts were used to specify the liveness of the requirement. LSCs are used to represent the system requirements/behavior in an inter-object style i.e. inter object behavior describes the interaction between objects per scenario [27]. And intra object behavior describes the way a single object behaves under all circumstances which is mostly used by state charts.

Figure 4 shows a live sequence chart of the process of a person withdrawing money from bank.

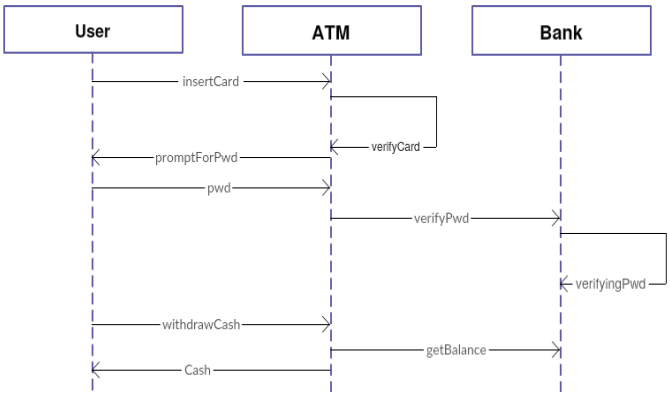


Figure. 4 Existential Live Sequence Chart

Problem frames have the potential to be combined with other RE techniques, such as goal-oriented approaches [1]. The complementary nature of problem frames and goal-oriented approaches suggest the combination of these techniques to capture complete requirement of stakeholders. Further by combining the problem frames with UCMs and LSCs it is possible to achieve greater confidence in modeling variability at an early stage.

3. Related Work

Jackson [15] provides numerous illustrative examples of the step-by-step transition from context diagrams to problem diagrams and from problem diagrams to sub-problem diagrams that involve the application of problem transformation tools.

The decomposition of complex problems into sub problems is also addressed by Bleistein et al. [9–13], where goal decomposition complements problem decomposition and is used to drive the decomposition process through Jackson's context diagrams. They apply the problem frames approach in the context of explicitly modeling and tracing from business strategy objectives, described through goal modeling [16,17], and business model context, described. Combined, they represent problem diagrams that capture the requirements problem from business strategy through to concrete system requirements. To add detail to the connections between goals and context, in [11, 12] they introduce process models in the form of Role Activity Diagrams, extending the work described in [12–14]. It should be stressed that, despite these efforts to ease the task of problem decomposition and matching basic problem frames, little is known of how the problem frames approach performs in the analysis of complex software problems.

A combination of use cases with problem frames is introduced in the work proposed by Del Bianco and Lavazza [28, 29]. The authors investigate the possibility of enhancing the problem frames with concepts derived from requirements modeling techniques like use cases based on scenarios and histories. This approach does not cover capturing and modeling the goals of the system. It is similar to the work proposed by Choppy and Reggio [30].

In an empirical study, Phalp and Cox [14] compare students' and practitioners' ability to match problem frames to a variety of realistic software problems. Transition from context diagrams to problem diagrams, and from problem diagrams to sub-problem diagrams that involve the application of problem transformation tools.

Liu and Jin use i* goal models and problem frames to enhance the information about the actors in an i* model [31]. The authors relate the domain constraints from problem diagrams to i* models. The tasks, which the machine would be assigned from the actor in a problem diagram are represented in i* using tasks. The authors neither consider the elicitation of the requirements nor the representation of goals or relating them to the context.

4. Framework for Context Analysis

We show how the problem frames approach can be augmented by coupling the contexts to problem diagrams. By using the problem diagram with contexts different variable states of the actor that contribute to the achievement of the goal can be identified. Use case maps and LSCs help to represent the dynamics of the problem diagram.

4.1 Functional Description

Step 0: Identify the environment: Identify the environment of the system and the environmental elements significant to the system i.e. its actors, dependencies their interactions.

Step 1: Set up the Goal model: i* goal model has been used to model social dependency relationships between domain actors, dependencies and interactions. i* shows each role (an actor, agent or position) as a large circle containing the goals, tasks, and resources which that role owns so it becomes very easy to identify actors and their tasks.
Step 2: Select an actor: After setting up the i* goal model, we select an actor and identify the high level goals of the domain actors and refine them into tasks and interaction relationships.

Step 3: Identify the problem contexts: For the selected actor we identify the various problem contexts which will contribute is setting up our problem diagram in the following step.

Step 4: Set up the problem diagram: By using the contexts identified for the actor in the previous step we set up the problem diagram. These problem diagram depict various variabilities that influence the actor with reference to the goal.

Step 5: Identify the flow of information in the context selected from problem diagram: Select the context of the actor and identify the information flow in the context.

Step 6: Set up the Use Case Map: For the selected context depict the entire scenario including the information flow. This enables reasoning about undesirable interactions of scenarios.

Step 7: Set up the LSCs: For the selected context depict the inter object scenario that specify possible mandatory, forbidden conditions between elements messages, location and conditions.

5. Location of Framework in RE Technology Map

Tsumaki and Tamai[32] focused on two dimensions in order to characterize the variety of techniques: one concerning the elicitation of operational types and other concerning the target object types. They divided the operational map into two categories i.e. static and dynamic and object type dimension into closed and open types.

Static: Requirements are elicited from the domain centralizing on its static structure. They are collected and sorted in a methodized and orderly way. There are rules to deconstruct/construct requirement related objects, including (sub) domains, goals and requirements. The methods that belong to this category in our approach are Problem Frames.

Dynamic: Requirements are elicited from the domain centralizing on their dynamic context. They are elicited immethodically or in an imaginative way. Therefore, this way of eliciting requirements is also regarded as “dynamic” activity, encouraging brainstorming and visualization. The methods that correspond to this category in our framework are Use case maps and LSCs.

The second dimension concerns with the properties of the target space to be evaluated, either comparatively closed or open.

1. Closed: The object space is comparatively stable, known and closed. It can be understood by concentrating mainly on forms.

2. Open: The object space is comparatively unstable, unknown, changing and open. To explore the space, meanings have to be much considered.

Figure 5 shows the allocation of techniques used in our framework in the RE technology map. The i* framework (Yu, 1997) overlaps with the goal oriented approach but extends to the right direction. Problem frames are closed and static. LSCs and use case maps are both closed and dynamic.

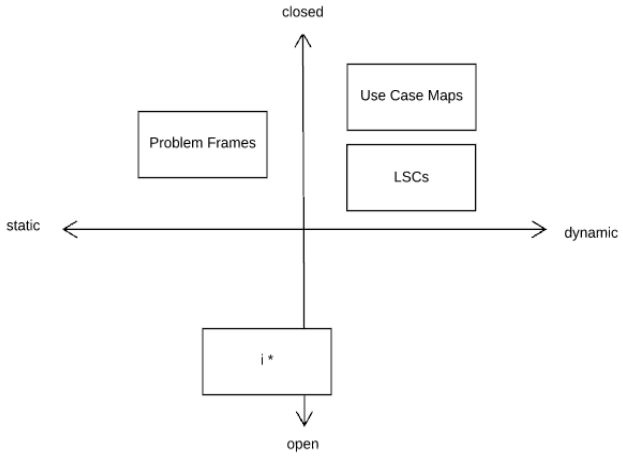


Figure. 5 RE Technology Map

6. Introduction to Case Study in Smart Grids”

Real time pricing of smart grid encourages customers to reduce their loads at peak time and also serves as an essential tool to develop competent demand side management strategies.

The RTP system runs an electricity market on a distribution feeder-by-feeder basis. A simplified drawing of one of the markets is depicted in Figure 6 .The main participants and their roles in the RTP system are explained below:

HVAC(heating, ventilation, and air conditioning) provides thermal comfort and adequate indoor air quality.

Wholesale Market: Refers to the electricity market where producers sell their electricity and it is mainly bought by electricity retailers. Price of the wholesale market varies strongly with time.

The Service Provider provides electricity services to customers. This domain acts as an intermediator among operations, markets and customers. It provides billing, customer management, and other emerging operations.

The dispatch system assembles the bids from all houses on the feeder along with the market price for supplying electricity and creates a cleared price.

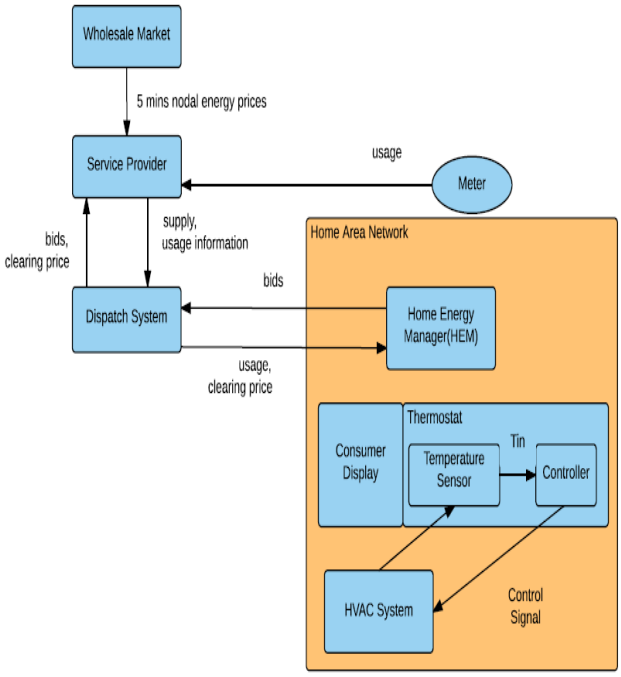


Figure. 6 RTP System Overview

The HEM makes electricity consumption smarter and more efficient and increased savings for consumers as well as utilities. It assembles all the bids at home and communicates to the dispatch system and takes the cleared price from the dispatch system.

Meter obtains the energy used during the 5-minute interval so the bill can be calculated and communicated to the billing system.

Consumer Display displays the estimated billing price for energy so the consumer can participate with other energy saving actions.

Thermostat is the main control unit for a heating or cooling system. It has a temperature sensor to detect the indoor temperature and output of sensor is sent to Controller which regulates the HVAC state.

HAN(Home Area Network) is a communication network within a home that allows transfer of information between thermostats, PC’s, home security devices, mobile phones etc.

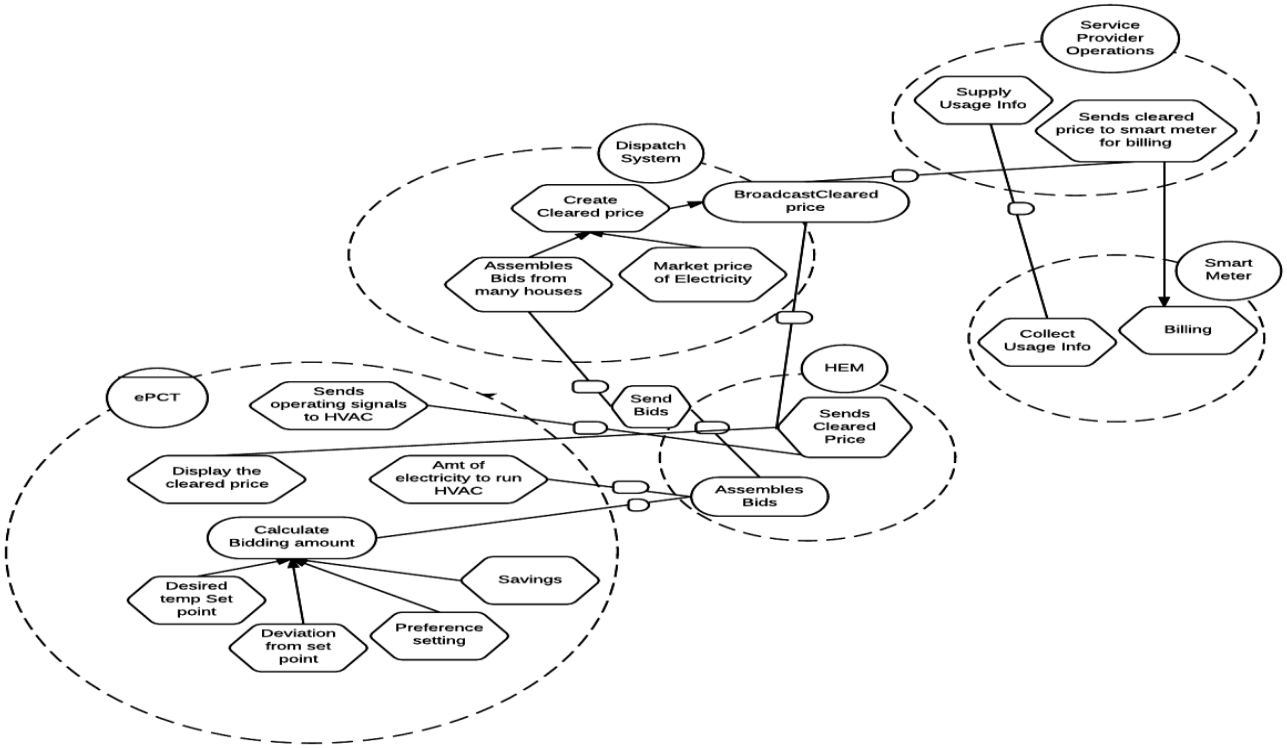


Figure. 7 i* Goal Model representation of RTP system

7. Application of the framework on RTP systems of Smart Grid

Step 0: Identify the environment: The first step involved identifying the overall goal of the system, its environment i.e. domain actors involved, their tasks, dependencies and relationships with other actor.

Step 1: Set up the Goal model: With the result of Step 0 i* goal model was set up as seen in Figure 7. The goal model helps us to understand the functioning of the RTP system and the roles of various actors involved in the system and how they coordinate to achieve the overall goal of the system. The domain actors include service provider operations, dispatch system, HEM and thermostat agent. The thermostat agent communicates with both the HVAC unit and home energy manager (HEM).The thermostat agent monitors the market price of electricity and converts the residents' desired temperature set point, the current deviation from that set point, and their preference setting for relative comfort and savings into an amount it is willing to bid for the following 5 minutes of electricity.

The HEM takes this price, along with the amount of electricity needed to run the HVAC unit, and assembles all bids in the home and transmits the bid information to the dispatch system.

The dispatch system collects the bids from all houses additionally with the market price for supplying electricity and creates a cleared price. The cleared price is transmitted to all homes' HEMs and forwarded to the service provider's operations system for billing. The billing system interchanges information with the advanced metering infrastructure smart meter the home to retrieve the energy used during the 5-minute interval so the bill can be determined. The HEM transmits the results of the auction to the ePCT which transmits the corresponding signal to the HVAC unit.

A consumer display is built into the ePCT; it displays the estimated billing price for energy so the consumer can check price of electricity, residents' desired temperature set point, the current deviation from the set point and resident's preference setting for relative comfort and savings

Step 2: Select an actor: After setting up the i* goal model of the RTP system, we select an actor for example smart thermostat agent whose overall goal is to address preference of comfort of the consumer along with savings.

Step 3: Identify the problem contexts: For the selected actor i.e. smart thermostat agent we identify the various variability contexts that influence its operation like market price of electricity, resident's desired temperature set point, the current deviation from that set point, and resident's preference setting for relative comfort and savings.

Step 4: Set up the problem diagram: By using the contexts identified in the previous step we set up the problem diagram as seen in Figure 8.

The problem diagram depicts a small part of problem context and offer a more objective and structured way to address the problem rather than focussing on the entire domain. The diagram explains that for the requirement set bid amount the machine thermostat agent is affected by market electricity price, resident's or consumer's set point, deviation from the set point and the preference setting by the consumer for his comfort. The notation a(EP! Electricity price) suggests that phenomena electricity price between the thermostat agent and market electricity price.

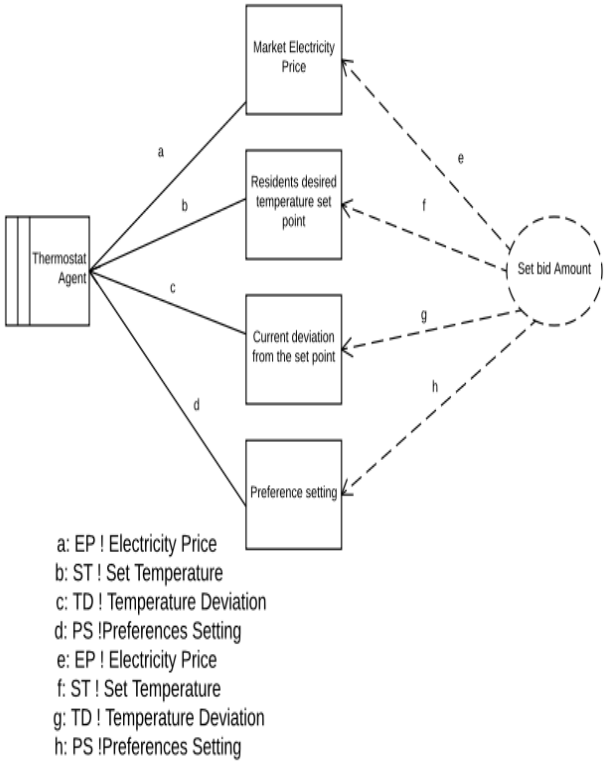


Figure. 8 Problem Diagram for Thermostat agent

Step 5: Identify the flow of information in the context selected from problem diagram: For the selected context for a thermostat agent determine how the information flow happens e.g. For day to day period of operation the end user sets his/her desired temperature ($T_{desired}$), and minimum and maximum temperatures (T_{min}, T_{max}), through a five-level setting for their preference for more comfort (tighter temperature control) or more savings (more flexible temperature control).Bids are submitted every 5 minutes. Bid price is calculated by each thermostat agent as shown below.

$$P_{bid} = P_{avg} + \frac{(T_{current} - T_{desired}) \times k \times P_{dev}}{T_{max} - T_{min}}$$

where
P_{avg} = the average price over the last 24 hours
P_{dev} = the standard deviation of the price over the last 24 hours
$T_{current}$ = the current indoor air temperature
$T_{desired}$ = the desired indoor air temperature
k = the responsiveness desired by the consumer
T_{max} = the maximum temperature limit
T_{min} = the minimum temperature limit.

Figure. 9 Bid price[33]

P_{bid} and q_{bid} are sent by the HEM to the dispatch system, where they are assembled with the other bids and the clearing price is decided. The cleared price (P_{clear}) is then published to all the HEMs on the feeder, which passes it on to the thermostat agents. In the cooling mode if clearing price(P_{clear}) is greater than the bid price(P_{bid}) results in set temperature(T_{set}) being higher than current temperature($T_{current}$)and less than T_{max} so the HVAC unit will not run.

Step 6: Set up the Use Case Map: For the determined flow of information path for a selected context in the previous step we depict the entire scenario using the Use case Map as shown in Figure 10. The Use Case map represents the interaction among the components HVAC, thermostat agent, HEM and dispatch system. It shows the simple scenario of setting the state of the HVAC using causal relationships between many use cases. The relationships are said to be causal as they relate concurrency and partial orderings of activities and also link causes (e.g., preconditions and triggering events) to effects.

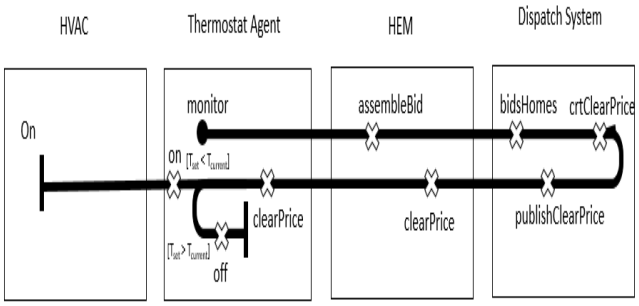


Figure. 10 UCM representing HVAC state scenario

UCMs also help in structuring the scenarios and also determine alternative flows. As seen in the figure 10 after receiving the cleared price from the HEM based upon the steps we have described before. The thermostat agent checks whether the set temperature is higher than the current temperature the HVAC will not run otherwise the HVAC will run.

Step 7: Set up the LSCs: For the selected context and scenario in the previous step depict the behaviour using the Live Sequence Chart as shown in Figure 11. LSCs are scenario based specifications which describe how the system components, environment and users collaborate to achieve system level functionality.

The vertical lines represent the components involved in the scenarios i.e. HVAC, Thermostat, HEM and Dispatch System. The interaction or communication pathways between the components are represented by arrows. The chart is read from top to bottom as the time is assumed to go from top to down. The LSCs setup helps to have a clear representation and better understanding of the flow of events.

As seen in Figure 11 the dispatch system receives bids from HEM and creates clear price which is broadcast to the HEM. The HEM broadcasts the cleared price to the thermostat which checks in cooling mode if set temperature (T_{set}) being higher than current temperature ($T_{current}$) and less than T_{max} so the HVAC unit will not run.

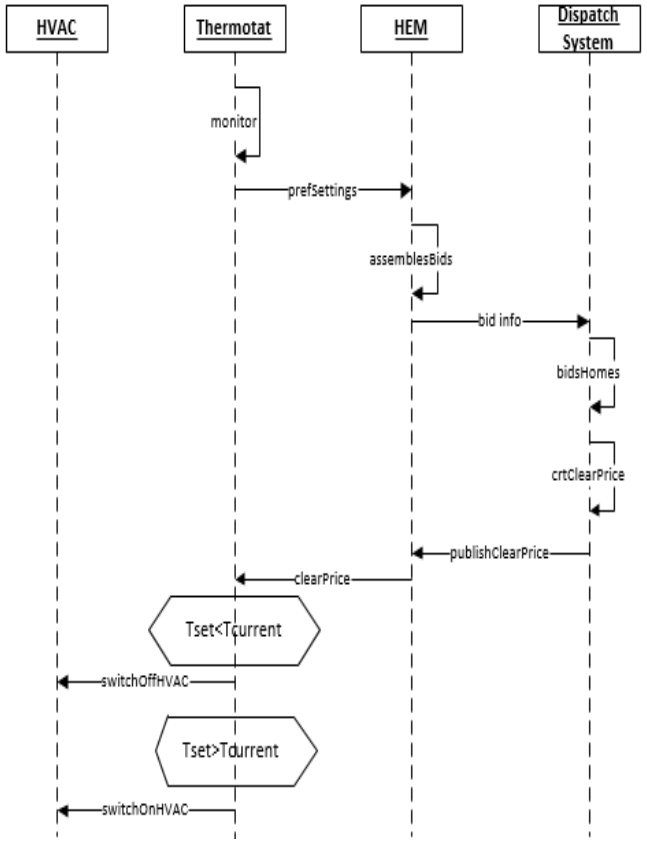


Figure. 11 LSCs representing the HVAC state scenario

From the above LSCs representation, its quiet easy to determine both the expected and unexpected scenario when a particular variability context is selected e.g. if due to blackout or malfunctioning of the thermostat agent the attribute values of set temperature and current temperature are lost then how should the HEM behave or how the bid from the house submitted. So situations like blackout and malfunctioning of thermostat agent should be handled.

Also from the above representation it's quite clear that monitoring is done by thermostat agent, assembling of bids is done by the HEM and dispatch system collects bids from homes and creates cleared price. So to ensure these functionalities are not ambiguous corresponding systems should have adequate measures in place.LSC representation also makes it quite clear what is being communicated between different components e.g. preference settings communicated from thermostat to HEM and cleared price published from Dispatch system to HEM so the system should ensure this communication is happening in a desired manner and appropriate measures should be taken to communicate its failure to the end user in a friendly way .

Applying our framework to select different variability contexts and construct LSCs it would be easy to determine various unexpected scenarios and increase the robustness of the system by handling such scenarios and also assist in requirements elicitation.

8. Conclusion

The paper describes an attempt to determine variability in RTP systems in smart grid using goal models, problem frames, use case maps and live sequence charts. The integrated approach uses i* goal model as in it the relationship between actors involved and their tasks becomes more explicit while the problem diagrams were used to model different variabilities in the behavior of the actors involved in order to achieve a goal. But the variability analysis using problem frames does not help the requirement engineer to analyze the entire scenario. Rather on extending the variability analysis using use case maps and live sequence charts it facilitates reasoning about possible unwanted interactions of scenarios. Usage of Use case maps to integrate many scenarios for a selected context and enabled reasoning about potential undesirable scenarios. Further UCMs acted as a bridge between problem frames and high level system design i.e. LSCs and also provided an abstract view of structural dynamics.LSCs also helped to model the dynamic behavior of a context in order to achieve a goal by providing a more expressive representation of the scenario.LSCs are more enriched with internal conditions on values of attributes, states of the involved objects and refined activation conditions. By taking into account expected attribute values and states when entering a given scenario helped in figuring out the unexpected conditions on selecting a particular variability in problem diagram.

In summary, the combination of goal models, problem frames, use case maps and live sequence charts provides a novel way towards analyzing the variabilities of the RTP system along with scenarios. The variability factors of different actors involved in the system were identified these variabilities were used to represent the contextual scenario using UCM and LSCs which helped in identifying the many interactions which could be the cause of problems could be avoided. The overall contribution of our approach can be summarized as follows modeling variability in context in

detail by representing the system behavior in a particular context using UCM and analyzing the unexpected conditions using LSCs. It also helped to increase the context completeness and context coverage of requirements.

Acknowledgements

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (No. 2013M3C4A7056233).

This research was supported by the MISP(Ministry of Science, ICT & Future Planning), Korea, under Graduate School of Software support program(R0346-15-1017) supervised by the IITP(Institute for Information & communications Technology Promotion).

References

[1] Spees, Kathleen, and Lester B. Lave. "Demand response and electricity market efficiency." *The Electricity Journal* 20.3 (2007): 69-85.

[2] Jackson, Michael. *Problem frames: analysing and structuring software development problems*. Addison-Wesley, 2001.

[3] Liu, Lin, and Eric Yu. "From requirements to architectural design-using goals and scenarios." *ICSE-2001 Workshop: From Software Requirements to Architectures (STRAW 2001) May*. 2001.

[4] Van Lamsweerde, Axel. "Goal-oriented requirements engineering: A guided tour." *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*. IEEE, 2001.

[5] Gross, Daniel, and Eric Yu. "From non-functional requirements to design through patterns." *Requirements Engineering* 6.1 (2001): 18-36.

[6] Jackson, Michael. "Problem frames and software engineering." *Information and Software Technology* 47.14 (2005): 903-912.

[7] Pohl, Klaus. *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010.

[8] Joshi, Hemant I., and Hemish R. Choksi. "Development of Infrastructure for Residential Load to Reduce Peak Demand and Cost of Energy in Smart Grid."Development 3.3 (2015).

[9] Bleistein, Steven J., Karl Cox, and June Verner. "Modeling business strategy in e-business systems requirements engineering." *Conceptual Modeling for Advanced Application Domains*. Springer Berlin Heidelberg, 2004. 617-628.

[10] Bleistein, S., Karl Cox, and June Verner. "Problem frames approach for e-business systems." *1st International Workshop on Advances and Applications of Problem Frames*. 2004.

[11] Bleistein, Steven J., Karl Cox, and June Verner. "Validating strategic alignment of organizational IT requirements using goal modeling and problem diagrams." *Journal of Systems and Software* 79.3 (2006): 362-378.

[12] Bleistein, Steven J., Karl Cox, and June Verner. "Requirements engineering for e-business systems: integrating Jackson problem diagrams with goal modeling and BPM." *Software Engineering Conference, 2004. 11th Asia-Pacific*. IEEE, 2004.

[13] Bleistein, Steven J., Karl Cox, and June Verner. "Strategic alignment in requirements analysis for organizational IT: an integrated approach. *"Proceedings of the 2005 ACM symposium on Applied computing*. ACM, 2005.

[14] Phalp, Keith, and Karl Cox. "Picking the right problem frame—an empirical study." *Empirical Software Engineering* 5.3 (2000): 215-228.

[15] Jackson, Michael. *Problem frames: analysing and structuring software development problems*. Addison-Wesley, 2001.

[16] GRL (2004). Goal-oriented requirements language. http://www.cs.toronto.edu/km/grl/. Last accessed November, 2004.

[17] Yu, Eric SK."Modeling organizations for information systems requirements engineering." *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*. IEEE, 1993.

[18] Peffer, Therese, et al. "How people use thermostats in homes: A review."*Building and Environment* 46.12 (2011): 2529-2541.

[19] U.S. Department of Energy Residential energy end-use by fuel. Table 2.15., (2011). [Online]. Available: http://buildingsdatabook.eren.doe.gov

[20] Meier, Alan K. "Residential thermostats: Comfort controls in California homes." *Lawrence Berkeley National Laboratory* (2008).

[21] Albadi, Mohamed H., and E. F. El-Saadany. "A summary of demand response in electricity markets." *Electric power systems research* 78.11 (2008): 1989-1996.

[22] Faruqui, Ahmad, and Sanem Sergici. "Household response to dynamic pricing of electricity: a survey of 15 experiments." *Journal of Regulatory Economics* 38.2 (2010): 193-225.

[23] Demand Responsive Control of Air Conditioning via Programmable Communicating Thermostats (PCTs), Codes and Standards Enhancement(CASE), California Energy Commission, (2006).

[24] Buhr, Ray JA, Ron S. Casselman, and Ron Casselman. "Use case maps for object-oriented systems." (1996).
[25] Buhr, Raymond JA. "Use case maps as architectural entities for complex systems." *Software Engineering, IEEE Transactions on* 24.12 (1998): 1131-1155.
[26] Amyot, Daniel, et al. "Use case maps for the capture and validation of distributed systems requirements." *Requirements Engineering, 1999. Proceedings. IEEE International Symposium on*. IEEE, 1999.
[27] Marelly, David Harel Hillel Kugler Rami. "The Play-in/Play-out Approach and Tool: Specifying and Executing Behavioral Requirements." (2002).
[28] Del Bianco, Vieri, and Luigi Lavazza. "Enhancing problem frames with scenarios and histories in UML-based software development." *Expert Systems* 25.1 (2008): 28-53.
[29] Del Bianco, Vieri, and Luigi Lavazza. "Enhancing problem frames with scenarios and histories: a preliminary study." *Proceedings of the 2006 international workshop on Advances and applications of problem frames*. ACM, 2006.
[30] Choppy, Christine, and Gianna Reggio. "A UML-based approach for problem frame oriented software development." *Information and software technology*47.14 (2005): 929-954.
[31] Jin, Zhi. "Integrating goals and problem frames in requirements analysis." in Proceedings of the 14th IEEE International Conference on Requirements Engineering (RE), 2006, pp. 349-350.
[32] Tsumaki, Toshihiko, and Tetsuo Tamai. "Framework for matching requirements elicitation techniques to project characteristics." *Software Process: Improvement and Practice* 11.5 (2006): 505-519.
[33] Widergren, Steven E., et al. "AEP Ohio gridSMART demonstration project real-time pricing demonstration analysis." *PNNL Report* 23192 (2014).
[34] Sibay, German, Sebastian Uchitel, and Victor Braberman. "Existential live sequence charts revisited." *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*. IEEE, 2008.

모델 기반 소프트웨어공학에 입각한 요구사항 추적성 메타모델 정의법 개관

변성훈, 이석원

아주대학교 소프트웨어특성화대학원, 아주대학교 소프트웨어융합학과
경기도 수원시 영통구 월드컵로 206
{hooon3, leesw}@ajou.ac.kr

요약: 소프트웨어공학 및 요구공학 영역에서의 비기능적 요구사항(Non-Functional Requirement)인 추적성(Traceability)은 소프트웨어 시스템의 유지 및 보수에 필수적인 속성으로 여겨져 왔다. 특히, 추적성이 실무에서는 작은 결함도 용인되지 않는 안전성이 중요시되는 영역(Safety-Critical Domain)에서 필수적인 속성이다. 추적성을 훌륭하게 구축한 소프트웨어 프로젝트는 소프트웨어 산출물(Artifact) 간의 연관성 이해와 변경 요청 관리(Change Request management)가 용이하게 된다. 모델 기반 공학(MDE: Model-Driven Engineering) 혹은 모델 기반 개발(MDD: Model-Driven Development)을 적용하여 추적성을 달성하고자 하는 연구가 활발히 진행되어 왔으며, 추적성 메타모델(Traceability Meta-model)을 정의하는 것이 핵심 논제로 여겨져 왔다. 본 논문에서는 소프트웨어 개발 과정에서의 요구사항과 산출물 간의 추적성 달성을 위한 현재까지의 연구를 개관하고, 추적성 메타모델의 정의 방법에 대한 향후 연구의 필요성을 제시하고자 한다.

핵심어: 추적성(Traceability), 모델 기반 공학(Model-Driven Engineering), 메타모델(Meta-model), 안전성 요구사항(Safety Requirements)

1. 연구배경

1.1 추적성(Traceability)

소프트웨어 요구사항 추적성은 지난 20 년 간 많은 연구자들에 의해, 끊임없이 연구되어 왔다. 요구사항 추적성은 요구사항의 일생을 따라가고 표현할 수 있는 소프트웨어 품질 속성(Quality Attribute)이다. 개발 초기의 고객의 요구에서부터, 명세화, 개발, 배포, 그리고 유지 및 보수 전 과정에 걸쳐 추적할 수 있어야 한다. 또한, 고객의 요구로부터 수반되는 개발 산출물을 추적하는 것뿐 아니라, 개발 산출물로부터 그와 연관된 최초 고객 요구사항을 추적하는 역방향으로의 추적성도 포함한다[1]. 특히, 요구사항 추적성은 안전성 요구사항 명세에서 중요한 속성이다. 그 이유는, 안전성 요구사항이 어디에서 기원하였는지를 보여주고, 안전성 요구사항의 존재 근거를 제공하기 위해서이다[9]. 요구사항 추적성이 소프트웨어 프로젝트의 성공에 주요한 영향을 미침에도 불구하고, 요구사항 추적성은 좋은 방법론 및 도구(tool)의 부족과 추가적인 인력 자원이 요구되기에 자주 무시된다[2, 3]. 소프트웨어 요구사항 추적성은 전세계의 많은 연구자들 마다 다른 접근법을 시도해 왔고, 통일된 방법론과 지침이 존재하지 않았다. 그렇기에, 소프트웨어 요구사항 추적성은 항상 충실하게 달성되지 않는 경우가 많아 왔다. Knethen 과 Paech 에 따르면, 추적성을 달성하려는 목적의 불분명함도 추적성에 대한 접근법을 제한시키는 요소라고 한다[4]. 추적성과 관련된 작업에 대한 이유와 목적을 분명히 해야 한다. 또한, 어떠한 추적성 접근법을 제안할 때에는, 반드시 추출되어야 하는 산출물 타입들(artifact types)과 그것들의 관계들(relations)을 분명하게 정의해야 한다. 이 추적성에 관련된 산출물 타입들과 그것들의 관계를 정의한 모델을 추적성 메타모델(Traceability Meta-model)이라고 할 수 있다. 이 추적성 메타모델은 많은 연구자들에 의해 만들어져왔다. 이 메타모델은 실무자들이 소프트웨어 시스템의 개발을 진행할 때에, 추적성을 추출하고 문서화 · 명세화 하는 데에 있어, 지침이 될 수 있다.

1.2 모델 기반 공학(Model-Driven Engineering)

적지 않은 기간에 걸쳐, 추적성은 여러 종류의 소프트웨어 개발 방법론(Software Development Methodology)에 있어 중요하게 논의되어 왔다. 그 중에 가장 주목을 받은 분야는, 모델에 기반한(Mode-Driven) 방법론 분야이었다. 모델 기반 공학(Model-Driven Engineering)은 소프트웨어 개발 단계의 주요 산출물로써 모델을 사용하는 분야이다. 그리고, 모델 기반 공학은 소프트웨어 개발 프로세스를 모델 변형(Model Transformation)을 통해 일정 부분 자동적으로 수행한다. 모델 변형의 과정과 결과를 기록하는 것이 굉장히 중요하고, 그 기록들은 추적성의 기반이 된다[5]. Model-Driven Engineering 과

Model-Driven Development 는 같은 개념으로 봐도 무방하다[5]. 소프트웨어 시스템이 모델의 변형을 통해 개발되기 때문에, 어느 한 모델의 변경(change)은 다른 모델에 영향을 미치게 된다. 각 모델 간의 변경 영향력 분석(change impact analysis)를 위해서는 추적성이 요구된다. 또한, 추적성은 소프트웨어 개발 프로세스 내에서의 각 단계의 변형을 문서화하는(documentation) 것에서 달성되기 때문에, 모델 기반 공학은 추적성 달성에 적합한 소프트웨어 프로세스라고 할 수 있다[5, 6]. 결국 이 모델 기반 공학 접근법 내에서는, 소프트웨어 산출물들(요구사항, 설계 명세서, 코드 등)의 모델이 메타모델에 의해서 추출된다. 훌륭한 메타모델이 개발되어 있다면, 실무자들은 개발 과정에서 각 산출물을 정의하는 데에 도움을 얻을 수 있다. [5]에 따르면 하나로 표준화된(standardized) 추적성 메타모델이 존재하지 않는다.

본 논문은, 2 장에서 모델 기반 공학을 적용하여 요구사항 추적성을 달성하고자 하는 연구들의 골자와 그 연구들의 추적성 메타모델을 개관한다. 그리고, 3 장에서 추적성 메타모델 개발을 위한 향후 연구의 필요성을 언급하고, 4 장에서 결론을 맺는다.

2. 관련 연구

본 장에서는 모델 기반 공학을 적용하여 요구사항 추적성에 메타모델에 대한 접근을 한 연구들을 개관하고자 한다.

[7]에서 정의한 추적성 메타모델은 TraceModel, TraceableArtifactType, ArtifactTraceType, 그리고 RelationTraceType 의 4 가지 클래스를 정의한다. 그림.1 은 4 가지 클래스를 나타낸다.

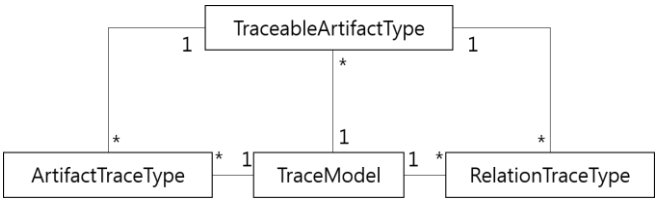


그림.1 추적성 메타모델 1

각각의 클래스가 구체화하는 것은 다음과 같다.

- TraceModel 은 특정한 산출물 타입을 정의하고, 그와 관련되어 추적되어야 하는 산출물과 관계를 정의한다. 한 개의 TraceModel 은 여러 개의 산출물 타입을 정의할 수 있다.
- TraceableArtifactType 은 TraceModel 의 실제 하나의 산출물 타입으로의 매핑(mapping)이다. 하나의 산출물 타입은 여러 개의 추적되어야 하는 산출물과 여러 개의 추적성 관계들로 구성된다.
- ArtifactTraceType 은 어떤 하나의 산출물 타입이 추적할 수 있어야 하는 실제 산출물들을 정의한다.
- RelationTraceType 은 산출물들 간의 가져야 하는 특정한 관계들을 정의한다.

[7]의 추적성 메타모델은 기본적인 표본(prototype)을 보여주는 연구 결과이며, 추적성 모델을 추적성 저장고(Repository)에 축적, 저장하고 사용하는 접근법을 역설한다.

다른 연구를 살펴보자면, [8]에서는 모델 변형(Model Transformation)의 과정에서 추적성 모델(Traceability Model)이 생성되고, 그 추적성 모델은 추적성 메타모델을 따른다고 설명한다. 이 논문에서 제시하는 모델 변형 프로세스는 다음과 같다. 요구사항 모델에서 콤포넌트 모델(component model)로, 콤포넌트 모델에서 UML 클래스 다이어그램으로, 그리고 UML 클래스 다이어그램에서 코드 구현(code implementation)으로의 모델 변형을 제시한다. 이러한 각 단계에서의 모델 변형 과정에서 추적성 모델이 생성되며, 이 추적성 모델들을 기술하는 데에 추적성 메타모델이 사용된다. [8]에서 정의한 추적성 메타모델은 그림.2 와 같다. 그림.2 는 [8]에서 발췌한 것이다.

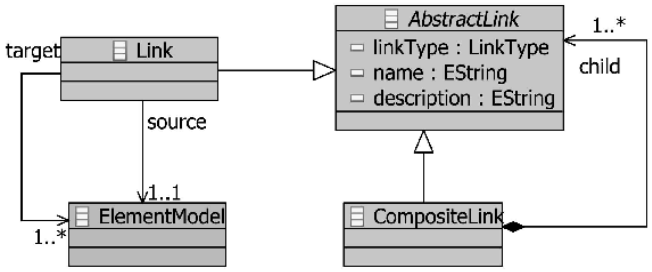


그림.2 추적성 메타모델 2

각각의 클래스가 구체화하는 것은 다음과 같다.

- Link 는 하나의 source ElementModel 을 하나 혹은 여러 개의 target ElementModel 과 연결한다. AbstractLink 의 자식클래스로써, 요소들 간의 관계를 정의한다.
- ElementModel 은 각각의 추적가능한 요소(traceable element)를 정의한다.
- AbstractLink 는 추적 관계(Trace Relations)를 정의하는 기본적인 속성(Attribute)들을 가지고 있는 상위 클래스이다. LinkType 속성은 존재하는 관계들을 범주화(categorize)하는 데에 도움을 준다.
- CompositeLink 는 개별의 Link 들을 다른 하나의 복잡한 그룹으로 모으는 역할을 한다. 이 CompositeLink 는 다른 수준의 결집도

(granularity)를 정의하는 데에 목적이 있다.

그림.3 은 추적성 메타모델로부터 하나의 추적성 모델이 추출됨에 따라 확인할 수 있는 Link 들의 타입에 관한 예시이다. 그림.3 은 [8]에서 발췌한 것이다. 상단에 있는 요소들은 추상화 수준이 N+1 인 소스 요소들이고, 하단에 있는 요소들은 추상화 수준이 N 인 타겟 요소들이다. 소스 요소들은 추적(trace)이 적용되지 않은 요소도 있고, 하나의 소스 요소로부터 여러 개의 타겟 요소로 Link 된 요소들도 있으며, 여러 개의 소스 요소가 하나의 타겟으로 Link 된 요소들도 있다. 타겟 요소들 또한, 추적이 적용되지 않은 요소도 존재하며, 여러 개의 소스 요소와 동시에 Link 된 타겟 요소들도 존재한다. 또한 어떻게 요소들을 그룹화하느냐에 따라 다른 Composite Link 가 생성되는 것을 보여준다.

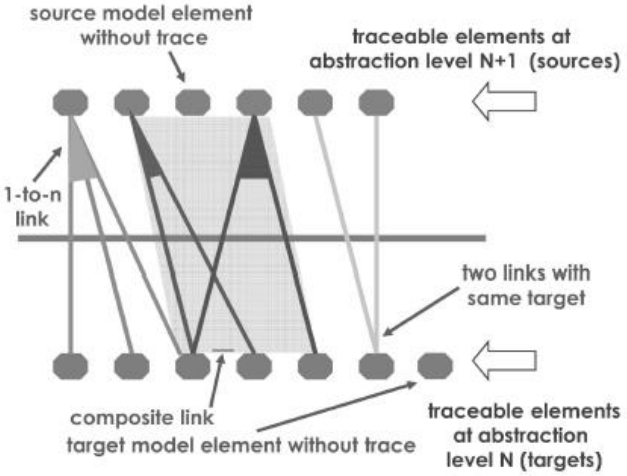


그림.3 링크 타입의 예

여기서 소개한 [8]에서의 추적성 메타모델은 앞서 설명한 [7]에서의 추적성 메타모델과 비교하였을 때, [7]이 일반적인 메타모델의 표본을 보여주었다면 [8]은 좀 더 관계([8]에서는 Link 로 , [7]에서는 Relation 으로 표현됨)와 그 관계들의 granularity 에 중점을 둔 Composite Link 를 설명하였다. 하지만, [8]에서 제시된 메타모델의 Composite Link 가 정확히 어떻게 그룹화되었을 때에 어느 정도의 granularity level 을 가지게 되는지는 모호하다. 요소들 간의 그룹화 기준 혹은 방식을 더욱 구체적으로 설명하고, granularity level 을 사전에 명세화할 필요가 있다고 보여진다.

또 다른 연구를 살펴보자면, [5]에서 언급하는 추적성 메타모델은 그림.4 와 같다. 그림.4 는 [5]에서 발췌한 것이다. 이 논문의 저자는 그림.4 의 메타모델을 기본 추적성 메타모델(The Basic Traceability Meta-model)이라고 표현한다. 그 이유는, 이 메타모델은 OMG(Object Management Group)의 모델 기반 공학에 대한 관점에서 표현되었기 때문이다[10]. 각각의 클래스가 구체화하는 것은 다음과 같다

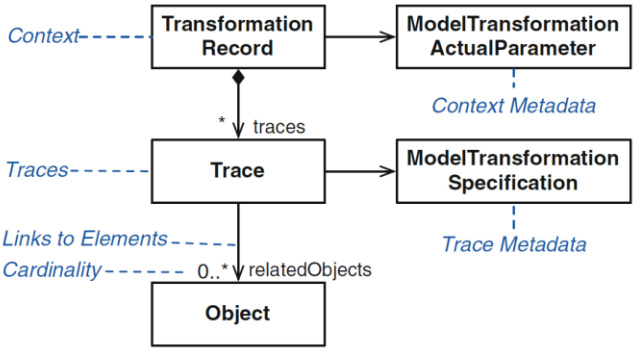


그림.4 추적성 메타모델 3

- Object 는 요소, 링크, 그리고 원시 타입으로 이루어지며, 모델 변형의 결과로 생성되는 Output 모델이다.
- Model Transformation Specification 과 이것이 가지고 있는 요소들을 기반으로 하여 모델 변형이 실행된다. 이것은 Input 모델의 파라미터와 Output 모델의 파라미터 모두를 바인딩한다.
- Trace 는 Input 모델들과 Output 모델들의 링크를 기록한다. 이 링크는 Model Transformation Specification 의 요소들을 기반으로 한다.
- Transformation Record 는 Input 모델의 object 들과 Output 모델의 object 들의 링크이며, 연결된 속성 자체를 나타낸다.
- Model Transforamtion Actual Parameter 는 모델 변형을 위해 명시되어야 할 실제로 사용되는 인수들에 대한 정의이다.

[8]에서 언급된 추적성 메타모델은 모델 변형의 관점에서 상세히 묘사하였다. 모델변형의 과정을 정확히 기록하고, 연관된 산출물들을 명세화하는 것은 추적성 달성에 필수 요소라고 역설하며, 이에 대한 메타모델을 제시하였다.

이번 장에서, 모델 기반 공학 프레임워크 내에서의 추적성 메타모델에 대한 몇몇의 연구들을 개관하였다. 추적성 구축을 위해, 많은 연구들은 소프트웨어 산출물과 산출물들 간의 연관성을 기록하는 것에 집중해왔다. 그 해결을 위해, 모델 기반 공학을 적용하였고, 추적성 메타모델을 정의하는 것이 굉장히 중요한 것으로 여겨졌다. 하지만, 각 연구자 및 실무자들마다 고유한 메타모델 접근법을 시도하는 경우가 많고, 표준화된 추적성 메타모델은 존재하지 않는다[5].

3. 추후 연구

2 장에서 언급되었듯이, 모델 기반 공학 내에서의 추적성 구축은 추적성 메타모델을 정의하는 것이 중

요하다. 본 논문에서 제시하고자 하는 연구 방향은, 추적성 메타모델에 정의된 소프트웨어 산출물과 그것들의 관계에 대한 용어에, 어의론(Semantics)를 부여하여 새롭게 메타모델을 정의하는 것이다. 의미론 혹은 어의론이라고 할 수 있는 Semantics 는, 구조적으로 대상을 분석하는 구문론(Syntax)와 다르게 대상의 의미를 분석하여 다른 개념을 추론할 수 있다. 본 저자는 안전성 요구사항에 존재하는 다양한 속성들의 단어 자체에 semtantics 를 부여하는 것으로부터 접근하려고 한다. 안전성 요구사항에는 대표적으로 Hazard, Risk, Probability, Avoidance, Exposure 와 같은 속성들이 있다[8]. 이러한 단어들이 등장할 때, 필수적으로 추출되어야 하는 디자인 패턴 혹은 시스템 아키텍처와의 연관성을 온톨로지(Ontology)를 활용하여 구축하려 한다. Protege 툴을 이용하여, 전형적으로 나타나는 안전성 요구사항과 설계 명세, 그리고 구현 코드 간의 연관되어 있는 성질들을 사전에 구축해 놓는다면, 더 나은 추적성이 달성될 수 있다고 기대한다. 그리고, 이를 통해 안전성 요구사항의 추적성 메타모델을 정의하는 것아 목표이다. 뿐만 아니라, 일일이 수동적으로 명시해야 하는(manual task) 추적성들을 일정 부분 자동적으로 추론하게 하여, 비용 절감에도 도움을 줄 수 있으며, 모델의 자동 변형에도(automatic model transformation) 기여할 것이라 본다. 본 저자가 추후에 사례 연구로써 접근하고자 하는 영역(domain)은, 특히나 안전성이 중요시되는 영역인 원자력 소프트웨어 혹은 자동차 소프트웨어이며, 이 영역의 요구사항들의 추적성을 온톨로지를 통해 구현하고, 메타모델을 제안하고자 한다.

4. 결론

아직까지 실제 현업에서의 소프트웨어 프로젝트는 각각의 고유한 추적성 접근법을 시도하고 있거나, 아예 추적성이 무시되는 경우가 많다. 연구자들 또한, 자기들만의 고유한 접근법을 시도하는 경우가 많다. 하지만, 모델 기반 공학이라는 큰 틀 안에서의 시도는 모두 통일되어 가고 있다. 특히나, 안전성이 중시되는 영역에서는 추적성을 달성하려는 노력이 굉장히 많이 이루어지고 있다.

본 논문에서는, 훌륭하게 정의된 추적성 메타모델이(Well-defined Traceability Meta-model) 필요하다는 관점으로 접근을 시도하였다. 추적성 메타모델은 소프트웨어 추적성을 도출하는 데에 있어, 강력한 도구가 될 수 있다. 그렇기 때문에, 본 논문에서는 현재까지의 추적성 메타모델에 관한 연구를 정리하고, 본 논문의 저자가 접근하고자 하는 향후 연구 방향을 제시하였다. 향후 연구 방향은 추적성 메타모델과 semantics 의 융합으로 정의할 수 있다.

Acknowledgements

이 논문은 2013 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (NO.2013R1A1A2009801).

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW 특성화대학원 지원 사업의 연구결과로 수행되었음 (R0346-15-1017).

참고문헌

[1] Orelena C. Z. Gotel, Anthony C. W. Finkelstein, “An Analysis of the Requirements Traceability Problem”, Proceedings of the First International Conference on Requirements Engineering, 1994.

[2] Vikash Katta, Tor Stalhane, “A Conceptual Model of Traceability for Safety Systems”, Electronic Proc. 2nd Complex Systems Design & Management Conference (CSD&M 2011), Poster-Session Paper, 2011.

[3] Kannenberg A, Saiedian H, “Why Software Requirements Traceability Remains a Challenge”, The Journal of Defense Software Engineering, July/August edn. CrossTalk: 14-19, 2009.

[4] Knethen AV, Paech B, “A Survey on Tracing Approaches in Practice and Research”, Research Report, ESE-Report, 095.01/E, Fraunhofer IESE, Kaiserslautern, 2002.

[5] Stefan Winkler, Jens von Pilgrim, “A Survey of Traceability in Requirements Engineering and Model-Driven Development”, Journal Software and Systems Modeling (SoSyM) archive Volume 9 Issue 4, September 2010 Pages 529-565, 2010.

[6] Ismenia Galvao, Arda Goknil, “Survey of Traceability Approaches in Model-Driven Engineering”, 11th IEEE International Enterprise Distributed Object Computing Conference, 2007. EDOC 2007.

[7] Stale Walderhaug, Ulrik Johansen, Erlend Stav, Jan Aagedal, “Towards a Generic Solution for Traceability in MDD”, ECMDA Traceability Workshop (ECMDA-TW) (pp. 41-50), 2006.

[8] Pedro Sanchez, Diego Alonso, Francisca Rosique, Barbara Alvarez, Juan A. Pastor, “Introducing Safety Requirements Traceability Support in Model-Driven Development of Robotic Applications”, IEEE Transactions on Computers, 2010.

[9] Vikash Katta, Christian Raspotnig, Peter Karpati, Tor Stalhane, “Requirements Management in a Combined Process for Safety and Security Assessments”, Eighth International Conference on Availability, Reliability and Security (ARES), 2013.

[10] Object Management Group, “A Proposal for an MDA Foundation Model”, Object Management Group, Needham, ormsc/05-04-01ed, 2013.

환경정보를 고려한 자가적응형 시스템을 위한 동적 의사결정 기술

김미수, 정호현, 이은석

성균관대학교 전자전기컴퓨터공학과
경기도 수원시 장안구 서부로 2066
{misoo12, jeonghh89, leees}@skku.edu

요약: 본 논문에서는 시스템 환경정보와 목표모델을 동적 결정 네트워크로 설계하고, 설계된 모델을 실시간 정보를 통해 갱신하여, 동적 환경에 적절히 대응 가능한 자가적응형 시스템을 위한 동적 의사결정 기술을 제시한다. 제안 방법론의 검증을 위해 로보코드에 적용하여 그 유효성을 확인하였다.

핵심어: 자가적응형 시스템, 동적 의사결정, 목표모델, 동적 결정 네트워크

1. 서론

기존 자가적응형 시스템 연구들은 설계 시점에 정의된 의사결정 기술을 사용하여 적응 정책을 결정한다[1]. 그러나 설계 시점에 시스템이 배치될 환경에 대해 완벽히 예측할 수 없기 때문에 실행 환경에 따른 적절한 적응성을 보장하기 어렵다. 상기 문제를 해결하기 위해서는 실시간 시스템의 목표들과 환경정보들을 동시에 고려하는 동적 의사결정 기술이 필요하다. 위의 필요성에 따라 동적 의사결정을 지원하는 동적 결정 네트워크(DDN, Dynamic Decision Network)[2]를 이용한다. 구체적으로 기존 연구[3]를 확장하여, 환경 정보를 고려한 목표모델기반 DDN 을 설계하고, 설계된 모델에 실시간 환경 정보를 지속적으로 반영한다. 결과적으로 시스템 목표 만족도의 정확한 예측을 통해 목표를 지속적으로 달성하는 환경정보를 고려한 자가적응형 시스템을 위한 동적 의사결정 기술을 제안한다.

2. 환경정보를 고려한 자가적응형 시스템을 위한 동적 의사결정 기술

그림 1 은 제안방법의 구성도를 보여준다.

2.1 확장된 DDN 설계 방법

우선 시스템이 관찰할 수 있는 변수들을 정의하고 도메인 전문가나 구조 학습 알고리즘들을 통해 환경정보를 표현하는 베이지안 네트워크로 생성한다.

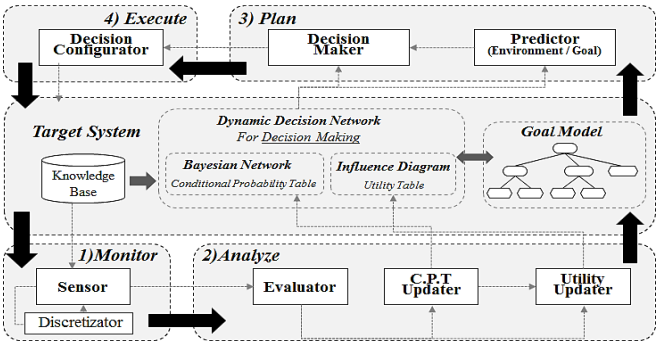


그림 1 환경정보를 고려한 동적 의사결정 방법의 구성도

다음으로 목표 모델의 목표 계층을 유틸리티 노드 계층으로, 수행 가능한 행동들은 결정 노드로, 목표 만족 기준의 시스템 변수들을 확률 노드로 표현하고 유틸리티 노드에 연결하여 영향도 다이어그램을 생성한다. 말단 유틸리티 노드는 행동에 따라 얻는 획득 가치를 표현하고, 유틸리티 노드들 사이의 관계는 상위 목표를 위한 하위 목표들의 가중치를 표현한다.

마지막으로 영향도 다이어그램의 확률 노드와 베이지안 네트워크의 관련 변수를 연결하여 통합하고, 과거 데이터를 통해 은닉마코브모델(Hidden Markov Model)을 생성하여 동적 결정 네트워크를 생성한다.

2.2 동적 의사결정 기술

관찰 단계에서는 시스템의 센서들로부터 수집 가능한 정보들의 이산화를 수행한다.

분석 단계에서는 정책이 수행된 이후 목표의 만족도를 평가하고, 목표가 만족되지 않을 경우 의사결정 모델을 갱신한다. 첫째로 정책 수행 전 가장 높은 확률을 보인 환경정보의 예측 값과 정책 수행 후 나타난 환경정보의 결과의 차이를 Dirichlet Distribution[4]에 기반하여 조건부 확률 테이블에 반영한다. 다음으로 이후에 동일한 행동을 수행했을 때의 가치를 예측하기 위해서 과거와 현재의 획득 가치와 k-NN[5]을 사용하여 유틸리티 테이블을 갱신한다.

의사결정 및 수행 단계에서는 다음 시점의 환경정보와 예측된 환경에서 정책의 가치를 예측하여, 가장 높은 가치가 예측되는 정책을 결정하고 수행한다.

3. 실험

제안 방법론의 유효성을 입증하기 위해 로봇간 포격 전투 시뮬레이션 도구인 로보코드[6]에 적용한다. 설계 시점에 가정한 적 로봇의 전략 외의 상황에서 적절한 행동을 수행하기 어려운 로봇의 문제를 해결하기 위해 본 방법을 적용하여 유효성을 확인한다.

3.1 실행 환경에 따른 적응성

실행 환경에 따른 적응성을 검증하기 위해, 한 로봇을 가정하고 설계된 DDN 이 제안 갱신 기법을 통해 다른 로봇과의 전투에서 활용가능한지 확인한다.

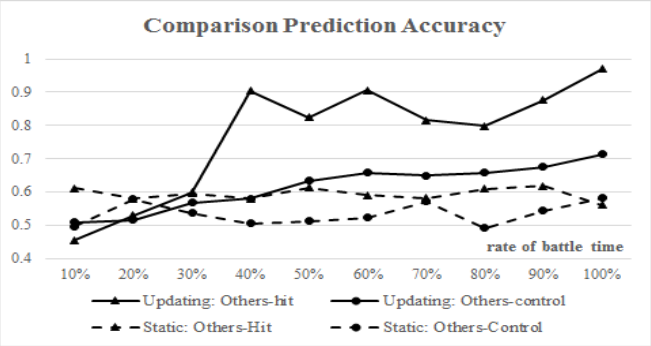


그림 2 다른 로봇들과의 예측 정확도 변화 비교

표 1 다른 로봇의 실시간 갱신 기반 평균 예측 정확도

	DDN		Updating DDN	
	Hit	**Control**	**Hit**	**Control**
SpinBot	54.61%	72.8%	63.8%	77.1%
Corner	67.47%	63.95%	67.9%	63.9%
FireBot	69.76%	40.48%	78.8%	53.7%
Crazy	43.74%	43.71%	88.9%	47%
FoilistMicro	61.62%	44.61%	65.4%	49%
Average	**59.44%**	**53.1%**	**72.96%**	**58.14%**
Effect			**+18.53%**	**+8.66%**

그림 2은 다른 로봇과의 전투에서, 총 전투 시간의 비율에 따라 각 목표에 대한 평균 예측 정확도의 변화 추이를 보인다. 표 1 은 갱신 기법을 적용하지 않는 DDN 과 적용한 로봇들에 대해 적용하여 얻은 평균 목표 예측 정확도이다. 정적 모델 기반 예측 정확도 대비 평균 13.595% 향상되었다.

3.2 동적 의사결정 기술

표 2 은 대상 로봇과 다른 로봇과의 대결 결과와 제안 방법을 적용한 로봇과 다른 로봇과의 대결을 수행한 후 얻은 대결 결과를 비교한다. 제안 방법을 통해 적 로봇의 점수는 11.1% 감소하고, 적용 로봇은 18.7% 향상된 점수를 획득하는 것을 확인하였다.

표 2 다른 로봇과의 대결 결과 점수 비교

	vs Base Bot (Epeest)		Vs DDN Bot	
	Enemy	Mine	Enemy	Mine
SpinBot	32	1387	16	1577
Corner	36	1345	34	1787
FireBot	23	1471	8	1754
Crazy	104	1614	103	1702
FoilistMicro	681	717	617	934
Average	175.2	1306.8	155.6	1550.8
Effect (%)			**-11.1%**	**+18.7%**

4. 결 론

본 논문에서는 설계 시점에 정의된 정적 의사결정 기술의 문제를 해결하기 위해, 환경정보와 목표모델을 통한 동적 결정 네트워크의 설계 방법과 실시간 정보를 의사결정 모델에 반영하는 환경정보를 고려한 자가적응형 시스템을 위한 동적 의사결정 방법을 제안하고 로보코드에 적용하여 그 효과를 증명하였다. 실험 결과, 갱신 기법을 통해 정적 의사결정 모델 대비 목표 만족도의 예측 정확도가 평균 14.643% 향상되었다. 마지막으로 정적 유틸리티 기반 의사결정과 비교하여, 적 로봇의 점수는 평균 11.1% 감소하고, 18.7% 향상된 점수를 획득하였다.

Acknowledge

이 논문은 2015 년도 정부(교육과학기술부)의 재원으로 한국연구재단-차세대정보컴퓨팅개발사업의 지원을 받아 수행된 연구임(No. 2015045358).

참고문헌

[1] B. Chen et al., "Requirements-driven self-optimization of composite services using feedback control." *IEEE Transactions on Services Computing*, Vol. 8, No. 1, pp. 107-120, 2015.

[2] S.J. Russell et al., Artificial intelligence: A modern approach, 2nd ed., Prentice Hall series in artificial intelligence. Prentice Hall. 2003.

[3] M. Kim et al., "An extended dynamic decision network." *Journal of Korean Institute of Information Scientists and Engineers (KIISE): Software and Applications*, Vol. 42, No. 7, pp. 889-900. 2015.

[4] T. T. Womg, "Generalized Dirichlet distribution in Bayesian analysis." *Applied Mathematics and Computation*, Vol. 97, No. 2, pp. 165-181, 1998.

[5] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression", *The American Statistician*, Vol. 46, No.3, pp. 175-185, 1992.

[6] ROBOCODE, http://robocode.sourceforge.net/

모바일 어플리케이션 GUI 사용성 저해요소 검출 도구

마 경 욱

서강대학교 컴퓨터학부
서울 마포구 신수동 백범로 35
makw@sogang.ac.kr

박 수 진

서강대학교 미래기술연구원
서울 마포구 신수동 백범로 35
psjdream@sogang.ac.kr

요약: 모바일 도메인에서의 한정된 자원 특성상 기능 요구사항뿐만 아니라 사용성, 성능 등의 요소 또한 모바일 앱 선택의 중요한 요소이다. 그 중, 사용성은 사용자들에게 가장 중요한 요소라 할 수 있다. 이러한 사용성 품질을 향상시키기 위한 테스트 작업이 개발자에 의해 반복적으로 이루어지는 실정이다. 본 논문에서는 모바일 어플리케이션 상에서 발생하는 사용자의 사용 로그로부터 추출 가능한 정보를 분석하여, 모바일 어플리케이션의 GUI 설계상에 내재된 사용성 저해 요소를 자동으로 검출하는 도구를 제안함으로써 반복적인 테스트 작업에 소요되는 시간 비용을 감소시키고자 하였다.

핵심어: 모바일 앱, GUI, 사용성

1. 서 론

그래픽 유저 인터페이스(Graphic User Interface: GUI) 는 데스크탑이나 모바일과 같은 소프트웨어 시스템에서 중요한 역할을 수행하고 있다. GUI 는 사용자들이 직접 접하게 되는 인터페이스이기 때문에 전체 소프트웨어 개발비용의 50-60%가 GUI 개발과 테스트에 할애된다[1]. GUI 의 기능 요구사항뿐만 아니라 사용자들에게 GUI 가 사용이 용이하게 설계되었는지, 효율적으로 기능을 수행할 수 있는지와 같은 비기능 요구사항 역시 테스트의 대상이다. 이러한 비기능 요구사항 중 사용자에게 효율성, 학습성 등을 제공하는 정도를 GUI 사용성(GUI Usability)이라 할 수 있다.

GUI 사용성은 사용자의 주관이 반영되고 실험 대상자에 따라 변하기도 하여 자동화가 어렵다. 그러므로 GUI 사용성을 테스트하기 위한 주된 작업은 수동으로 이뤄지고 있다. 많은 시간이 소요되는 GUI 사용성 테스트를 효율적으로 하기 위하여 사용자 터치를 기록 및 실행[2], 화면 캡쳐를 통한 테스트[3] 등의 연구가 진행되었다. 그러나 주관적 평가에 의존하거나 추가적인 카메라나 녹화된 영상을 처리하기 위한 기기가 필요한 등 테스트에 많은 비용이 소모되는 등의 문제점이 있다. 이러한 문제점을 해결하기 위하여 이번 연구에서는 모바일 앱에서의 GUI 사용성을 실제 사용자들의 행동을 모델링 하여 정량적으로 측정하는 방안을 제안하고자 한다.

본 논문의 구성은 다음과 같다. 2 장에서는 사용성 측정을 위해 진행되었던 선행연구에 대해 살펴본 후 3 장에서는 사용성 저해요소 검출 도구의 구조 및 동작 과정을 설명한다. 4 장에서는 도구를 통해 사용성 저해요소를 검출하는 알고리즘에 대해 기술한다. 마지막으로 5 장에서는 해당 도구의 실효성과 추후 연구 방향을 제시하고 있다.

2. 관련 연구

이전의 연구에서는 시스템의 사용성을 측정하고 향상시키기 위한 방식으로 주로 사용자를 대상으로 설문조사를 수행하였다. [4]의 연구에서는 목표 질의 메트릭(Goal Question Metric)을 도입하여 특정 시나리오를 수행할 때 소요시간, 인식성에 대한 사용자 설문조사를 통한 실험을 수행하였다. 하지만 이 같은 사용자 설문을 기초자료로 사용성을 평가하는 방식은 설문 문항의 구성방식, 응답자의 특성 등의 변수에 따라 그 결과가 상이할 수 있다는 문제점을 가지고 있다. A/B 테스트[5]는 사용자들에게 똑같은 페이지/내용을 제공하지만 메세지의 구체적인 형식이나 디자인에 변형을 주어서 각자의 반응을 살피는 테스트 방식이다. 다양한 GUI 대안(Alternative)에 대한 사용자들의 반응을 얻기 위해서는 대안 숫자만큼의 버전으로 소프트웨어를 제작해야 되기 때문에 테스트에 소요되는 시간이 증가한다. 터치 기록/실행을 수행하는 비교적 저-수준의 연구부터 모델 기반으로 사용성 테스트를 하는 고-수준 접근법이 존재한다. 모델 기반의 접근법은 GUI 설계의 내부 구조를 분석한 후 사용성 모델을 이용하여 변환 규칙을 생성한다. 이러한 규칙을 통해 플랫폼에 독립적인 사용성을 측정하는 방법을 사용한다. 즉, 화면 내 뷰(View)들의 정렬 정도, 위젯 넓이/높이 등을 측정하여 사용자가 편안하다고 느끼는 정도를 가늠할 수 있다. 그러나 다양한 디바이스의 해상도

이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터(No.10044457, 자율지능형 지식/기기 협업프레임워크 기술개발)와 2015 년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보 · 컴퓨팅기술개발사업 (No. 2012M3C4A7033348)의 지원을 받아 수행된 연구임

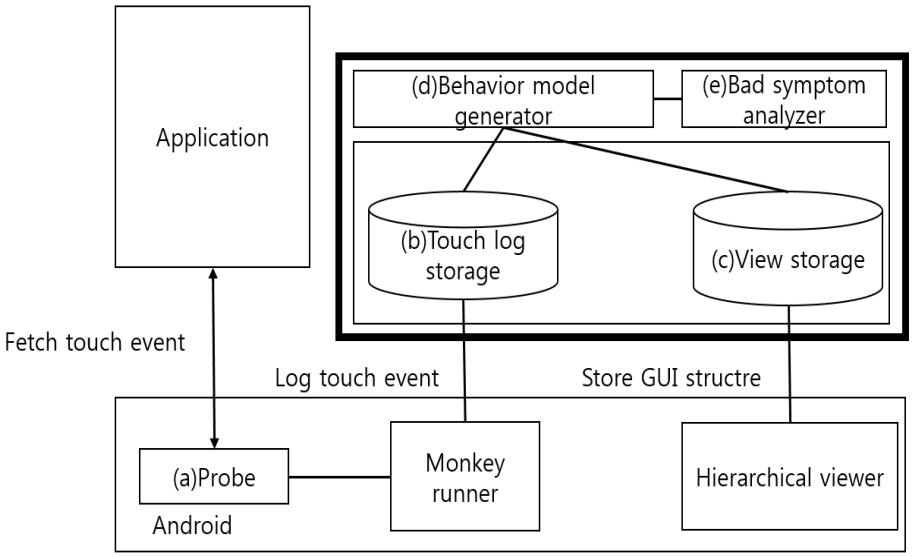


그림 1 사용성 저해요소 검출 도구

를 지원하고, 화면회전과 같은 기능을 지원해야 하는 모바일 앱의 특성상, 해당 측정법으로는 기기의 해상도에 따라 GUI 의 배치가 달라지므로 같은 앱일지라도 메트릭 수치가 급변하여 정확한 측정이 어려운 한계점이 존재한다 본 연구는 설계자가 예상한 사용자 행동 모델과 실제 사용자들의 사용 행동에서 캡쳐된 사용자 행동모델간에 차이점이 발생하는 지점에 모바일 앱의 사용성의 문제점이 내재해 있다는 것을 정의하는 것으로부터 시작되었다. 이러한 사용성 검증을 위한 기준을 제시함으로써 객관적인 메트릭을 확보할 수 있으며, 설계자의 의도를 나타내는 모델과 사용자의 행동 모델 비교를 통한 자동화된 사용성 저해 요소 검출이 가능함에 따라, 기존 연구들의 문제점으로 지적된 자의적 해석 가능성과 고비용 소모를 완화시킬 수 있을 것으로 기대한다.

3. 사용성 저해요소 검출 도구(RUID : Real-time Usability Issue Detector)

본 논문에서는 사용자들의 앱 조작을 로그로 기록하고 행동 모델 생성 도구를 개발하였다. 그림 1 은 GUI 사용성 저해 요소 검출 도구의 각 컴포넌트를 도식화하였다. 컴포넌트 별 역할은 아래와 같다.

(a)Probe: Android 플랫폼 상에 내장시켜 Android 플랫폼 상에서 동작하는 모바일 어플리케이션에 사용자가 발생시키는 이벤트들을 패치(fetch)한다 .

(b)Touch log storage: Android 매크로인 Monkeyrunner 를 이용하여 사용자들 이벤트가 발생한 좌표, 타임스탬프, 제스처 종류, 터치한 GUI 의 고유한 ID 값을 수집 하여 저장한다.

(c) View storage: 검사의 대상이 될 GUI ID 정보를 획득해야 하기 위해 Android 도구인 Hierarchical viewer 를 이용하여 현재 화면의 전체 GUI ID 값이 포함된 구조 정보를 XML 형태로 저장한다.

(d) Behavior model generator: (b), (c) 로 부터 검사의 대 상이 되는 GUI 를 선정하고 사용자들의 터치 입력을 모델링 한 후 행동 모델들을 병합하여 사용자들의 행동을 대표할 수 있는 하나의 행동 모델을 생성한다.

(e) Bad symptom analyzer: 병합된 사용자 행동모델과 사전에 정의된 저해요소간의 비교를 통해 행동 모델에서 저해요소의 유무를 판별한다. 개발자들은 전체 GUI 중 저해요소가 발생할 것이라 예상되는 GUI 의 ID 값을 입력한 후, 실제 사용자들에 게 테스트를 진행하여 저해요소를 검출할 수 있다.

해당 도구는 Android 앱을 대상으로 GUI 의 이상 징후를 검출하고 있으며 Java 로 구현되었다.

4. RUID 동작 메커니즘

4-1. 사용자 행동 기록

Android 내부에서 생성하는 관련 로그를 Monkeyrunner 에서 기기에서 발생하는 이벤트 로그를 기록한다. 터치 로그에는 터치 시의 타임스탬프, x, y 좌표, 제스처 유형의 정보를 포함하고 있다. 이 때, 어떠한 GUI 객체에 대해서 조작을 하였는지의 정보를 알아야 특정 GUI 에서 사용성 문제가 발생하는지 파악할 수 있다. 이를 위해, Android 운영체제에 내장시킨 Probe 에서 사용자가 GUI 로그를 저장한다. 그림 2 와 같이 위 정보를 합성하여 어떠한 GUI 에 어

떠한 제스쳐가 행해졌는지를 알 수 있다. 타임스탬프 기반으로 합성된 로그는 (b)Touch log storage 에 저장된다. 또한 사용자가 터치를 수행할 때의 화면상태를 저장하기 위해 View storage 에 계층적 형태의 뷰 정보를 저장한다.

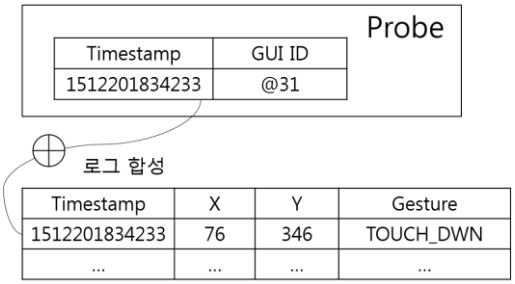


그림 2 사용자 행동 기록 로그

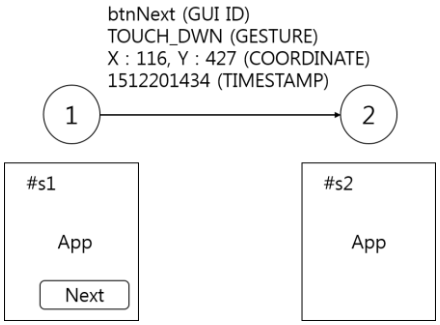


그림 3 행동 모델 상태 및 상태간 전이 표현

4-2. 모델 생성

저장된 로그로부터 FSM(Finite State Machine)을 생성한다. 이 때, 각 상태(State)는 사용자가 모바일 앱에 조작할 때의 화면 상태를 나타내고, 상태 간 전이(Transition)은 화면 상태간 사용자가 취한 행동을 나타낸다. 아래 그림 3 에서 도식화하듯이, 사용자의 한 행동으로부터 하나의 상태간 전이가 발생한다. 이러한 과정을 통해 특정 모바일 앱의 서비스를 실행할 때 행동하는 모델을 생성할 수 있다. 하나의 모델이 가진 상태의 개수가 n 개라 가정할 때, 상태 간 전이의 경우의 수를 고려하여 생성 가능한 행동 모델의 개수는 ${}_nP_2$ 개이다. 그러나 이러한 행동 모델이 전부 다른 의도를 내포하고 있지는 않다. 즉, 다수의 사용자들은 개인별로 모바일 앱을 조작하는 특성을 가지고 있다. 따라서 사용자 행동 모델은 개인의 특성이 반영되어 모두 다른 형태일 수 있으며 이를 기반으로 한 모델도 모두 다른 형태를 가질 수 있다. 따라서 각 사용자들이 아닌 전체 사용자들의 행동 모델을 대표하기 위해서는 각 사용자 별 행동 모델을 병합하는 과정이 필요하다. 병합 과정을 통해 서로 다른 형태의 행동 모델이지만 동일한 의도를 표현하고 있는지를 판단할 수 있다. 이러한 병합된 모델을 통해 사용성 저해 요소 검출의 성능을 향상시킬 수 있다. 이전 연구[6]에서는 각 사용자 별 행동 모델을 병합하는 방안을 제안하였다.

4-3. 이상 징후 검출

이전 장에 기술된 사용행동모델 자동생성 및 동일한 의도를 가지는 행동모델간의 병합 작업은 그림 1 의 플랫폼 상에 정의된 Behavior model generator(그림 1 의 (d)) 에 의해 수행된다. 사용자 로그를 입력으로 받아 병합된 몇 가지의 사용행동모델과 미리 정의된 예측행동모델을 Bad symptom analyzer(그림 1 의 (e))에게 인자로 보내어 두 모델간의 비교를 요청한다. 그림 1 의 (e) 컴포넌트에서 행동 모델을 비교한 결과 어떠한 이상 징후가 나타냈는지 식별한다.

그림 4 의 상단에 보이는 설계자의 예측행동모델은 사용자의 어플리케이션 사용 행위가 {①→②→③} 의 상태 전이 흐름을 지닐 것으로 예상하고 있음을 뜻한다. 그러나 실제 각 사용자들의 모바일 사용 로그로부터 생성된 사용행동모델을 병합하여 구성한 사용행동모델은 예측행동모델과 달리 전체 사용자의 44%가 {①→⑤→③} 혹은 12%의 사용자가 {①→⑧→③} 의 상태흐름으로 어플리케이션을 사용하고 있음을 알 수 있다. 예상 행동모델과 비교했을 때 실제 사용자 행동 모델의 상태흐름은 56%의 사용자들이 개발자의 설계의도와는 다른 형태의 행위를 취하고 있음을 의미하며, 해당 어플리케이션의 사용에 있어서 불필요한 뷰 간의 이동을 수행하고 같은 기능을 수행할 때 5423ms 로 더 많은 시간을 소요하고 있다. 즉, 사용성 저해 요소가 존재함을 의미한다. 이러한 '예상하지 못한 사용자 행동' 의 이상 징후는 실제 사용자의 행동 모델과 예상행동모델간의 비교를 통해 개선 가능하다. 설계자의 의도대로 어플리케이션을 수행한 결과 얻어지는 행동모델과 실제 사용자

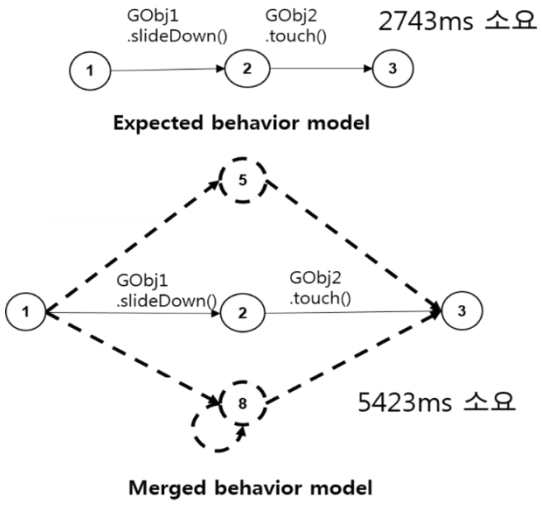


그림 4 사용성 저해요소 검출 예

의 로그로부터 생성된 사용자 행동 모델과 차이점이 감지될 경우, 사용성 저해 요소가 있다고 판별한

다. 설계자가 예상행동모델과 차이점이 존재한다고만 해서 모두 저해요소로 식별되지는 않는다. 현재의 적응 목적은 대다수의 사용자의 사용성 정도를 최대화하는 데 있으므로, 설계자의 예측행동모델과 다른 양상을 보이는 행동모델이 차지하는 백분율에 임계값을 적용하여, 적정 임계값 이상의 실제 사용자 행동모델이 예측행동모델과 상이한 경우, 저해요소로 식별된다. 동일한 기능을 수행하기 위하여 복수의 사용자 경험(User Experience)이 존재하는 경우, 사용자에게 불편함을 주거나 기능을 수행하는데 소요되는 시간을 불필요하게 연장시키기도 한다. 사용자들의 행동 모델 중 설계행동모델과 달리 다수의 사용자로부터 관할된 행위모델 상에 존재하는 이상 징후를 식별하여 자가 적응을 수행함으로써 어플리케이션의 사용성을 개선하는 의사 코드(pseudo code)는 표 1 과 같다.

저해요소를 분석하기 위해 병합된 사용자 행동 모델과 예상 행동 모델과의 차이 나는 부분 AFSM 집합을 추출한다(Analyze 의 1-2 번 줄). 추출된 AFSM 에 포함된 모든 상태간 전이 중 예상되는 GUI 에 예상된 터치 제스처를 취한 상태간 전이 t 를 저장한다(Analyze 의 2-3 번 줄). 두 AFSM 에서 추출한 차이나는 부분집합에서 아래와 같은 저해요소를 분석한다.

표 1 저해요소 검출 알고리즘 의사 코드

```
* Designed Behavior Model(DBM), User Behavior Model(UBM)

Analyze (AFSM dbm, AFSM ubm)
1: badSymptoms = []
2: Transition diff = GetDiffrence(dbm, ubm)
3: count = countSameTransaction(t , diff)
4: if wrong_pos_GUI(dbm, ubm)
5:       badSymptoms.add(WRONG_POS_GUI)
6: else if unexpected_gesture(dbm, ubm)
7:       badSymptoms.add(UNEXPECTED_GEST)
8: else if repeat_gesture(diff)
9:       badSymptoms.add(REPEAT_GEST)
10: return badSymptoms

GetDiffrence(AFSM src, AFSM dst)
1:   if src.size > dst.size return src.removeAll(dst)
2:   else return dst.removeAll(src)

GetRatio(Transition t)
1:   return t / wholeTransition.length()
```

5. 결 론

모바일 앱 제작에 활용 가능한 라이브러리와 도구들의 지원으로 인해 앱의 기능적, 기술적 요소는 감소되는 반면, 상대적으로 앱의 사용성을 향상시키는 데에는 여전히 많은 시간이 할애되고 있다. 사용성 향상을 위해서는 현재 모바일 앱이 제공하는 사용성 품질을 검증하는 일이 선행되어야 한다. 본 논문에서는 특정 모바일 앱의 GUI 설계상에 내재된 사용성 측면의 문제점들을 자동으로 검출해 내는 도구를 제안하고 있다. 제안된 도구는 사용자의 행동 로그 정보로부터 사용자별 행위 모델을 생성한 후, 이를 모바일 앱의 GUI 설계자의 의도를 반영하는 예상행동모델과 비교하여 둘 간의 상이한 점을 식별하는 방식으로 GUI 설계상의 사용성 문제를 검출해 낸다. 이와 같은 도구를 통해 개발자의 주관적인 판단이 아닌 미리 정의된 사용성 저해요소 타입별 문제점들을 자동으로 검출 할 수 있다. 따라서, 결과적으로 사용성 품질을 만족하기 위해 소모되는 비용을 감소시킬 수 있을 것이라 기대할 수 있다. 본 논문에서 제안된 도구에서 검출된 오류들 중에서 해결방안이 규칙화 될 수 있는 오류에 대해서는 모바일 어플리케이션 스스로가 자가 적응 방식을 통해 해결해 나가는 방안을 연구해 나갈 계획이다.

참고문헌

[1] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein. “RERAN: timing- and touch-sensitive record and replay for Android,” In Proceedings of the 2013 International Conference on Software Engineering, pp.72-81, 2013.

[2] Y. Lin; J.F. Rojas, E.T. Chu, and Y. Lai, "On the Accuracy, Efficiency, and Reusability of Automated Test Oracles for Android Devices," in Software Engineering, IEEE Transactions on, vol.40, no.10, pp.957-970, 2014.

[3] M. Zen. “Metric-based evaluation of graphical user interfaces: model, method, and software support”. In Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems, pp. 183-186, 2013

[4] S.Abrahao, and E. Insfran, "Early Usability Evaluation in Model Driven Architecture Environments," Quality Software, International Conference on, pp. 287-294, 2006.

[5] W.A. Munson, and M. Gardner, "Standardizing Auditory Tests," Acoustical Society of America 22

[6] 마경욱, 박수진, GUI 자가 적응을 위한 모바일 어플리케이션의 사용성 저해요소 자동 검출 기법, 2015 한국 소프트웨어공학 학술대회 논문집 제 17 권 제 1 호, pp.411-412, 2015.

시간-주파수 논리를 활용한 클래식 형식 분석

박수정, 권혁주, 권기현

경기대학교 컴퓨터과학과
경기도 수원시 영통구 광교산로 154-42
{amy900, hyuk7410, khkwon}@kgu.ac.kr

요약: 소리는 주파수를 가지는 신호이다. 소리를 정확하게 인지하기 위해서는 분명하고 기계처리가 쉬운 정형 명세가 필요하다. 본 논문에서는 시간-주파수 논리를 활용한 클래식의 형식 분석 명세를 제안한다. 푸리에 변환은 주파수에 대한 정보만 가질 뿐 시간에 대한 정보가 없으므로 시간과 주파수 정보를 모두 고려하는 시간-주파수 논리를 사용하였다. 본 논문에서는 클래식 중에서 엄격한 형식을 따르는 곡을 선정하여 명세를 통하여 형식을 분석하였다. 제안된 방식을 이용하여 형식 분석뿐만 아니라 여러 분야로 활용될 수 있을 것이다.

핵심어: 클래식 형식, 주파수, 푸리에 변환, STFT, TFL, Breach

1. 연구배경

우리 주변에는 다양한 소리들이 있다. 사람의 귀로 인지하기 쉬운 소리도 있지만, 인지하기 어렵거나 심지어 안들리는 미세한 소리도 있다. 예를 들어, 자동차 내부의 미세한 소리, 음악의 연속적인 음들이 있다. 자동차 내부의 미세한 소리는 주변의 소음 등으로 인해 사람들이 잘 인식하지 못하고, 음악의 연속적인 음들은 전문가가 아닌 일반 사람들은 해당 곡의 정확한 음계들을 구분하지 못한다. 이런 소리들을 인지하기 위해서는 소리가 가진 주파수를 정형 명세한 소프트웨어가 필요하다. 특히 안전과 관련된 분야에서 정형 명세를 이용하면 소프트웨어 내부적으로 소리를 정확하게 인지할 수 있을 것이다.

따라서 본 논문에서는 소리 중 우리 주변에서 많이 접할 수 있는 클래식을 예로 들어 정형 명세를 통하여 형식을 분석하고자 한다. 음악은 주파수를 가지는 신호이기 때문에 주파수를 이용하여 정형 명세를 하면 정확하게 음 탐지가 가능하게 된다. 이를 통하여 향후 음 탐지가 필요한 다양한 산업으로 발전에 도움을 주고자 한다.

"본 연구는 경기도의 경기도 지역 협력 연구 센터 사업(GRRC)의 일환으로 수행하였음." [2015-B01, 소셜 서비스 융합 플랫폼 요소기술 개발 및 산업화]

본 논문에서는 관련 연구에 대해서 푸리에 변환, STFT, 시간-주파수 논리를 설명하고, 엄격한 형식을 가지는 클래식 음악 3 개를 시간-주파수 논리를 이용한 명세를 통해 어떤 형식을 가지는 지 분석하였다. 아래 그림 1 은 변주곡 형식을 가지는 곡의 악보이다.

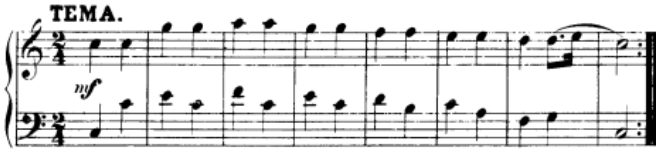


Figure 1. 모차르트의 작은별 변주곡 악보 일부

2. 기본 개념

2.1 푸리에 변환

푸리에 변환(Fourier Transform, FT)은 시간 영역의 신호를 주파수 영역으로 변환하는 것을 말한다. 주파수는 다양한 사인(Sine)곡선과 코사인(Cosine)곡선의 가중합으로 표현되어 있다. 푸리에 변환은 그 주파수 신호를 정현파(sinusoidal)들의 합으로 분해하는 것이다. 음악을 예로 들면 어떤 화음을 그 화음을 구성하는 음들의 진폭에 따라 표현하는 것과 같다. 이 외에 자세한 내용은 [1]을 참조한다.

2.2 STFT

2.1 의 푸리에 변환은 주파수 영역만 고려하므로 시간에 대한 정보를 보여주지 않는다. 따라서 시간에 따른 주파수 변화 신호에 적합하지 않은 방법이다. 이런 단점을 보완하고자 나온 것이 Short-Time Fourier Transform(STFT)이다. STFT 는 분석하고자 하는 신호에 Window 함수를 적용하여 일정 구간에 대해 FT 를 실행하는 방법으로 신호의 시간, 주파수의 정보를 동시에 분석할 수 있다. [2] STFT 식(1)은 다음과 같다. 이 외에 자세한 내용은 [1]을 참조한다.

$$\hat{x}_L(\omega,\tau) = \int_{-\infty}^{+\infty} x[t] g_L(t-\tau) e^{-2i\pi\omega t} dt \qquad (1)$$

2.3 Time-Frequency Logic

Time-Frequency Logic(TFL)은 주파수 도메인 속성과 시간 도메인 속성에 시제 논리 속성을 결합한 신호의 새로운 공식이다.즉, TFL 은 신호 시제 논리(Signal Temporal Logic, STL)에 시간과 주파수를 고려한 신호 연산자를 더하면서 얻어지는 공식을 말한다. STL 은 연속적인 시간 개념을 가진 신호를 다루는 논리이다. STL 의 더 자세한 내용은 [1]을 참조한다. TFL 을 음악에 적용하여 음 탐지를 할 수 있고 더 나아가 멜로디를 인지할 수 있다. 이 외에 자세한 내용은 [1]을 참조한다.

3. 본 론

3.1 분석 대상

본 논문에서는 클래식의 여러 형식 중 변주곡과 캐논을 분석 대상으로 하였다. 변주곡은 한 주제를 박자, 빠르기, 음계를 바꾸어 조금씩 다르게 연주하는 형식이고, 캐논은 같은 선율을 일정한 간격으로 반복해 연주하는 돌림노래 형식이며, 론도는 어떤 하나의 주제가 다른 주제를 사이에 두고 여러 번 반복되는 것이다. 언뜻 보기에 비슷해 보이는 형식들이지만 엄연히 다른 종류이므로 세 대상을 분석하고자 한다. 곡은 다음 표 1 과 같이 선정하였다.

Table 1. 분석에 사용된 곡 목록

형식	작곡가	제목
변주곡	모차르트	작은별 변주곡 C Major
캐논	파헬벨	Canon in D Major
론도	베토벤	엘리제를 위하여

3.2 분석 방법

본 논문에서는 변주곡, 캐논, 론도 순서로 분석한다. 사용한 툴은 Matlab 의 Breach Tool 이다. FT 는 주파수 영역의 성분만 고려하므로 음악과 같이 시간에 따라 변하는 주파수를 효율적으로 분석하기 어렵다. 따라서 STFT 를 사용하여 시간에 따른 주파수 변화를 분석하였다. 음높이(pitch)는 [1] 논문에서 정의한 공식을 사용하였다. 계이름 "라"를 예로 들면, (2)의 식이 나온다.

$$pitch\ \ \omega_A = 440Hz,\ \mu_A = pitch_{\omega_A}(x) > \theta \qquad (2)$$

3.2.1. 변주곡

작은별 변주곡은 그림 1 과 같이 C G A G F E D C 로 8 마디의 기본 코드로 구성되어 있다. 변주곡 형식이므로 약간씩 변화는 있지만 기본 코드는 유지한다. 파일은 미리 노이즈가 제거된 wav 파일이고, 분석 구간은 기본코드 한 구간과 변화된 코드 한 구간이다. 이를 바탕으로 속성 φ_{Star} (3)을 TFL 문법으로 명세하였다.

$$\varphi_{Star} = \mu_C \wedge \Diamond_{[b,2b]} \mu_G \wedge \Diamond_{[2b,3b]} \mu_A \wedge \Diamond_{[3b,4b]} \mu_G \wedge \Diamond_{[4b,5b]} \mu_F \wedge \Diamond_{[5b,6b]} \mu_E \wedge \Diamond_{[6b,7b]} \mu_D \wedge \Diamond_{[7b,8b]} \mu_C \qquad (3)$$

3.2.2. 캐논

캐논은 바이올린 1, 바이올린 2 의 두 성부가 동일한 선율을 시간 간격을 두고 연주한다. 분석 구간은 한 선율을 바이올린 1, 2 가 연주한 구간이다. 해당 파일은 13 초 간격을 가지고 동일한 선율이 나온다. 이를 바탕으로 속성 φ_{canon1} (4)을 TFL 문법으로 명세하였다. 그러나 eventually 는 한 번이라도 참이면 참인 연산자이므로 횟수는 알 수 없다. 이 부분에서는 두 번 나오는지도 증명해야하기 때문에 속성 φ_{canon2} (5)를 따로 명세하였다.

$$\varphi_{canon1} = \Diamond_{[0,13]} (\mu_{FS}\ until\ \mu_E\ until\ \mu_D\ until\ \mu_{CS}\ until\ \mu_B\ until\ \mu_A\ until\ \mu_B\ until\ \mu_{CS}) \qquad (4)$$

$$\varphi_{canon2} = \mu_{FS}\ until\ \mu_E\ until\ \mu_D\ until\ \mu_{CS}\ until\ \mu_B\ until\ \mu_A\ until\ \mu_B\ until\ \mu_{CS} \qquad (5)$$

3.2.3. 론도

엘리제를 위하여는 메인 주제 A 와 부주제 B, C 가 A B A C A 형식으로 되풀이 되는 작은 론도 형식이다. 즉, 전체 구간에서 메인 주제인 A 가 세 번 나와야한다. 이 부분에서는 횟수가 중요한 형식이기 때문에 until 을 사용하였다. 이를 바탕으로 속성 φ_{elise} (6)을 TFL 문법으로 명세하였다.

$$\varphi_{elise} = \mu_E\ until\ \mu_{DS}\ until\ \mu_E\ until\ \mu_{DS}\ until\ \mu_E\ until\ \mu_B\ until\ \mu_D\ until\ \mu_C\ until\ \mu_A \qquad (6)$$

3.3 분석 결과

아래 그림 2 는 작은별 변주곡을 3.2.1 에서 지정하였던 명세 (3)에 따른 결과 그래프이다. 그림 2 에서 두 구간에 한 번씩 True 가 보였으므로 이 곡은 해당 명세를 만족한다. 따라서 이 곡은 변주곡 형식이다. (x 축은 시간, y 축은 φ_{star} 이다.)

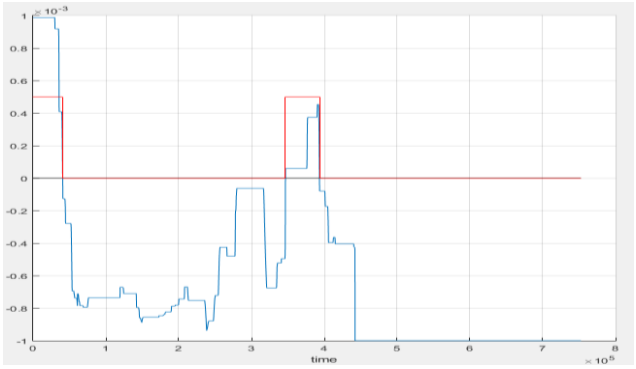


Figure 2 작은별 변주곡 결과 그래프

아래 그림 3 은 캐논을 3.2.2 에서 지정하였던 명세 (4)에 따른 결과 그래프이다. 그림 3 에서 전체 구간 중 최소한 한 번 참이라는 eventually 명세를 만족하였기 때문에 13 초 간격으로 명세한 선율이 있다고 할 수 있다. 그리고 총 몇 번 나왔는지에 대한 명세 (5)의 결과 그래프는 그림 4 와 같다. 그림 4 에서는 선율이 총 2 번 나오는 것을 볼 수 있다. 따라서 이 곡은 캐논 형식이다. (그림 3, 그림 4 모두 x 축은 시간이고, y 축은 φ_{canon1}, φ_{canon2}이다.)

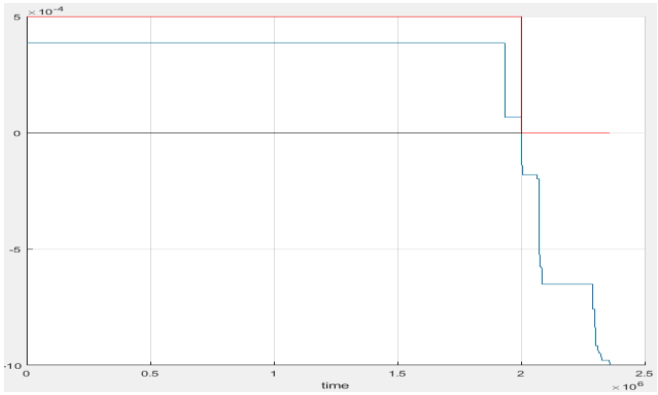


Figure 3 캐논 결과 그래프

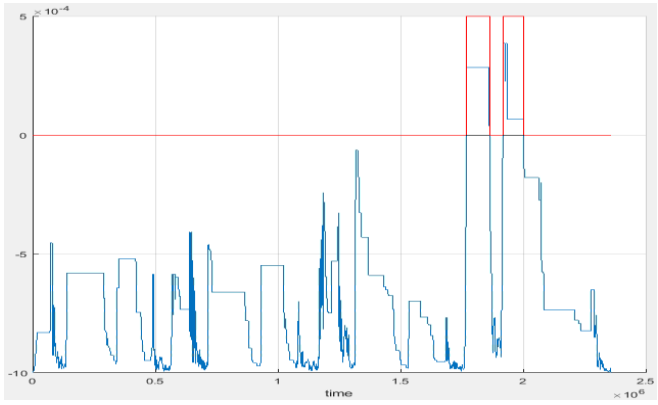


Figure 4 캐논 단순 선율 결과 그래프

아래 그림 4 는 엘리제를 위하여를 3.2.3 에서 지정하였던 명세 (6)에 따른 결과 그래프이다. 그림 4 에서 총 세 번의 True 가 보였으므로 이 곡은 해당 명세를 만족한다. 따라서 이 곡은 론도 형식이다. (x 축은 시간, y 축은 φ_{elise}이다.)

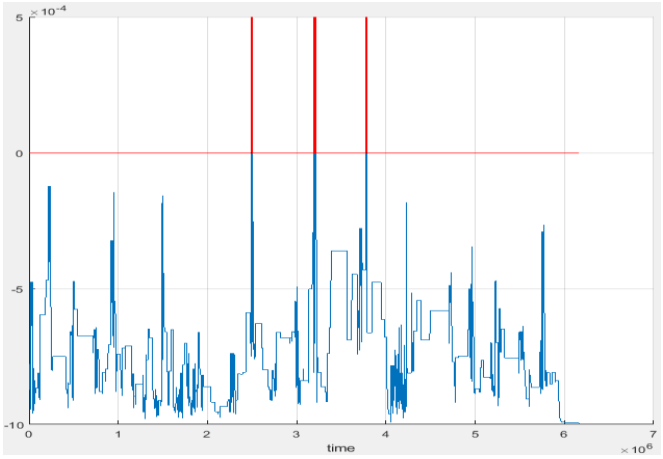


Figure 5 엘리제를 위하여 결과 그래프

4. 결 론

본 논문에서는 주파수를 가지는 다양한 소리 중 우리 주변에서 흔히 접하는 클래식을 예로 들어 TFL 을 이용하여 해당 음악이 어떤 형식을 가지는 지 명세한 뒤 만족도 그래프를 통하여 증명하였다. TFL 을 사용함으로써 클래식의 형식들 중 비슷해 보이는 3 가지 형식을 정형 명세를 하여 정확히 구분하였으므로 소프트웨어 내부적으로 정확한 분석이 가능하게 되었다. 이런 분석을 활용하면 음악 분야에서는 표절시비 해결이 가능하고, 이외에 자동차와 항공기의 내부 엔진, 소리를 분석하는 로봇 등 소리와 관련된 여러 분야에서도 활용이 가능할 것이다.

향후 연구 방향은 TFL 을 이용한 음 탐지 방법을 더욱 발전 시켜서 형식 뿐만 아니라 음 하나 하나를 정확하게 명세할 예정이다.

참고문헌

[1] Alexandre Donze, Oded Maler , Ezio Bartocci, Dejan Nickovic, Radu Grosu and Scott Smolka "On Temporal Logic and Signal Processing", 2012

[2] 황보석, 천선용, 강소영, 이찬수 "음악의 주파수 분석을 이용한 조명 제어", 2013

레거시(PL/SQL) 코드에서 변경 영향을 검출하는 오염 분석기 설계 방법: 요약 해석을 기반으로

김용기
㈜ 삼성전자
서울대학교 컴퓨터 공학부
서울 관악구 관악로 1
yongki82.kim@samsung.com

요약: 오염 분석을 통해 레거시(PL/SQL) 코드로 작성된 삼성 반도체 생산공정 소프트웨어의 변경 영향을 실행 전에 빠트림 없이 감지하는 방법을 제안한다. 이 방법은 요약 해석 프레임워크에 기반하여 엄밀한 수학적 이론을 바탕으로 하면서 구현이 간단하고 성능과 정확도 튜닝도 용이하다. 본 연구에서는 코드 변경 영향을 오염 전파로 정의하고, 대상 언어인 PL/SQL 을 분석에 용이한 CFG 언어로 변환하여 실제 실행 의미와 이를 요약한 요약 실행 의미를 정의하였고, 이를 바탕으로 유한 시간 내에 변경 영향 지점을 빠트림 없이 찾아내는 분석기를 설계하였다. 개념 검증을 위해 설계 명세를 준수하는 간단한 분석기를 구현하여 실험하였으며 6 개 대표 테스트 케이스에 대해 분석기가 변경 영향을 올바르게 찾아냄을 확인할 수 있었다.

핵심어: 변경영향분석, 정적분석, 오염분석, 요약해석, PL/SQL

1. 서 론

레거시(PL/SQL) 코드로 작성된 삼성 반도체 생산공정 소프트웨어는 대규모 분산 시스템의 핵심 로직이 담겨 있어 매우 복잡하고 잘못 변경하면 막대한 피해로 이어진다. 리팩토링 없이 이십여 년간 축적된 코드는 의존 관계가 뒤얽혀있고 모듈화도 빈약하여 고쳤을 때 전체 시스템에 미칠 영향을 정확히 파악하기 어렵다. 반도체 제품은 고가이고 생산량도 엄청나기 때문에 잘못된 변경이 조기에 발견되지 않으면 수십억 이상의 피해를 초래할 수 있다.

잘못된 코드 변경에 의한 영향을 실행 전에 빠짐없이 파악하는 정적 분석 도구가 필요하다. 테스트나 코드 리뷰에서 발견하지 못하는 변경 영향의 피해가 계속 발생하고 있기 때문이다. 중요한 DB 에 발생하는 영향을 빠트림 없이 파악할 수 있는 분석기를 개발하여 코드 리뷰의 베이스라인으로 활용한다면 지금까지 경험과 감에 의존해 놓쳤던 변경 위험을 미리 방지할 수 있을 것이다.

또한 특수한 도메인 언어(PL/SQL)에 적합한 분석 도구 연구가 필요하다. PL/SQL 은 주류 언어가 아니라 시중에 정적 분석 도구가 거의 없고 관련 연구 또한 매우 드물다. 따라서 특수한 상황에도 적용 가능한 일반적이고 실용적인 분석 기술 개발이 필요하다. 요약해석 프레임워크는 특수한 도메인 언어에도 적용 가능한 일반적 설계 방법을 제시할 뿐 아니라 성능과 정확도를 향상시킬 수 있는 실용적인 방법 또한 제공한다.

2. 관련 연구

2.1 변경 영향 분석

변경 영향 분석은 변경에 따른 모든 종류의 잠재적 위험을 식별하는 것으로 지난 20 년간 많은 연구가 이루어져왔다[6]. 변경 영향 분석의 대상은 광범위한데 소스코드, 아키텍처, 요구사항, 기타 아티팩트(문서, 구성 등)들을 포함한다. 이 중 소스코드를 대상으로 한 영향 분석이 가장 큰 비중을 차지한다.

소스코드를 대상으로 하는 영향 분석은 주로 의존 분석이나 실행 과정 추적을 통해 이루어지나 본 연구와 같은 특수 언어의 영향 파악에는 한계가 있다. 의존 분석의 경우 주로 자원이나, 상속 관계, 메소드, 변경과 같은 큰 범위의 의존관계를 추적하기 때문에 [7] 코드로 인한 프로그램 제어 흐름의 변경 영향의 분석은 어렵다. 실행 과정 추적의 경우 역시, 동적 분석에 의존하기 때문에 루프나 대량의 DB 데이터를 처리하기 힘들고, 빠트림 없는(soundness) 분석이 불가능하다.

이 연구는 오염 분석을 통해 단위 코드의 변경 영향을 분석하는 일반적인 설계 방법을 제시한다. 오염 분석을 통한 변경 영향 분석 사례는 거의 찾아볼 수 없으며, PHP 프로그램의 변경 영향을 오염 분석으로 검출한 연구 사례가 있지만 프로그램의 실행 의미를 온전히 다루지 않아 본 연구와 같은 다른 분야의 사례에 적용하기 어렵다[8].

2.2 오염 분석

오염 분석은 보통 동적 오염 분석(Dynamic Taint Analysis)로 대표되며 보안 연구를 위해 광범위하게 사용되는 분석 기술이다[9]. 프로그램을 실행해가며 사용자 입력이나 민감한 API 호출 지점 같은 오염원(taint source)으로부터 영향 받는 지점을 관찰하는 방법으로, 이를 통해 악성코드, 개인정보 유출, 포멧 스트링 버그 등의 보안 취약점을 찾아낸다.

관련 연구와 이 연구의 차이점은 정적 분석을 기반으로 한다는 점이다. 즉, 요약 해석이라는 강력한 정적 분석 프레임워크를 통해 동적 오염 분석으로 정의하기 어려운 PL/SQL 의 프로그램 실행 의미를 안전하게 요약했다는 점이다. PL/SQL 은 SQL 을 내장되어 있고 데이터베이스의 방대한 데이터를 다루기 때문에 실제 실행 실행으로 영향 지점을 파악하는 것이 어렵다.

다른 장점은 요약 해석을 통해 분석기가 오염 분석에 한정되지 않고 일반적으로 확장될 수 있다는 점이다. 동적 오염 분석을 위한 일반적인 프레임워크나[10] 정확도 향상 기법의[11] 연구가 연구되고 있지만 동적 오염 분석에 한정된 기법이기 때문에 다른 분석 프로그램 분석에 적용하기 어렵다. 반면 요약 해석은 버퍼 오버런 분석, 배열 인덱스 분석 등 다양한 종류의 분석으로 범위를 확장할 수 있다.

3. 배 경

3.1 레거시 코드: PL/SQL

PL/SQL 로 정의된 비즈니스 규칙은 생산 시스템의 처리 결과의 무결성을 체크한다. 그림 1 은 반도체 생산 라인 설비에서 제품이 공정을 마쳤을 때의 규칙 일부를 간략화 한 것으로, PL/SQL 코드의 특징을 살펴볼 수 있다. 루프와 복잡한 조건 절로 이루어져 있으며, SQL 을 코드 내에서 사용할 수 있고 사용자 정의 함수 생성도 가능하다.

PL/SQL 프로그램이 갖는 레거시 코드의 문제점은 오랜 시간 리팩토링 없이 축적되어 와서 모듈화가 거의 되어 있지 않고 복잡한 제어 구조와 의존 관계로 얽혀있어 영향 파악이 어렵다는 점이다. 전문가의 경험이나 통합 테스트 환경의 시뮬레이션으로 대응하고 있으나 중요한 변경 영향 지점을 빠트리는 경우가 있고 개발 인력과 시스템 자원도 낭비되고 있다.

하지만 모듈화가 잘 되어 있지 않고 포인터나 메모리 동적 할당이 없는 특징 때문에 함수 인라이닝과 같은 최적화를 통해 프로그램 구조를 간단하게 요약할 수 있어 정적 분석을 하기 쉬운 점은 긍정적이다.

```
1  CREATE PACKAGE BODY pkg1 AS
2    PROCEDURE main IS
3      TYPE eqp IS TABLE OF VARCHAR2(100) INDEX BY INTEGER;
4      i NUMBER; y NUMBER; z NUMBER;
5    BEGIN
6      WHILE i < 100 LOOP
7        -- 복잡한 IF 구분
8        IF i > 50 THEN
9          /* ... */
10         IF x < 10 THEN
11           y := y + 2;
12         END IF;
13       END IF;
14       z := reset_count(eqp(i), y);
15       i := i + 1;
16     END LOOP;
17   END main;
18
19   FUNCTION reset_count(
20     id VARCHAR2(100),
21     y  NUMBER) RETURN NUMBER IS
22   BEGIN
23     IF y > 100 THEN
24       UPDATE critical_db SET limit_count = 0
25       WHERE item_id = id;
26       RETURN 100;
27     END IF;
28     RETURN y;
29   END reset_count;
30 END pkg1;
```

그림 1 PL/SQL 예제 코드

3.2 오염 전파를 통한 변경 영향 정의

변경의 종류에 따라 관련 변수나 문장을 오염원으로 지정하고, 오염이 전파되는 과정을 정의하면 변경 영향을 오염의 전파로 모델링 할 수 있다.

변경 영향 정의 그림 2 와 같이 변수 또는 문장 전체의 오염으로 정의한다. 코드가 수정되거나 추가될 경우 해당 라인에서 정의되는 변수를 오염시킨다. 코드가 삭제된 경우 삭제된 라인에서 정의된 변수를 변경 후 대응되는 라인에 추가한 뒤 오염을 주입한다.

분류	변경 후	변경 전
코드 수정	**x** := 2 y := x + 1	x := 1 y := x + 1
코드 추가	**x** := 1 y := x + 1	 y := x + 1
코드 삭제	x := k **x** // 오염주입 y := x + 1	x := k x := x + 1 y := x + 1

그림 2 단위 코드 변경의 오염 정의

할당문의 변경 영향 전파 규칙 그림 3 과 같이 할당문의 경우 오염값을 사용한 문장과 그 값이 할당된 변수를 오염시킨다. 배열 값에 접근할 때, 인덱스가 오염될 경우 배열 전체에 오염이 전파되어야 한다. (변경에 의해 배열의 어떤 인덱스에 값이 할당될 지 알 수 없기 때문) 마찬가지로 `le.x` 와 같은 레코드 필드 값에 접근할 때도 `le` 가 오염되었을 경우 `le.x` 에도 오염이 전파되어야 한다.

```
rec1.x := 0;
rec1.y$ := rec1.x + 1; -- 오염 강제 주입: rec1.y = (⊤,1)
rec1.z := rec1.y;  -- 오염 전파: rec1.z = (⊤,1)
arr1(0) := 0;      -- 오염 전파.
arr1(rec1.y) := 1; -- arr에 rec1.y인덱스의 오염이 전파
rec1.x := arr1(1); -- arr을 사용하는 rec1.x에도 오염 전파
```

그림 3 배열과 레코드에 대한 오염 전파

조건문의 변경 영향 전파 규칙 조건식이 변경되었을 경우 변경 후 어떤 분기를 진행할 지 알 수 없다. 따라서 그림 4 과 같이 조건문 내의 모든 분기에 정의된 변수값을 오염시킨다.

```
IF x > 0 THEN
  z := y;
  IF z > 0 THEN
    i := 1;
  END IF
ELSE
  z := 1;
END IF;
j := i;
```

그림 4 조건문의 오염 전파

SQL 문의 변경 영향 전파 규칙 SQL 문으로 테이블에서 오염된 값을 할당할 경우 역시 오염이 전파된다. 문제는 `WHERE` 조건절의 경우인데, 조건 계산에 참조되는 컬럼 내의 값이 하나라도 오염되었을 때 테이블의 모든 값을 오염시켜야 한다. 오염된 값이 SQL 문에서 선택되지 않아 제외될 수 있다고 하더라도 "선택 될 수도 있었던" 값을 다른 SQL 문장에서 참조할 수 있기 때문이다.

그림 5 의 경우 테이블의 `c1` 컬럼의 첫 행 값이 변경되었고 `c2` 값은 변경되지 않았지만, `c1` 컬럼을 조건절에서 사용했기 때문에 `c2` 값을 참조하는 `x, y` 의 값도 영향을 받았다.

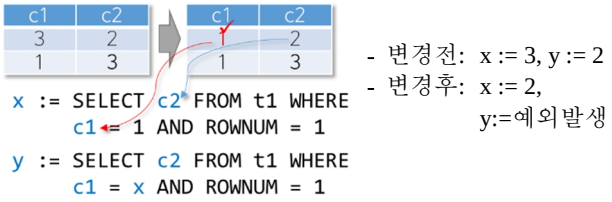


그림 5 SQL 조건 절에 의한 오염 전파

3.3 요약 해석

요약 해석(Abstract Interpretation)은 프로그램의 실제 실행 의미와 이를 포섭하는 요약 실행의미를 함수로 정의하고 요약된 실행 의미의 최소 윗뚜껑(least upper bound)를 유한 시간 내에 계산하는 분석 틀이다[1].

예를 들어, 정수의 덧셈을 짝수와 홀수의 덧셈으로 요약할 수 있다. 조금 더 정교하게 요약하면 그림 **6** 과 같은 범위값의 덧셈으로 요약할 수 있을 것이다. 이와 같은 래티스 도메인에서 두 값이 결합하는 (JOIN) 최소 윗뚜껑이 항상 존재한다. 요약 해석은 실제 실행의미의 도메인을 래티스의 완화 조건인 CPO(Complete Partial Order) 공간으로 요약하고, 요약 공간에서의 요약된 실행 의미가 항상 실제 값을 포섭함을 보장하는 분석 프레임워크이다.

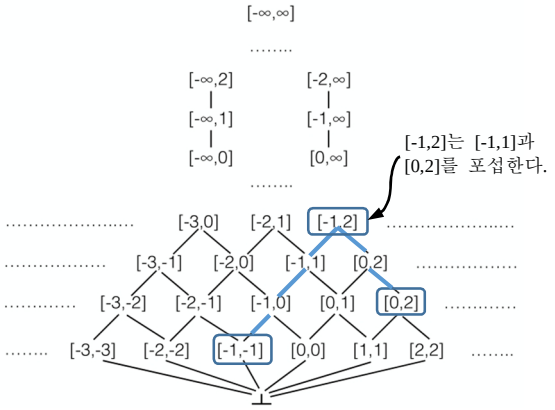


그림 6 정수 범위 도메인의 래티스

요약 해석은 간단하면서 강력한 프로그램 분석 틀(Framework)이다. 생각하고 체크할 사항이 별로 없으면서 거의 모든 프로그램 분석을 이 틀이 제공하는 가이드라인을 따라 만들어 낼 수 있다.

4. 요약 해석 기반 분석기 설계

요약 해석 틀에 따라 변경 영향 분석기를 설계한 과정을 설명한다. PL/SQL 핵심 언어, 실행 의미, 요약 의미의 완전한 정의는 [12]에서 받아볼 수 있다.

4.1 구문 단순화: 중간언어와 CFG

분석기 디자인을 단순화 하기 위해 실제 현장에서 사용하는 문법만을 PL/SQL 핵심 언어로 정의한 뒤 중간언어를 거쳐 CFG(Control Flow Graph)로 단순화한다. 반도체 도메인에 특화된 분석기를 대상으로 하기 때문에 수천 쪽에 달하는 PL/SQL 언어 명세를 모두 고려할 필요가 없다. 중간 언어로 변환하는 이유는 SQL 문법을 할당문으로 바꿔 오염 전파 규칙을 적용하기 쉽게하고, 추후 C#, Java 등 다른 언어로 분석기를 확장할 때 중간언어를 공유하여 변경을 최소화하기 위함이다. 또한, CFG 로 변환을 통해 중간 언어를 극단적으로 단순화시킨다. IF/WHILE/함수 호출/예외처리 같은 제어 흐름을 그래프 상의 노드와 엣지로 녹이고, 함수 호출이나 예외도 인라이닝을 통해 녹인다.

그림 7 은 SQL 실행 결과를 `FOR` 문으로 탐색하며 DB 를 업데이트 하는 PL/SQL 코드를 CFG 로 단순화하는 과정을 보인다. 중간 언어 변환 후 `SELECT` 문에 대한 암시적인 커서 생성과 사용이 `@cursor` 라는 임시 커서 변수에 대한 명시적인 정의와 `while` 문

내에서의 할당으로 바뀌었다. `while` 문은 CFG 변환 후 조건 노드와 엣지로 더 단순하게 녹아진다.

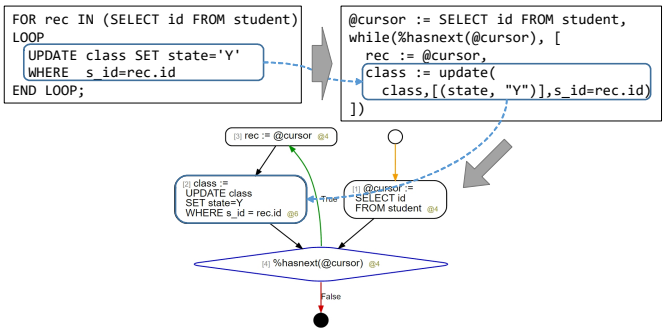


그림 7 PL/SQL 구문 단순화

변환된 CFG 언어의 정의는 그림 8 과 같다. PL/SQL 의 십여 개 이상의 실행문이 할당문 하나로 녹여짐을 확인할 수 있다.

$$
\begin{array}{lcll}
CFG & = & (Node, \rightarrow) & \\
Node & = & N_{cmd} & (statement\ node) \\
 & \uplus & N_{cond}(e) & (condition\ node) \\
 & \uplus & N_{skip} & (skip\ node) \\
 & & & \\
(\rightarrow) & = & Node \times Node & (normal\ edge) \\
(\overset{b}{\hookrightarrow}) & & Node \times Node & (cond.edge) \\
 & & (b \in \{true, false\}) & \\
(\overset{x}{\rightsquigarrow}) & & Node \times Node & (handleredge) \\
 & & & \\
cmd & \rightarrow & le := e & \\
le & \rightarrow & x \mid le.x \mid le[e] & \\
e & \rightarrow & le \mid const \mid le(e^*) \mid e\ bop\ e \mid uop\ e \mid cop\ e \mid sql &
\end{array}
$$

그림 8 CFG 언어 정의

4.2 실제 실행 의미 정의

그림 9 와 같이 프로그램 실행 중 변화하는 모든 상태들의 족적(Trace)으로 프로그램의 실제 실행 의미를 정의한다. 상태는 CFG 의 노드마다 매달린 메모리와 오염맵($Taint$)으로 정의된다. 메모리는 식별자($Loc = Id$)가 가리키는 값이며 값의 종류는 스칼라값, 테이블, 커서가 있다. 오염맵은 어떤 노드가 오염되었는지 여부를 가리킨다.

그림 10 은 실제 실행 의미 함수($next$)의 전이 규칙이다. 분자/분모에서 분자는 조건을 분모는 상태변화를 나타낸다. 예를 들어 첫번째 규칙은 할당문의 오염 전파 규칙으로 값의 오염 정보와 주소의 오염정보, 그리고 이 노드를 생성한 조건 노드의 오염 정보 중 하나라도 오염되었을 경우($b_1 \vee b_2 \vee b_r$) 오염을 전파한다. $\langle n : le := e, m, T\rangle$는 현재 노드가 할당문임을 나타내며, $\mathcal{V}(m,e) = \langle v, b_1\rangle$는 현재 메모리 m 에서 표현식 e 를 계산한 결과가 $\langle v, b_1\rangle$ 임을 나타낸다. $\mathcal{L}(m, le)$은 L-value 의 주소를 계산한 값이고 $T(R(n))$은 현재 노드가 참조하는 조건식의 오염 여부를 나타낸다.

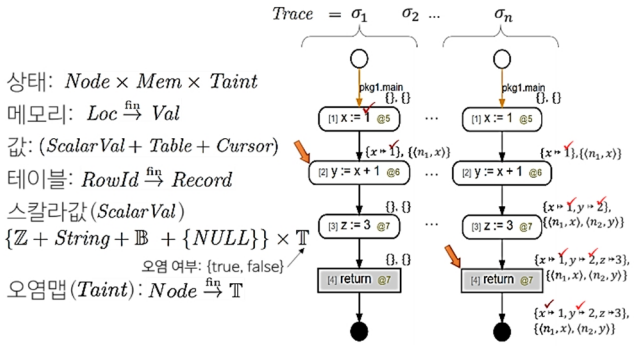


그림 9 실제 도메인 정의

$$
\boxed{next : \langle n, m, T\rangle \rightarrow \langle n', m', T'\rangle}
$$

$$
\frac{\mathcal{V}(m,e) = \langle v, b_1\rangle \quad \mathcal{L}(m, le) = \langle x, b_2\rangle \quad T(R(n)) = b_r \quad n \rightarrow n'}{\langle n : le := e, m, T\rangle \rightarrow \langle n', m\{x \mapsto (v, b_1)\}, T\{n \mapsto b_1 \vee b_2 \vee b_r\}\rangle}
$$

$$
\frac{\mathcal{S}(m, e_{sql}) = v_t \quad \mathcal{L}(m, le) = \langle x, b\rangle \quad T(R(n)) = b_r \quad n \rightarrow n'}{\langle n : le := e_{sql}, m, T\rangle \rightarrow \langle n', m\{x \mapsto v_t\}, m, T\{b \vee b_r \mapsto\rangle\}}
$$

$$
\frac{\mathcal{V}(m,e) = \langle v_c, b\rangle \quad T(R(n)) = b_r \quad n \overset{v_c}{\rightsquigarrow} n'}{\langle n : N_{cond}(e), m, T\rangle \rightarrow \langle n', m, T\{n \mapsto b \vee b_r\}\rangle}\ (\cdot \text{ 는 더미 주소})
$$

$$
\frac{\mathcal{S}(m, e_{sql}) = v_t \quad tainted(v_t) = b \quad n \rightarrow n'}{\langle n : x := e_{sql}, m, T\rangle \rightarrow \langle n', m\{x \mapsto \langle v_t, 1\rangle\}, T\{n \mapsto b\}\rangle}
$$

$$
\frac{m(x_c) = \langle v_t, k\rangle \quad v_t(k)(col_i) = \langle v_i, b_i\rangle \quad T' = T\{n \mapsto b_i\} \quad n \rightarrow n'}{\langle n : x_r := x_c, m, T\rangle \rightarrow \langle n', m\{(x_r.col_i \mapsto \langle v_i, b_i\rangle,\ x_c \mapsto \langle v_t, k+1\rangle\}, T'\rangle}
$$

$$
\frac{\mathcal{V}(m, e_2) = \langle 0, b\rangle \quad n \overset{\text{OTHERS}}{\hookrightarrow} n'}{\langle n : le := e_1/e_2, m, T\rangle \Rightarrow \langle n', m, T\rangle}
$$

$$
\frac{\mathcal{V}(m,e) = \langle v, b\rangle \quad v = \langle v_t, i\rangle \in Cursor \quad i > |\mathrm{dom}(v_t)| \quad n \overset{\text{NOT\_FOUND}}{\hookrightarrow} n'}{\langle n : le := e, m, T\rangle \Rightarrow \langle n', m, T\rangle}
$$

$$
\frac{n \rightarrow n'}{\langle N_{skip}, m, T\rangle \rightarrow \langle n', m, T\rangle}
$$

그림 10 실제 실행 의미 함수의 상태 전이 규칙

요약 해석에서 프로그램의 실제 실행은 CPO(완화된 래티스) 의미공간 D에서 정의된 함수 $F: D \rightarrow D$의 고정점으로 정의된다. 그런데 앞에서 정의한 $Trace$는 단순 시퀀스로 CPO 를 만족하지 않는다. CPO 를 만들기 위해 멱집합 2^{Trace}로 모듬 의미를 구성한다. 프로그램 실행은 반드시 2^{Trace}의 부분 집합에서 맴돈다.

4.3 요약 실행 의미 정의

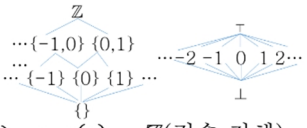
그림 11 실제 도메인과 갈로아 연결[1]을 이루는(요약 결과가 실제를 포섭) 요약 도메인 정의의 일부와

요약 방법을 설명한 것이다. 실제 도메인의 요약 함수를 α라 하고 요약 도메인을 다시 실제 도메인을 구체화 하는 함수를 γ라 한다. 정수의 경우 실제 도메인인 멱집합을 올려붙인 도메인으로 요약한다. 예를 들어 실제 실행으로 가능한 값들이 원소가 한 개 한 집합($\{n\}$)이라면 요약 도메인 값 n으로 요약되고, 가능한 값이 여러 개라면 모든 가능성을 포섭하는 의미인 $\top$으로 요약된다. 테이블은 전체 행을 하나로 뭉쳐 컬럼별로 요약한다.

$$
\begin{array}{lcrll}
\hat{v} & \in & \hat{Val} & = & (\hat{ScalarVal} \times \hat{Table} \times \hat{Cursor}) \\
\hat{v}_s & \in & \hat{ScalarVal} & = & \hat{\mathbb{T}} \times \hat{\mathbb{Z}} \\
 & & \hat{\mathbb{T}} & = & \{\bot, \top\} \\
\hat{z} & \in & \hat{\mathbb{Z}} & = & \mathbb{Z}_\bot \cup \{\top\} \qquad (\text{올려붙인 } \mathbb{Z}) \\
\hat{v}_t & \in & \hat{Table} & = & \hat{ColId} \xrightarrow{\text{fin}} \hat{ScalarVal} (\text{모든 행 뭉침})
\end{array}
$$

정수

- $2^{\mathbb{Z}} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \{\bot, \mathbb{Z}, \top\}$



- 예: $\alpha(\{1\}) = 1, \alpha(\{1,2\}) = \top$, $\gamma(\top) = \mathbb{Z}$(정수 전체)

테이블

- 전체 행을 하나로 뭉쳐 컬럼별로 요약

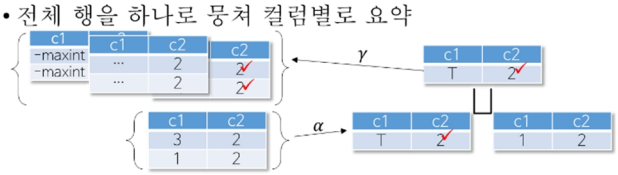


그림 11 요약 도메인 정의

요약 실행 의미는 실제 실행 상태 변화의 족적을 파티션($\Delta = Node$)별로 요약한 것이다. 그림 12 는 할당문 노드의 요약 실행 상태 전이 규칙으로 그림 10 의 실제 상태 전이 함수($\hat{next}$)의 규칙과 매우 유사함을 알 수 있다. 추가된 부분은 $n \overset{*}{\hookrightarrow} N_h$ 조건인데, 실제 실행과 달리 요약 실행에서는 현재 노드와 연결된 모든 예외 노드들(N_h)에도 상태를 전이시킨다. 실제 실행의 경우 한 번에 한 노드만 상태 전이가 가능하다는 것과 대조되는 점이다.

$$
\frac{\hat{\mathcal{V}}(\hat{m}, e) = \langle \hat{b}_1, \hat{z}, \hat{t} \rangle \qquad \hat{T}(R(n)) = \hat{b}_r \qquad n \to n_1 \\ \hat{\mathcal{L}}(\hat{m}, le) = \langle x, \hat{b}_2 \rangle \qquad \hat{b}' = \hat{b}_1 \sqcup \hat{b}_2 \sqcup \hat{b}_r \qquad n \overset{*}{\hookrightarrow} N_h}{\langle n : le := e, \hat{m}, \hat{T} \rangle \hat{\to} \{\langle n', \hat{m}\{x \mapsto \langle \hat{b}', \hat{z}, \hat{t} \rangle\}, \hat{T}\{n \mapsto \hat{b}'\} \rangle \mid n' \in (\{n_1\} \cup N_h)\}}
$$

그림 12 할당문 노드의 요약 상태 전이 규칙

프로그램의 요약 실행은 실제 실행과 마찬가지로 요약 의미공간 $\hat{D}$에서 정의된 함수 $F: \hat{D} \to \hat{D}$의 고정점으로 정의된다. 요약 실행은 모든 조건분기와 예외 핸들러에 상태를 전이하기 때문에 필연적으로 여러 값이 뭉치게 된다. 하지만 도메인이 CPO 이기 때문에 여러 값을 포섭하는 최소 윗뚜껑으로 결합(JOIN)할 수 있고, 이 요약값은 실제 값을 포섭한다.

본 연구에 사용된 요약 도메인은 높이가 유한하기 때문에 유한 시간 내에 분석이 끝난다. 그림 6 과 같이 높이가 무한한 도메인의 경우는 넓히기(Widening)와 좁히기(Narrowing)을 통해 유한 시간에 종료하도록 만들 수 있다.

5. 구현 및 사례 연구

5.1 구현

지금까지 설계한 요약 도메인과 요약 실행 함수를 구현하고, 워크리스트 알고리즘으로 고정점을 계산하면 요약 해석기가 완성된다. 요약 해석기는 Scala 나 OCaml 같은 함수형 언어를 이용하면 쉽게 구현할 수 있으며[12] 공개된 요약 해석기 엔진을 활용할 수도 있다[13].

워크리스트 알고리즘은 이전 반복에서 변화된 노드들을 할 일로 모으고, 그 노드들에 반복적으로 요약 상태 함수 $\hat{next}$ 를 실행한다. 더 이상 할 일이 없으면 분석이 완료된다. 그림 13 은 워크리스트 알고리즘과 Scala 구현 예제이다.

CFG 에서 요약 실행이 올바르게 진행되기 위해서는 강연결요소(Strongly Connection Components)로 싸이클을 묶고 역위상정렬(reverse topological sort)를 통해 가장 말단의 노드가 낮은 우선순위(높은 노드 번호)를 갖도록 번호를 붙여야 한다. 워크리스트 알고리즘은 정렬된 CFG 에 대해 항상 올바른 실행 결과를 보장하지만 최적 실행 순서를 위해서는 워크리스트 실행 순서의 튜닝이 필요하다.

$T, T' : \Delta \to \hat{State}$;
$W : 2^{\Delta}$; (* worklist *)
begin
  $T := T' := \alpha(T_0)$; $W := \Delta$;
  repeat
    $T' := T$;
    $T := \alpha(T_0) \sqcup ((\wp\sqcup) \circ \hat{\pi})(\bigcup_{i \in W} \hat{next}\, T[i])$;
    $W := \{i \in \Delta \mid T[i] \not\sqsubseteq T'[i]\}$;
  until $W = \{\}$; (* no more increase *)
return T';
end

```
1  def worklist(trace: HTrace.elt, todos: List[Node], remains: List[
       Node] = Nil)
2    : HTrace.elt = { // Trace를 받아 Trace를 리턴
3    // 실제 도메인과 요약도메인의 파티션이 같으므로 (℘⊔) ∘ π̂ 는 무시
4    // T = trace, W = todos
5    todos match {
6      case Nil => trace       // no more works
7      case n :: todos1 =>      // n = i
8        val state = HTrace.image(trace, n)            // state = T[i]
9        val (state1, nexts) = hnext(i, trace, state) // next(n, state)
10       val trace1 = nexts.foldLeft(trace) { (trace, n) =>
11           HTrace.weakupdate(trace, i, state1)
12       } // ⋃_{i∈W} next(n, state)
13       val (todos2, remains1) =
14         nextWorks(trace, state1, nexts, todos1, remains)
15       worklist(trace1, todos2, remains1)
16   }
17 }
```

그림 13 워크리스트 알고리즘과 그 구현

5.2 사례 연구

개념 검증을 위해 대표 테스트 항목 6 개에 대해 20 개 이상의 테스트를 만들어 수행 했으며, 이를 통해 변경 영향 분석기가 올바로 구현되었음을 확인했다. 대표 테스트 항목 6 가지는 다음과 같다.

- IF 문의 조건식 오염에 의한 오염 전파 테스트
- 루프에서 오염된 변수에 의한 영향 전파 테스트
- 배열 인덱스 오염에 의한 오염 전파 테스트
- SQL WHERE 조건식에 의한 오염 전파 테스트
- 함수 호출에 의한 오염 전파 테스트
- 예외 처리에 의한 오염 전파 테스트

다음 두 사례를 통해 변경 영향 분석기의 동작 원리와 영향 분석 결과를 보인다.

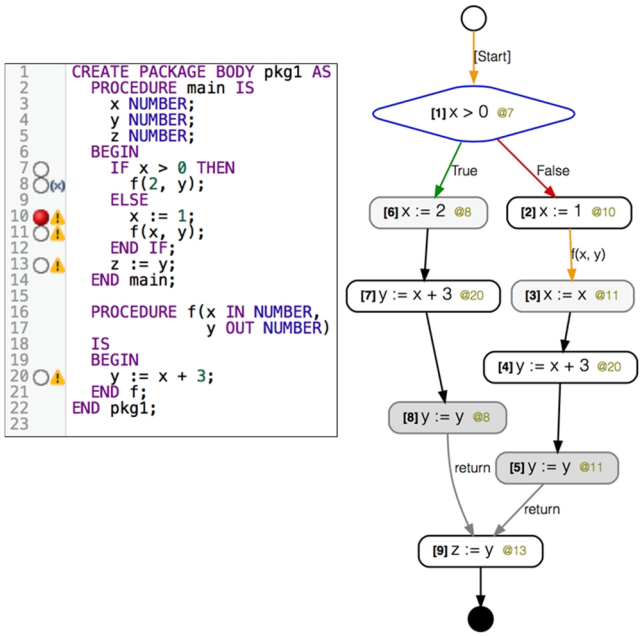


그림 14 분기문과 프로시저가 결합된 예제 코드

분기문의 요약 실행 그림 14 는 IF 분기 내의 할당문 하나가 변경된 코드이다. 그림 15 는 이 코드의 변경 영향을 워크리스트 알고리즘으로 실행한 단계별 상태를 나타낸다. 실행 과정은 단순하다. 현재 노드와 연결된 다음 노드들을 워크리스트에 올려 놓고 할당문 실행하여 현재 노드 상태를 변경하여 다음 노드의 상태로 전달한다. 만일 워크리스트에 노드가 여러 개 있을 경우 경우 낮은 노드 번호를 우선 진행한다. 최종적으로 양쪽 분기의 실행 결과가 만나는 지점(노드[9])에서 두 상태값이 최소 밑뚜껑 값으로 결합(JOIN)하고, Exit 노드를 만나 프로그램이 종료한다.

공정 제어 로직의 변경 영향 분석 그림 16 은 반도체 설비에서 공정이 끝났을 때 실행되는 규칙 일부를 간략하게 예제로 만든 것이다. 11 번째 라인에서 변수 할당문이 변경되어 변수 y 가 오염되었고, 함수 `reset_count` 의 인자로 이 오염이 전파되어 `critical_db` 라는 중요 테이블에 영향이 전달되었음을 올바르게 찾아냈다.

현재 노드	할 일 (워크리스트)	상태 변화 -상태: 노드 → (메모리, 노드오염여부) -메모리: 식별자 → (오염여부, 요약정수)
[1]x>0	[2]x:=1, [5]x:=2	[1] → (⊥,⊥)
[2]x:=1	[3]x:=x, [6]x:=2	[2] → ({x3:(⊥,1)},⊥)
[3]x:=x	[4]y:=x+1, [6]x:=2	[3] → ({x3:(T,1), x16:(T,1)}, {[3]:T})
[4]y:=x+3	[5]y:=y, [6]x:=2	[4] → ({x3:(T,1), x16:(T,1), y17:(T,4)}, {[3]:T,[4]:T})
[5]y:=y	[6]x:=2, [9]z:=y	[5] → ({x3:(T,1), x16:(T,1), y4:(T,4), y17:(T,4)},{[3]:T,[4]:T,[5]:T})
[6]x:=2	[7]y:=y+3, [9]z:=y	[6] → ({x16:(⊥,2)},⊥)
[7]y:=y+3	[8]y:=y, [9]z:=y	[7] → ({x16:(⊥,2), y17:(T,5)},⊥)
[8]y:=y	[9]z:=y (← [5]) [9]z:=y (← [8])	[8] → ({x16:(⊥,2), y4:(⊥,5), y17:(⊥,5)},⊥)
[9]z:=y ⊔ [9]z:=y	[Exit]	[11] ({x3:(T,1), x16:(T,1), y4:(T,4), y17:(T,4)}, ⊔ ({x16:(⊥, 2), y4:(⊥,5), y17:(⊥,5)}, ⊥) {[3]:T,[4]:T,[5]:T}) ↓ ({x3:(T,1), x16:(T,T), y4:(T,T)},y17:(T,T)}, {[3]:T,[4]:T,[5]:T,[9]:T})

그림 15 변경 영향이 전파되는 과정

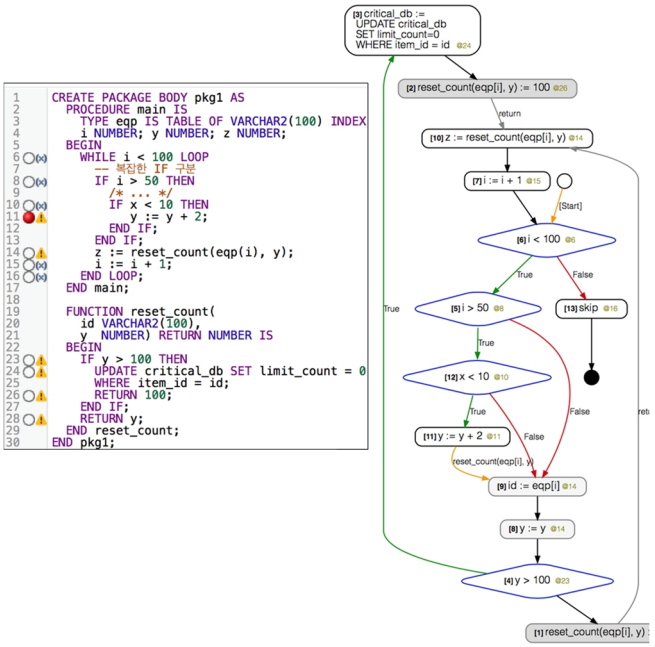


그림 16 공정 제어 로직의 변경 영향 분석 사례

6. 결론 및 향후 과제

요약 해석에 기반한 오염 분석으로 PL/SQL 코드로 작성된 반도체 생산공정 소프트웨어의 변경 영향 지점을 탐지하는 분석기를 설계하고 구현했다. 먼저 프로그램 변경의 의미를 오염 지점의 전파로 정의했으며, 이 오염 전파 규칙을 바탕으로 PL/SQL 의 실제 실행 의미를 정의했고, 요약 해석의 틀로 이를 안전하게 요약하고 구현하는 방법을 제시했다. 또한 사례 분석을 통해 이 분석기가 올바르게 구현 되었음을 보였다.

본 연구는 실제 개발 현장에서 사용할 실용적인 분석기를 개발하는 것을 목표로 시작되었기 때문에 실제 개발자가 사용할 수 있을 정도의 정확도와 성능을 목표로 다음 연구를 계속 진행할 예정이다.

- **정확도 향상:** 본 연구의 분석기는 정적 분석에 기반했기 때문에 과다 근사로 인한 가짜 알람이 많이 발생한다. 이런 문제를 해결을 위해 다양한 연구가 진행되어 왔으며[1][2][3], 선별적 문맥감지는 프레임워크로[4] 정착되었다. 앞으로, 이런 연구 결과를 적용하며 분석기 정확도를 높여나갈 예정이다.
- **성능 향상:** Sparse 프레임워크를 이용해 노드마다 꼭 필요한 메모리를 매달도록 하여 엄청난 성능 향상과 메모리 절감 효과를 얻을 수 있음이 증명되었다[3]. PL/SQL 은 함수 호출이 정적으로 이뤄지기 때문에 Sparse 프레임워크를 적용하기 적합하다. 현재 분석기는 1,000 라인 미만을 5 분 내에 처리하는 수준이며 Sparse 프레임웍 적용을 통해 최대 50,000 라인까지 처리 가능하도록 성능을 향상시킬 예정이다.
- **C# 분석 지원:** 반도체 시스템에서 C#으로 작성된 설비 제어 소프트웨어의 변경 실패로 인한 피해가 이슈가 되고 있다. 본 분석기는 요약해석을 기반으로 하였고, 언어 중립적인 중간 언어와 CFG 를 사용하기 때문에 실행의미와 도메인을 일부 수정하여 C#을 지원하는 분석기로 충분히 확장 가능하다. 이와 관련된 연구도 진행할 예정이다.

참고문헌

[1] Cousot, Patrick, and Radhia Cousot. "Abstract interpretation frameworks." *Journal of logic and computation 2.4* (1992): 511-547.

[2] Cousot, Patrick. "Abstract interpretation based formal methods and future challenges." Informatics. *Springer Berlin Heidelberg*, 2001.

[3] Oh, Hakjoo, et al. "Design and implementation of sparse global analyses for C-like languages." *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. Vol. 47. No. 6, pp. 229-238, ACM, 2012.

[4] Oh, Hakjoo, et al. "Selective context-sensitivity guided by impact pre-analysis." *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. Vol. 49. No. 6, pp.475-484, ACM, 2014.

[5] Cousot, Patrick, and Radhia Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints." *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 1977.

[6] Lehnert, Steffen. "A review of software change impact analysis." *Ilmenau University of Technology, Tech. Rep* (2011).

[7] Ren, Xiaoxia, et al. "Chianti: a tool for change impact analysis of java programs." *ACM Sigplan Notices*. Vol. 39. No. 10. ACM, 2004.

[8] Wang, Xiaoyin, et al. "Automating presentation changes in dynamic web applications via collaborative hybrid analysis." *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012.

[9] Schwartz, Edward J., Thanassis Avgerinos, and David Brumley. "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)." *Security and Privacy (SP), 2010 IEEE Symposium on. IEEE*, 2010

[10] Clause, James, Wanchun Li, and Alessandro Orso. "Dytan: a generic dynamic taint analysis framework." *Proceedings of the 2007 international symposium on Software testing and analysis*. ACM, 2007.

[11] Kang, Min Gyung, et al. "DTA++: Dynamic Taint Analysis with Targeted Control-Flow Propagation." *NDSS*. 2011.

[12] 본 논문에서 정의된 도메인, 실행 의미의 완전한 정의 및 구현 예제 코드 https://github.com/cloudeyes/kcse2016/blob/master/kcse2016_design_detail.pdf

[13] yoyak – generic program analysis engine based on the theory of abstract interpretation, https://github.com/ihji/yoyak

Embedded S/W 개발에서 Issue 해결의 어려움에 대한 고찰

임순일, 정병민, 민상윤
KAIST 대학교 소프트웨어대학원
서울시 강남구 논현로 28 길 25
Sunding.lim@kaist.ac.kr, Jessejung79@kaist.ac.kr, sang@sol-link.com

요약: 본 논문에서는, Embedded Software 개발이 일반 소프트웨어 개발과 어떤 차이점이 있는지 이야기하고, 현재 Embedded Software 개발 조직이 System test 단계에서 겪고 있는 Issue 처리 문제점을 분석한다. 나아가 Embedded Software 특성을 고려해 분석한 문제점들의 해결 방안을 제안한다.

핵심어: Embedded Software, Software 개발 조직, Embedded Software Leader, Issue 처리, It's NOT my issue

1. 연구배경

대체적으로 Software 개발은 사용자가 직접 체감할 수 있는 응용 프로그램 분야에 많이 집중되어 있다. 하지만, 최근에는 Smart Phone 의 개발과 더불어 IoT(Internet Of Things)에 대한 관심이 급증하면서 Embedded 개발에 대한 중요도가 증가하고 있다. 더불어 Hardware 성능의 한계를 Software 로 해결하려는 노력도 증가하고 있고, 사용자들의 요구사항이 높아지고 복잡해 지면서 Embedded Software 개발에 대한 중요성이 높아지고 있다. [1]
Embedded Software 개발은 일반적인 Application Software 개발과는 달리 Software 와 Hardware 의 Co-Design 이 진행되기 때문에 Software 개발자의 Hardware 관련 지식이 요구된다. Hardware 관련 지식이 없으면 필요한 기능에 대한 Software 구현은 물론이고, Issue 가 발생했을 때 어떻게 접근해야 하는지에 대해서 많은 어려움을 겪게 된다.[2][3]
Embedded Software 의 개발 수요가 증가하고 있는 현 실정에서 효율적인 개발 조직 구성을 통해 효율적으로 개발 및 Issue 를 관리할 필요가 있다.
대규모의 프로젝트(프로젝트를 4~6 개월에 완료하기 위해 100~500 명의 인원이 투입)에서 각각 5 년, 11 년의 Embedded Software 개발 경험을 바탕으로 Embedded Software 개발과 다른 Software 개발의 차이점과 그로 인해 발생하는 문제점에 대해 분석하고, 조직을 구성하는 구체적인 방법을 제안하면서 조직의 리더가 어떻게 운영을 할 것인지에 중점을 둘 것이다.

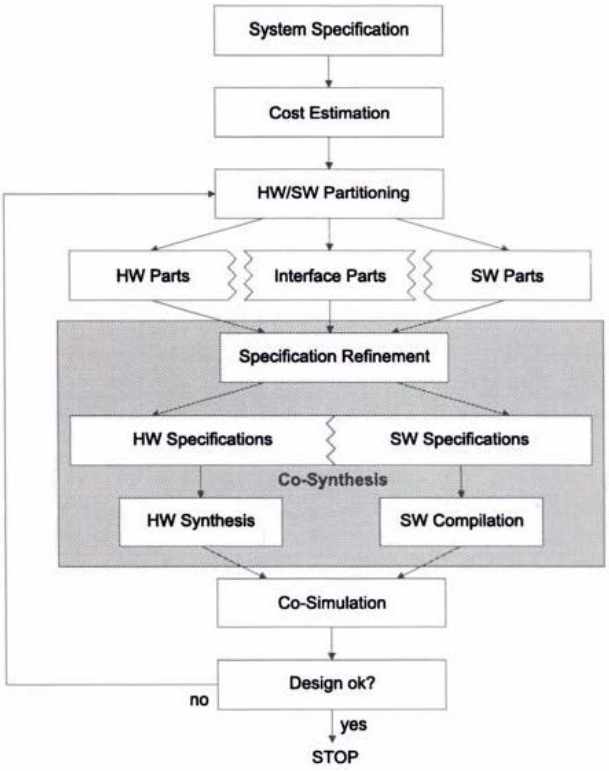


그림 1. Hardware/Software Co-design [3]

2. 현재의 Embedded Software 개발 상황

2.1 Embedded Software 개발의 특징

앞에서 설명했듯이, Embedded Software 개발은 Hardware 와 밀접한 관계가 있다. 개발하는 Hardware 의 동작에 직접적으로 영향을 주기 때문에 충분한 이해를 하고 있어야 요구사항에 맞는 기능을 구현할 수 있다. 그리고 Hardware Team 에서의 내부 요구사항에 대해서도 같이 수정하고 개발을 진행해

야 한다. [2][3]

Embedded Software 영역에서 Issue 가 발생하는 경우에는 Software Code 뿐만 아니라 Hardware 에 미치는 영향에 대해서도 고려해야만 원인과 해결방법을 찾아낼 수 있다. 하지만, 외부적인 요소에 의한 Hardware 특성의 변화에 따른 Issue 라던가 Hardware 내부의 동작의 오류로 인해 발생하는 Issue 들은 Root Cause 가 쉽게 드러나지 않을 뿐만 아니라 재현하기 어렵기 때문에 Debugging 하기가 쉽지 않다. 같은 이유로, Issue 를 해결하기 위해 Code 를 수정했다 하더라도 Regression Test 를 통해 Root Cause 가 해결 되었는지, 또 다른 Side Effect 가 없는지 확인하는 것이 쉽지 않다.

2.2 Embedded Software 개발 조직 사례 연구

이러한 Embedded Software 개발의 특징이 실제 개발 조직에서는 얼마나 잘 반영이 되고 있는지 조사해 보았다. 대규모의 프로젝트를 진행하는 개발조직 중, Embedded Software 개발 조직이 별도로 존재하는 경우에 대해서 세 가지 사례를 들어 문제점을 분석해 본다.

사례 1 : A 사의 경우, 개발 조직을 Hardware 개발팀과 Software 개발팀으로 구분해서 운영을 하고 있다. 그 중에서 Software 개발팀을 총괄하는 Project Leader 가 있고, 그 밑에 Application 을 개발하는 Team 과 Embedded Software 를 개발하는 Team 으로 나뉘어 있다. 그리고 Embedded Software Team 을 총괄하는 Leader 가 따로 존재한다. Embedded Software Team 산하에는 각 Module 별로 분화된 작은 Team 이 존재하고, 각 Team 별로 역시 Leader 가 존재한다.

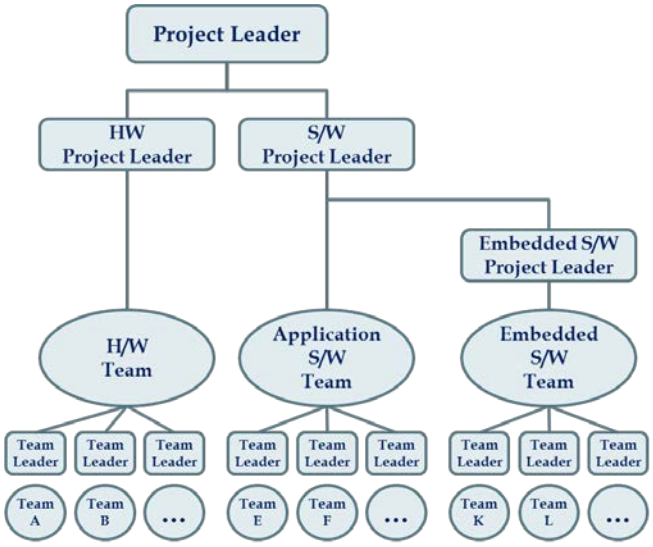


그림 2. A 사 개발 조직도 (예시)

만약, Software Issue 가 발생하게 되면 Software Project Leader 의 할당으로 Application Software Team 에서 먼저 분석이 진행되고, 만약, Embedded Software 에서 분석을 해야 한다고 판단되면 Embedded Software Team 에 할당하는 복잡한 Process 가 진행된다. 혹은, Hardware 에서 할당된 Issue 중에서 기구나 회로상의 문제가 아니라 Embedded Software 로 해결해야 하는 경우에도 Embedded Software Team 에 다시 할당하는 Process 를 거치게 된다.

Embedded Software Team 이 Issue 를 받게 되면 Embedded Software Project Leader 는 어느 Team 이 해당 Issue 를 처리해야 할지 분석 후 할당하게 되는데, 실제 담당자가 Issue 를 처리하게 되는 경우는 Issue 가 최초 발생한 이후 2~5 일, 오래 걸릴 경우는 2 주 후에 받게 된다. Embedded Software 는 앞에서 얘기한 것처럼 원인을 분석하고 해결하는데 시간이 많이 소요될 뿐만 아니라 내부 Test 도 거쳐야 하기 때문에 충분한 시간이 필요하게 된다. 하지만, Project Leader 의 입장에서는 Issue 가 빨리 해결되길 원하기 때문에 Embedded Software Team 은 항상 압박을 받게 된다

사례 2 : B 사에서 Smart Phone 을 개발 할 때 조직은 아래와 같은 형태로 구성된다.

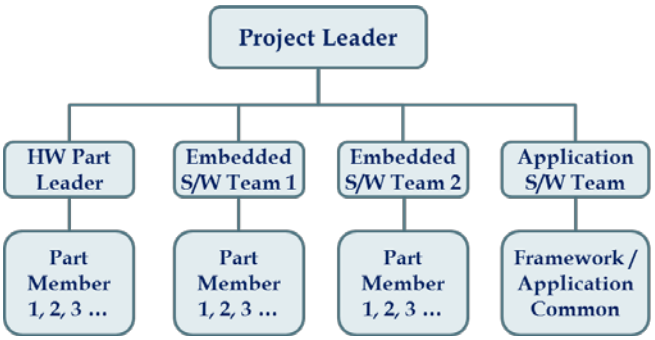


그림 3. B 사 개발 조직도 (예시)

프로젝트를 관리하는 리더가 있으며 하위에 Hardware Part 와 Embedded Software 개발 Part 가 존재한다.

Embedded Software 개발 Part 의 경우는 개발 규모 및 중요도에 따라 적게는 2 개의 Part 에서 많게는 5 개 Part 이상으로도 구성된다. 그리고, 모델 단위로 공통으로 사용될 수 있는 Framework 및 Application 부분은 따로 팀으로 운영되면서 그 팀과 연결을 담당하는 Window 가 포함된다. Embedded Software 개발 Part 의 경우는 보통 10 명 이상으로 구성이 되며 Part 내의 구성원들은 서로 특별한 연관성 없이 구성되어 있는 것이 보통이다.

이런 경우 Part 내에서 서로가 무슨 일을 하는지 모르는 경우가 다분하게 발생하며 심지어 Part 원이

많은 경우에는 Part Leader 가 파트 원에 대한 업무 정도를 판단하기도 힘든 상황이 발생 할 수 있다. 이러한 예는 System Test 단계에서 문제가 Assignment 되고 담당자가 그 문제를 Debugging 하는 상황에서 확인 할 수 있다.

사례 3 : C 사의 경우, Embedded Software 개발 조직은 각 Project 마다 Module 별로 담당자가 1~2 명 할당되어 있다.

동일한 Project 에 속해 있는 다른 Module 담당자와의 협조 및 Communication 에 대해 이점이 있지만, 각 Module 담당자는 다른 Project 에 속해있는 동일한 Module 담당자와는 서로 다른 Team 이기 때문에 담당 Module 에 대한 정보 공유가 이뤄지기 어렵다. 그리고 Embedded Software Project Leader 가 해당 Project 의 Embedded Software Team 을 총괄 관리하지만, 각 Module 에 대한 지식이 부족하기 때문에 Issue 의 할당 및 Balance 조절이 어렵다.

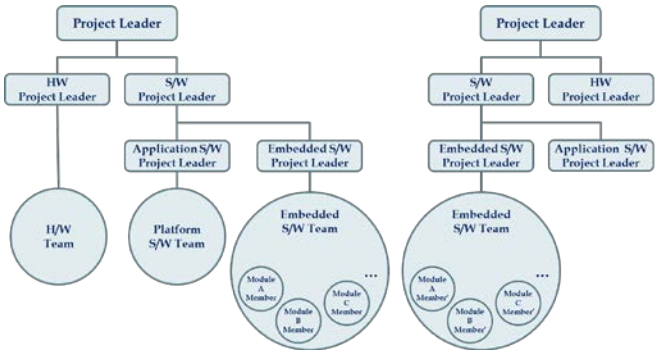


그림 4. C 사 개발 조직도 (예시)

실제 Embedded Software 개발팀에서는 Embedded Software 개발의 특징이 고려되어 개발이 진행되는 경우가 적은 것을 알 수 있다. 해당 사례들을 통해 문제점을 정리하면 아래 표 1 과 같다.

	A 사	B 사	C 사
Issue 받는데 걸리는 시간	2~14 일	1~5 일	1~5 일
Issue 해결 요구시간	1~3 일	1~3 일	1~3 일
Issue 할당 Balance	업무 능력 좋은 사람에게 치중	업무 능력 좋은 사람에게 치중	업무 능력 좋은 사람에게 치중
개발 관련 정보 공유	좋음	나쁨	나쁨

표 1. Embedded S/W 개발 조직 사례 분석

A 사의 Embedded Software Issue 는 여러 단계를 거쳐 분석이 이루어 지기 때문에 실제로 담당자에게 할당이 되는데 까지 오랜 시간이 걸리게 된다. Project 의 Leader 의 경우에는 Issue 들이 다음 Software Release 때 모두 해결되기를 바란다. 보통 Software Release 는 1 주일에 1~3 회 실시 되기 때문에 Embedded Software 개발 조직원들은 시간에 대한 압박을 받기 마련이다. 그리고, 명확하게 해결 담당자를 지정하지 못하게 되면, 해결 능력이 좋다고 판단되는 특정 인물들에게 Issue 해결이 몰리게 되는 문제도 발생하게 된다.

이러한 문제점들로 인해 실제 Embedded Software 개발에서 어떤 악영향이 미칠 수 있는지 알아보도록 하겠다.

3. Embedded Software 개발 시 문제점

3.1 It's NOT my Issue!

Test 과정 중에 문제가 발생하였을 때, Embedded Software 의 경우 누구의 Issue 인지 구분하기가 쉽지 않다. Leader 가 명확하게 누구의 Issue 인지 분석하지 못하는 경우에 적합한 담당자를 찾지 못하고 임의의 담당자에게 문제를 할당하게 된다. 이 때, 이미 분석중인 Issue 가 있거나 Due Date 에 대한 부담 때문에 문제의 원인을 찾아내기 보다는 자신과 관련이 없다는 근거만 찾고 다른 사람에게 넘겨버리는 폭탄 돌리기가 계속된다. 명확하게 Issue 의 원인을 분석하고 적합한 담당자를 찾기 위해서는 Embedded Software Leader 의 역할이 중요하다.

3.2 Interrupt Syndrome

정의 : 조직 구성원에게 계획에 의해 할당된 업무 외에 Interrupt 로 발생하는 추가적인 업무가 많아지게 되는 경우가 있다. 그런 Interrupt 로 인하여 해당 인원은 계획에 대한 신뢰가 떨어지게 되고, 본래 할당된 업무에 대한 효율과 의욕이 떨어진다. 나아가서는 Interrupt 로 발생하는 업무만 하게 되다 보니 자신의 업무의 우선순위가 낮아지고 언제 올지 모르는 Interrupt 업무만 기다리는 소극적인 업무 처리 태도를 갖게 되는 증상을 말한다.

Embedded Software 의 경우, 위에서 언급한 "It's NOT my Issue!" 로 인하여 폭탄 돌리기가 계속되다가 결국 누군가가 문제 해결을 해야 하기 때문에 Leader 는 업무 능력이 좋은 사람에게 업무를 할당해 버린다. 해당 개발자에게 이런 Interrupt 가 너무 많아 지게 되면, Multi-tasking 으로 업무를 처리할 수 밖에 없게 된다. 다른 Issue 분석을 위해 진행중인 Issue 분석을 중단해야 하는 경우가 발생하게 되고, 업무에 대한 집중과 효율성이 떨어지게 되는 문제가 발생하게 된다. [4]

3.2 Work-Around Code

Embedded Software Issue 의 원인의 분석과 해결 방안을 찾는데 시간이 많이 소요되는 경우가 대부분이지만 Due Date 가 충분하지 않고 Leader 에게 압박을 받게 되면 정상적으로 Issue 에 접근하기 어렵게 된다. 결국 Due Date 를 지키기 위해 문제를 해결하기 보다 회피하게 되는 Work-Around Code 를 반영하게 된다.

실제로도 Embedded Software Issue 의 상당수를 Work Around 로 해결하는 경우가 많고, 충분한 Test 도 진행하지 못해 Issue 가 실제로 해결되었는지 확인되지 않거나 잠재적인 Side Effect 가 우려되기도 한다. 그리고 Work Around Code 는 전체적인 Code 흐름 복잡해 지게 되고 해당 Code 를 재사용하게 될 경우에도 불필요한 Code 흐름을 거친다.

4. Embedded Software 개발 조직 구성

Embedded Software 분야는 분석 및 해결이 오래 걸린다. 그렇기 때문에 조직 구성의 측면에서는 Software 개발에 관련된 정보가 공유되어야 Issue 분석 및 해결에 소요되는 시간이 줄어든다. 그리고, Issue 의 Root Cause 를 찾기 어려워 담당자 할당에 문제를 겪는 경우가 많다. 이런 경우에는 경험과 관련 지식이 풍부한 개발자들이 함께 분석을 하면 Issue 해결 Point 를 보다 쉽고 빠르게 찾을 수 있다.

이런 조건들을 만족하기 위해서 Jelled Team 이라는 조직 구조를 Embedded Software 개발조직에 적용하는 것을 제안한다. 더불어 각 구성원들의 역할에 대해서도 살펴본다.

4.1 Making Jelled Team

Jelled Team 은 사람들끼리 강하게 엮여 있어서 개인들의 합보다 팀으로써 더 좋은 결과를 만들어 낼 수 있는 팀을 의미 한다. [5]

Embedded Software 개발은 일반적인 Software 개발보다 자신의 분야에 대한 boundary 가 좀더 강하게 작용 한다. 그렇기 때문에 Goal 의 공유를 위해서 적절한 단위로 구성원들을 조합하는 Jelled Team 을 만드는 것이 중요하다.

예를 들어 Smart Phone 을 개발하는 팀에 대해서 생각해 보자. 모두 주어진 기간 내에 고품질의 제품을 완성해야 한다는 공통의 목표는 있지만 그러한 목표는 너무 막연해서 Jelled Team 형성하기는 어렵다. 그렇다면 Storage 의 Read/Write 속도, Camera 의 초당 Capture 수 등의 현실적인 목표를 공유할 수 있는 단위의 그룹으로 나뉘어 진다면 어떨까?

그렇다면 좀더 많은 분야에서 Synergy 를 발휘 할 수 있을 것이다. 아래와 같은 조직 구성을 제안한다.

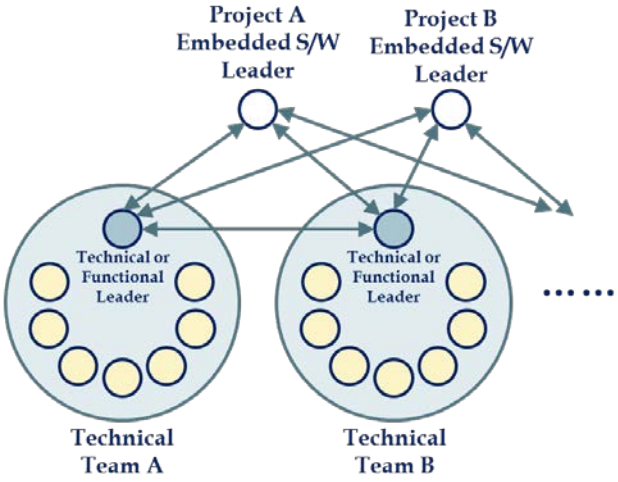


그림 5. Controlled Decentralized Team Structure [6]

위와 같이 Technical Leader 에 의해서 같은 Goal 을 공유할 수 있는 작은 조직(경험적으로 8 명 이내)을 구성함으로써 Jelled Team 을 형성 할 수 있다. 이러한 Team 은 Code 에 대한 Review 를 함께 진행할 수 있으며 System Test 에서 발생하는 문제에 대해서도 함께 Debugging 을 진행할 수 있게 된다. 그리고 해당 Module 에 대해 공통의 정보를 공유하고 있기 때문에 Issue 도 해당 팀 내에서 적절히 할당할 수 있고 팀의 기술적 숙련도도 향상된다.

4.2 Issue Assignment Process 및 Technical Leader 의 역할

앞에서 언급 하였던 "It's NOT my Issue!" 에 대해서 생각해 보자. Tester 에 의해서 발생한 Issue 가 Embedded Software 개발팀에 할당되고 Debugging 이 진행 된다. Issue 를 할당 받은 개발자는 촉박한 Due Date 로 인해 제일 먼저 자신의 문제인지 아닌지를 확인해 보게 된다. 만약, 자신의 문제가 아니라고 판단될 경우 다른 개발자에게 넘기거나 Leader 에게 재할당을 요청하게 된다. 이런 과정 중에서 명확한 담당자를 찾게 되면 상관없지만, 그렇지 않을 경우에는 계속 "It's NOT my Issue!" 를 외치면서 폭탄 돌리기가 끊임없이 되풀이된다. 특히나 Embedded Software 의 경우, Hardware 에 밀접한 Issue 나 재현이 어려운 사유 등으로 인해 담당자를 찾기 위한 Debugging 이 굉장히 힘들다. 이럴 때 일수록 Technical Leader 의 역할이 중요하다.

명확하게 담당자를 찾지 못하고 폭탄 돌리기가 계속되는 Issue 에 대해서는 Embedded Software Project Leader 와 Technical Leader 들이 함께 Issue 분석을 통해 담당 Team 을 찾을 수 있어야 한다. 물론,

Technical Leader 들이 충분한 Competence 를 갖고 있어야 가능하다.

뿐만 아니라 Technical Leader 는 항상 구성원들의 역량과 담당하고 있는 Issue 들에 대해 파악하고 있어야 한다. 어느 특정 담당자에게 Issue 가 집중되지 않도록 해야 하고, 함께 실력을 쌓는 것도 중요하다.

또한, Embedded Software 의 Issue 분석과 해결에 많은 시간이 소요되는 만큼, Project Leader 를 설득해서 충분한 시간을 확보 받는 것도 중요하다. 충분한 Issue 분석 시간은 Code 의 Quality 를 높이고 Side Effect 와 잠재적 Issue 를 줄이는 것이 중요하다.

4.3 Individual Competence

만약, 개발자들이 자신의 업무에 대해서 능숙(Competence)하지 못하다면, Project 를 성공적으로 완수 할 수 없게 된다. [7] Embedded Software 개발에 있어서 이 말은 더욱더 강조되어야 한다.

Embedded Software 는 Issue 에 대한 분석이 어렵고 시간이 오래 걸리는 특징이 있다고 앞에서 얘기하였다. Hardware 내부 동작과 관련되거나 System Software 의 Core 쪽 Issue 일수록 어디서부터 문제를 분석해야 하는지 알기 어렵다. 심지어는 자신이 분석해야 할 Issue 이지만, "It's NOT my Issue"로 다른 사람에게 전달하여 분석 시간을 더 소요된다. 그렇기 때문에 기본적으로 자신이 담당하고 있는 영역에 대해서는 어디서부터 문제를 접근해야 하고, 어떻게 해결해야 하는지 숙달되어야 한다.

Embedded Software 는 Hardware 와 Application Software 의 중간에서 동작한다. 그렇기 때문에 Hardware 와 Software Issue 에 대한 구분, Embedded 영역에서 Application 영역에 이르는 곳까지의 문제 파악을 쉽게 할 수 있는 능숙함이 필요하다. 또한, 자신이 해결해야 할 Issue 와 그렇지 않은 Issue 에 대한 구분을 얼마나 잘할 수 있는가도 Issue Debugging 시간에 많은 영향을 준다. 그렇지 않으면 Interrupt Syndrome 이 발생하게 된다.

Embedded System 개발에서는 System Test 의 비중이 매우 크다. 실제로 A 사와 B 사 의 System Test 의 비중은 전체 process 의 70% 이상을 차지한다. 그만큼 자신의 분야에 대해 확실히 알고 있어야 System 의 어느 부분이 잘못되었는지를 파악하고 해결할 수 있다. 이런 점을 고려했을 때 한가지 업무에 대한 깊이 있는 이해 여부가 중요하다. Hardware 에서부터 Application Software 영역까지 이해 할 수 있을 만큼 Training 과 경험을 통해 능숙함을 쌓는 것이 필요하다.

5. 결 론

Embedded Software 의 Issue 는 Application Software 의 Issue 와 비교하면 발생 건수가 비교적 적은 대신, 해결하는데 소요되는 시간이 길고 제품 자체에 Critical 한 경우가 많다. 이런 특징이 있기 때문에 Embedded Software 개발이 갖고 있는 특이사항을 고려하여 조직을 구성하고 운영해야 한다. Embedded Software 의 특성을 고려하지 않고 진행하게 되면 엄청난 잠재적 위험을 가져올 것이다.

Embedded Software 개발팀을 Technical Leader 를 필두로 하여 담당하는 Boundary 에 맞게 Team 을 구분해야 한다. 각 Team 에 속해 있는 담당자들은 담당하는 Boundary 에 대해 숙달되어야 하는 것은 물론이고 관련된 Hardware 나 Application Software 영역에 대해서도 관련 지식을 쌓아 전문성을 키워야 한다. 그리고 이런 Team 의 효율적인 운영을 위해 Technical Leader 가 Issue 를 나누고, 분석이 어려운 Issue 에 대해서는 Technical Leader 들간의 협력도 중요하다.

Embedded Software 개발 조직의 특성을 고려하여 조직을 운영하게 된다면 조직원들의 부담도 줄어들게 되고, Code Quality 도 향상될 수 있게 되어 성공적으로 Project 를 완수할 수 있다.

참고문헌

[1] 이정배, 이두원, "임베디드 시스템 연구 동향" 정보처리학회지 , 9 권, 1 호, pages 13-27

[2] Peter Marwedel, "Embedded System Design", Springer Science+Business Media, 2011

[3] Ralf Niemann, "Hardware/Software Co-Design for Data Flow Dominated Embedded Systems", Kluwer Academic Publishers, Pages 4~27

[4] Meade, Rodger, "Critical Chain Concepts" Scitor Corporation, January 2000

[5] Tom DeMarco, Timothy Lister, "Peopleware – Productive Projects and Teams", Dorset House Publishing Co., Pages 123-128, 1999

[6] Marilyn Mantei, "The effect of Programming Team Structures on Programming Tasks" Communications of the ACM, Volume 24 Issue 3, Pages 106-113, March 1981

[7] Alistair Cockburn, Jim Highsmith , " Agile Software Development : The People Factor", Software Management , November 2001

가전제품 사업에서의 소프트웨어 버그리포트 재현가능성 향상에 관한 연구

이창훈
㈜ 삼성전자
KAIST S/W 대학원
서울시 강남구 논현로 28 길 25
defkorn128@kaist.ac.kr

이미연
㈜ LG 전자
KAIST S/W 대학원
서울시 강남구 논현로 28 길 25
josephine.lee@kaist.ac.kr

민상윤
KAIST
KAIST S/W 대학원
서울시 강남구 논현로 28 길 25
symin@kaist.ac.kr

요약: Open source 를 기반으로 한 기존 연구에서 재현가능성 향상을 위해 다양한 정보를 제안하였다. 하지만 non-OS 기반의 가전제품 소프트웨어 개발에서는 버그 재현을 위해 제공되는 정보가 제한적이기 때문에 버그리포트의 재현성은 매우 중요한 요소이다. 이 연구에서는 버그리포트의 재현가능성을 높이고 non-OS 기반 가전제품 소프트웨어 개발에 적용할 수 있는 버그리포트 양식을 제안한다.

핵심어: 버그리포트, 버그리포트 양식 개선, 재현가능성

1. 서론

기존에 다양한 방법을 통해 버그리포트의 속성에 대한 연구가 계속 되어 왔다 [1,2]. 그 중에서도 버그리포트의 재현가능성은 문제점의 원인을 정확하게 파악하고 개선 검증을 위한 중요한 커뮤니케이션 속성이다. 개발자의 경우에는 버그리포트를 바탕으로 디버깅을 위한 정보를 얻기 때문에 버그리포트의 재현가능성이 중요하며, 테스터의 경우에는 버그리포트 내용에 대한 개발자와의 커뮤니케이션 오버헤드와 개선 후 검증 시 재현가능성이 업무 효율에 큰 영향을 미친다.

하지만 많은 연구가 open source 를 바탕으로 사용자-개발자 간의 버그리포트와 그 특성을 기반으로 하여 가전제품의 버그리포트에는 잘 적용되지 않는 특성이 있다 [3,4]. 예를 들어 가전제품의 경우 non-OS software 로 버그리포트에서 개발자가 원하는 정보를 줄 수 있는 Trace log, debug DLL 등의 사용이 어렵다. 이러한 제약사항으로 인해 버그리포트의 재현성은 매우 중요한 요소로 작용한다.

이 연구에서는 소프트웨어 버그리포트의 품질 속성 중 재현가능성을 높이고 non-OS 기반 가전제품 소프트웨어 개발에 적용할 수 있는 버그리포트 양식을 제안한다. 이것을 통해 개발자의 문제점 재현을 위한 시간을 절약하고 테스터와 개발자 간의 커뮤니케이션 오버헤드를 개선하여 생산성과 효율성을 제고한다.

본 논문은 다음과 같이 구성된다. 2 장에서는 현재 버그리포트 재현가능성 문제점을 커뮤니케이션의 이해관계자인 개발자와 테스터 관점에서 확인한다. 3 장에서는 버그리포트 재현가능성 향상을 위한 버그리포트 양식을 제안하며, 4 장에서는 기대효과 및 향후 계획을 포함한 결론을 기술하도록 한다.

2. 문제점 분석

현재 버그리포트의 문제점을 파악하기 위해 두 단계의 활동을 진행하였다. 먼저 A 사의 가전 제품을 담당하는 개발자와 테스터를 15 명 선정하여 그룹 인터뷰와 설문 조사를 수행하였다.

이후 인터뷰와 설문 조사를 통해 나온 재현가능성에 영향을 주는 정보의 유용성과 버그리포트의 만족도를 확인하기 위해 개발자에 대한 설문 조사를 추가로 진행하였다.

2.1 일반적인 버그리포트의 구성 요소

일반적으로 사용되는 버그리포트의 구성 요소는 아래와 같은 항목으로 이루어져 있으며, 이러한 구성요소를 기반으로 설문조사를 진행하였다 [5].

- 문제점 ID: 시스템에 저장되는 문제점의 ID
- 발생 부서: 문제점을 발견한 부서
- 버전: 문제점이 발생된 버전
- 우선 순위: 문제점의 해결 우선 순위
- 심각성: 문제점으로 인한 심각성
- 담당자: 문제점 해결 담당자
- 문제점 제목: 문제점의 제목

- 입력: 문제점이 발견된 테스트의 입력
- 예상되는 동작: 입력에 따른 예상 동작
- 실제 동작: 입력 시 실제 동작
- 테스트케이스 ID: 테스트 케이스의 ID

2.2 인터뷰와 1 차 설문 조사

사전 그룹 인터뷰와 설문 조사 각 1 회를 통해 재현을 위해 소요된 시간과 2.1 에서 제시된 일반적인 버그리포트의 각 구성 요소에 대한 유용성, 그리고 테스터에게 추가 질의한 사항에 대해 조사를 진행하였으며 그것의 세부적인 내용은 다음과 같다.

1) 대상자 통계 정보

설문에 참가한 대상자의 통계 정보는 다음과 같다.

	Product A	Product B	Product C	Product D	
Developer	6	2		3	11
Tester	1	1	1	1	4
	7	3	1	4	15

표 1 대상자 통계

총 4 개의 제품군에서 15 명 (개발자 11 명, 테스터 4 명)이 조사에 참가하였으며, 개발자의 경력 평균은 9.73 년이다.

2) 설문 내용

설문은 개발자와 테스터에게 각각 다음의 내용을 질문하였다.

A. 개발자
- 재현을 위해 소요된 평균 시간
- 재현을 위해 사용하는 정보의 유용성
- 재현을 위해 테스터에게 추가 질의한 정보
- 버그리포트에 추가 되어야 한다고 생각하는 데이터나 자료

B. 테스터
- 버그리포트를 작성하는 데에 소요되는 평균 시간
- 개선 검증을 위한 재현 시험에 소요되는 평균 시간
- 테스터에게 개발자가 문의한 추가 정보

3) 조사 결과

인터뷰 및 1 차 설문 조사를 통해 다음과 같은 내용을 도출하였다.

A. 재현을 위해 소요된 평균 시간

재현에 보통 소요되는 시간은 평균 1.9 시간으로, 응답자의 36%가 1 시간 이내로 재현을 위해 소요된다고 답변하였다.

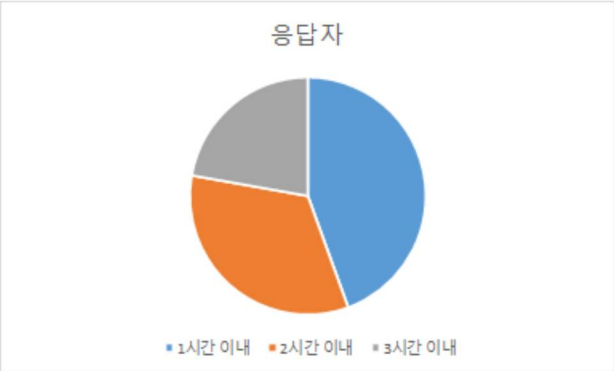


그림 1 재현 소요 시간

B. 재현을 위해 사용하는 정보의 유용성

재현을 위해 사용하는 정보는 Input/procedure (4.64), Actual Output (4.45), 그리고 Expected Output (4.27) 순서로 유용하다는 결과를 도출하였다.

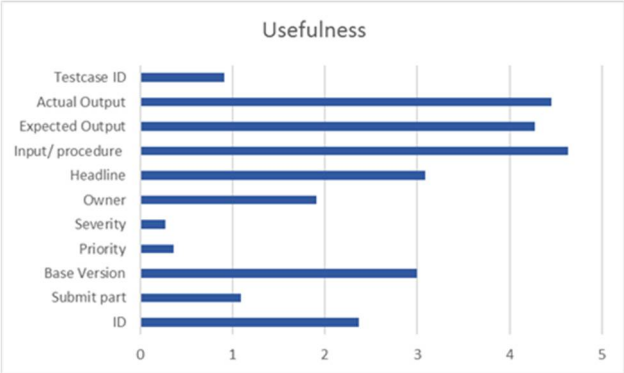


그림 2 재현 정보의 유용성

C. 재현을 위해 테스터에게 추가 질의한 정보

테스터에게 추가 질의한 내용으로는 description 에서 빠진 input(6 명), 재현 시 상황(4 명), 재현율(3 명) 등이 있다.

앞의 두 가지는 버그리포트에 포함된 input/procedure 의 내용이 불충분 하여 발생하는 것으로 해석되며, 재현율의 경우는 시료 등의 편차로 발생하는 문제점에 대한 보완 자료로 파악된다.

D. 버그리포트에 추가 되어야 한다고 생각하는 데이터나 자료

재현율 향상을 위한 추가 정보에는 동영상 및 차트 데이터(7 명)를 가장 많이 응답하였다. 이는 test input/procedure 의 내용만으로 전체 시험과정을 이해하기 어렵기 때문에 필요한 보조 자료로 파악된다.

E. 버그리포트를 작성하는 데에 소요되는 평균 시간

테스터들은 버그리포트를 작성하는 데에 평

균 12.5 분이 소요된다고 답하였다.

F. 개선 검증을 위한 재현 시험에 소요되는 평균 시간

개선 검증을 위한 재현 시험에 소요되는 평균 시간은 평균 1.5 시간으로 개발자와 유사한 시간을 재현 시험을 위해 투입하였다.

G. 테스터에게 개발자가 문의한 추가 정보

추가 문의된 자료로는 재현 시 상황에 대한 문의사항이 가장 많았다. 이는 개발자가 테스터의 description 에 따라 동일한 순서로 재현을 시도하였을 때 재현이 되지 않아 추가 환경 정보를 얻고자 한 것으로 파악된다.

2.3 개발자 대상 2 차 설문 조사

설문 조사 결과 재현 경로(input/procedure)에 대한 선호도가 제일 높은 것으로 파악되어, 이에 대한 상세 내용 확인 및 개선 전후의 정도를 확인하기 위한 정성적 평가를 진행하였다.

1) 설문 내용

2 차 설문은 개발자에게 다음의 내용을 질문하였다.

- 버그리포트의 내용으로 재현이 불가능한 경우 테스터에게 추가 문의한 정보의 유용성
- 현재 버그리포트의 재현가능성 측면 만족도

2) 조사 결과

2 차 설문 조사를 통해 다음과 같은 내용을 도출하였다.

A. 재현이 잘 되지 않는 버그리포트를 위해 테스터에게 추가 문의한 정보의 유용성

재현이 잘 되지 않을 경우 개발자가 테스터에게 문의한 정보에 대한 유용성에 대해 정확한 test sequence (4.8), 문제점에 대한 동영상이나 동작 상태 데이터 (4.7), 그리고 문제점 발생 시점의 시험 환경 (4.3)의 순서로 유용하다고 평가하였다.

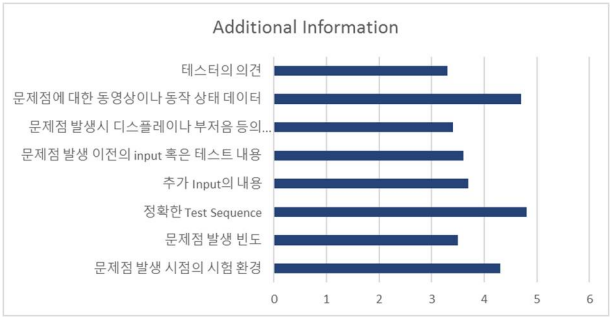


그림 3 추가 정보의 유용성

B. 현재 버그리포트의 재현가능성 측면 만족도

현재 제공되는 버그리포트에 대해서 '재현가능성' 측면에서의 만족도는 2.9 이다.

3. 개선안

2 차에 거친 설문 조사를 통해 시험 환경, 재현을 위한 입력 내용 및 순서에 대한 정확한 정보를 개발자가 가장 크게 필요로 함을 확인하였다.

제품 개발 과정 시험의 경우 일반적으로 테스트케이스 기반의 시험과 random test 두 가지로 운영된다. 이에 따라 이번 연구에서는 두 가지 다른 시험 방식에 대해 각각 재현가능성을 높이는 버그리포트 양식을 제안한다.

3.1 기존 테스트케이스 기반 시험을 위한 버그리포트

실행 조건, 입력 값 그리고 예상 결과 값의 조합으로 이루어진 테스트케이스 기반의 시험의 경우, 테스트를 위한 입력 값이 테스트케이스에 기록되어 있기 때문에 테스터가 입력 값에 대한 정보를 놓칠 가능성이 낮다.

하지만 여러 테스트케이스들을 연속적으로 수행할 경우에는 테스터가 연속된 테스트케이스 중 일부 입력 값을 놓칠 가능성이 있다. 또한, 이전에 수행한 테스트케이스가 현재 수행하는 테스트케이스에서 검출된 문제점의 원인이 될 수 있다. 이러한 경우에 이전에 수행한 테스트케이스에 대한 정보가 없다면 문제점 재현이 되지 않을 수 있다. 이를 방지하기 위해 각 테스트케이스의 시작에 입력 정보와 수행된 테스트케이스의 번호를 입력할 수 있도록 버그리포트 양식에 추가해야 한다.

또한, 개발자가 문제점 재현을 하기 위한 시험 환경과 문제점을 발견한 테스터의 시험 환경이 항상 일치하지는 않으며 테스트케이스의 입력 값만으로는 시험 환경의 차이를 알 수가 없다. 이것으로 인해 문제점 재현이 되지 않는 경우가 발생한다. 이러한 문제점을 개선하기 위해 테스터가 입력 값에 시험 환경에 대한 추가적인 정보를 기입할 수 있도록 해야 한다.

1) 제안 도구의 구성

A. 시험 환경에 대한 정보 항목 수집

테스트케이스의 입력 값만으로 표현하지 못하는 시험 환경에 대한 항목을 수집한다. 프로젝트 매니저와 테스트 매니저가 개발 초기에 기존에 수행된 유사 프로젝트를 선정한다. 선

정된 프로젝트에서 문제점 재현이 어려운 경우에 개발자가 테스터에게 문의한 내용들과 테스터들이 개발자에게서 요청 받은 내용들을 수집한다. 이러한 항목들을 일반화하여 요청 빈도가 높은 순서로 테스트 환경에 대한 부가 정보 항목을 선정한다.

B. 선정된 항목을 문제점 입력 시스템에 추가

개발자와 테스터로부터 수집한 시험 환경에 대한 부가 정보 항목들을 문제점 입력 시스템에 추가한다. 그림 4 와 같이 문제점 입력 초기 화면에서 입력란에 부가 정보를 입력을 위한 기본 값 추가하여 테스터가 해당 부가 정보를 입력하도록 한다.

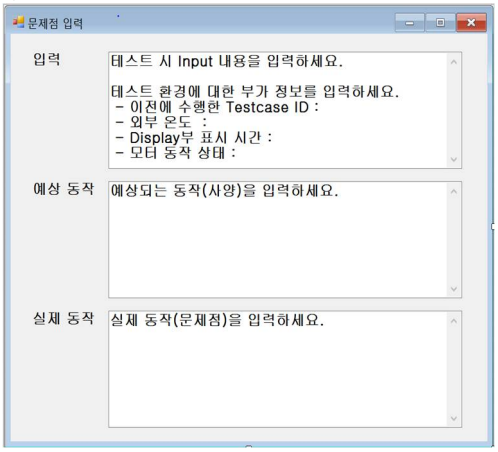


그림 4 부가정보 입력을 위한 양식 예시

3.2 Random test 를 위한 버그리포트

Random test 의 경우, 입력 내용과 순서는 상대적으로 테스터가 초기부터 입력을 시작하지 않으면 실제 문제 현상을 확인한 후 복기 하기 어렵고 테스터의 경험과 역량에 버그리포트의 품질의 편차가 생긴다. 따라서 테스트 중간에 계속 테스터가 입력 내용을 표시할 수 있는 보조 수단이 필요하다고 보고 이 연구에서는 다음의 도구를 제안한다.

1) 제안 도구의 구성

A. 제품 동작의 추상화

개발 초기에 프로젝트 매니저와 테스트 매니저가 테스트에 투입되는 제품의 사양에 기반하여 테스트 되어야 하는 제품의 동작을 추상화하여 동작 목록을 작성한다.

예를 들어 4 개의 일반 버튼과 1 개의 조그 버튼, 전원 버튼, 시작/종료 버튼, 그리고 취소 버튼으로 구성된 제품이 있을 경우 다음과 같은 동작 목록을 얻을 수 있다.

- 버튼 A 를 누른다
- 버튼 B 를 누른다
- …
- 취소 버튼을 누른다

B. 추상화된 단계의 엑셀 파일화

단계 1 에서 추상화된 동작은 따로 입력 방법을 익히지 않아도 되도록 익숙한 엑셀 파일을 이용하여 입력이 가능하도록 템플릿을 작성한다. 위에서 추상화한 제품에 대한 입력 템플릿은 다음과 같다.

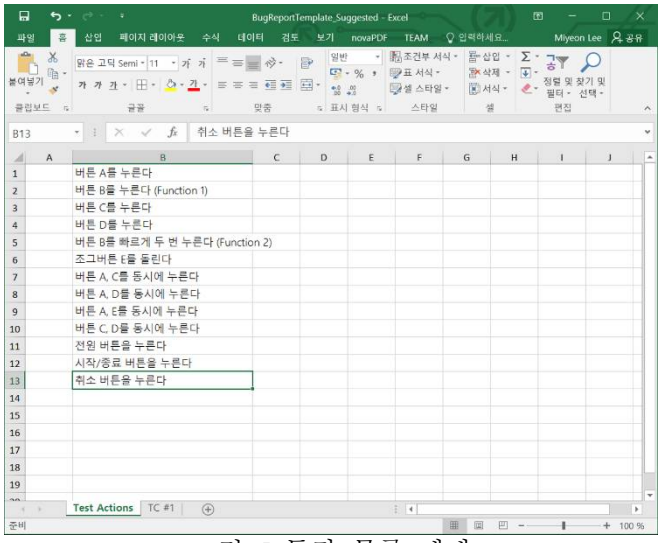


그림 5 동작 목록 예제

C. 활용

단계 2 를 통해 테스터는 테스트 동작 시 엑셀 파일에서 해당 동작을 선택하여 문제가 발생할 때까지 입력을 누적한다. 테스트가 완료되면 테스터는 버그리포트 입력 시 작성된 엑셀 파일을 첨부로 전달하여 입력 내용을 개발자가 파악할 수 있도록 한다.

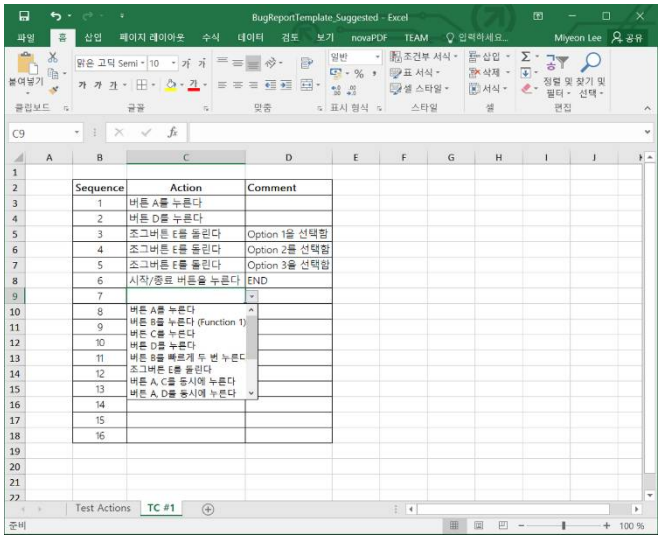


그림 6 시험 시 입력 예제

4. 평가

현재 사용 중인 버그리포트와 비교를 위해, 제안된 방식으로 작성된 버그리포트를 제시하여 비교 설문을 실시하였다.

4.1 개발자

1) 설문 내용

앞에서 제안한 방법으로 작성한 버그리포트의 예시를 제공하여 다음의 질문을 개발자에게 제시하였다.

- 제안된 양식을 사용하였을 경우 기존 버그리포트 대비 만족도

2) 조사 결과

개발자의 경우 만족도가 2.9 에서 4 로 37% 증가하였다. 이는 개발자가 원하는 테스트 환경에 대한 부가 정보가 제공되어 만족도가 증가한 것으로 보인다.

4.2 테스터

1) 설문 내용

앞에서 제안한 방법으로 작성한 버그리포트의 예시를 제공하여 다음의 질문을 개발자에게 제시하였다.

- 제안된 템플릿을 사용하였을 경우 기존 버그리포트 대비 예상 작성 시간 조사

2) 조사 결과

테스터의 경우, 제안한 양식을 이용하여 작성할 경우 16.25 분이 소요될 것으로 예상하였다. 기존 12.5 분 대비 30% 증가할 것으로 파악하였는데, 이는 테스터가 버그리포트 작성을 위해 테스트 환경 정보 및 각 step 을 기억하여 입력을 해야 하는 편의성의 감소가 반영된 것으로 보인다.

5. 결론 및 향후 계획

정성적 평가 결과, 개발자에게는 기존 버그리포트 대비 재현가능성 측면에서 높은 만족도를 부여함을 확인하였다.

제안된 양식은 기존 테스트케이스 기반의 테스트 및 random test 에서 제안한 도구 모두 테스터가 진행한 활동(사전 환경 및 동작)을 가능한 빠지지 않도록 한다는 장점을 통해, 설문 조사에서 확인된 test sequence 관점에서 기존 대비 나은 입력을 보장한다.

특히, random test 의 경우, 입력하는 동작에 대해 고정된 문장을 부여하여 다양한 경력의 테스터들로부터 일정한 수준 이상의 버그리포트를 확보할 수 있다. 또한 입력하는 동작에 대한 고정 문장을 사전에 테스트 매니저와 프로젝트 매니저가 함께 작성함으로써 테스터와 개발자가 같은 단어/문장을 사용하게 되어 동일한 이해를 보장하고, 더 나은 커뮤니케이션을 할 수 있게 한다.

하지만, 테스터의 정성적 평가 결과에서 볼 수 있는 것과 같이, 테스터는 입력을 해야 하는 정보의 증가에 따라 편의성이 일부 감소한다. 이는 이후 실제 양식을 개발 과정에 적용하여 커뮤니케이션 오버헤드와 함께 trade-off 를 검토할 필요가 있다.

또한 random test 에서 이 연구에서 제안한 도구의 경우, 입력 타이밍까지 누적하기에는 테스터에게 입력 로드가 걸리기 때문에 제약 사항이 확인되었다. 테스터의 입력 로드를 줄이면서 입력에 도움을 줄 수 있는 방법이 추가로 연구되어야 한다.

또한 기 존재하는 버그리포트 시스템에 직접 연동할 수 있는 plug-in 형태로 생성이 되면 따로 엑셀 파일을 첨부할 필요가 없어 더 단순한 사용이 가능할 것으로 보인다.

그 외에 설문 조사에서는 동영상 데이터에 대한 강한 needs 가 확인되었는데, 비디오의 경우 테스트 환경의 특성에 따라 바로 적용하기 어렵고(예. 물을 사용하는 제품의 경우 삼각대만으로는 테스트 환경의 안정성을 보장하기 어려움) 비용의 문제가 있기 때문에 장기적인 관점에서 접근할 필요가 있다.

참고문헌

[1] Nicolas Bettenburg: “What makes a good bug report?”, SIGSOFT '08/FSE-16, 2008

[2] Silvia Breu: “Information needs in bug reports : improving cooperation between developers and users” CSCW ’10, 2010

[3] Dhaval Vyas: “Bug Reproduction: A Collaborative Practice Within Software Maintenance Activities”, COOP 2014, 2014

[4] Mona Erfani Joorabchi: “Works for me! characterizing non-reproducible bug reports”, MSR 2014, 2014

[5] i Nicolas Bettenburg: “Quality of Bug Reports in Eclipse”, eclipse ’07, 2007

메모리 영역 분할을 통한 메모리 갱신 정보 기반 결함 후보 축소 기법

김관효, 최기용, 이정원

아주대학교 전자공학과
경기도 수원시 영통구 월드컵로 206
khyo0317@gmail.com, ki815kaisian@gmail.com, jungwony@ajou.ac.kr

요약: 최근 자동차 기술의 발전은 전자장치와 소프트웨어에 기반하여 발생하고 있다. 이에 따라 차량용 소프트웨어의 중요성이 증가하면서 소프트웨어의 고품질을 보장할 수 있는 체계적인 테스트 기법이 요구되고 있다. 하지만 차량용 소프트웨어 테스트 및 디버깅 과정 특성상, 개발자가 디버깅을 수행할 때 디버깅 정보의 부족으로 인해 결함 위치를 찾는 과정에서 막대한 비용과 시간이 소모된다. 따라서 본 논문에서는 개발자의 효율적인 디버깅을 위해, 검사자가 테스트를 할 때 결함이 발생한 테스트에 대해 결함 후보를 축소할 수 있는 기법을 제안한다. 본 기법에서는, 메모리 맵에 의해 분할된 메모리 영역에 메모리 갱신 정보를 적용하여 분할 영역들의 결함 의심도를 계산한다. 본 기법을 이용하여 결함 후보 영역을 전체 메모리 크기의 1/6 정도로 축소시킬 수 있음을 확인하였다.

핵심어: 차량용 전자장치, 차량용 소프트웨어, 결함 위치 추정, 임베디드 테스팅, 메모리 맵, 메모리 갱신

1. 서론

최근 자동차 업계에서 발생한 자동차 기술의 발전은 전자장치와 소프트웨어를 기반으로 하고 있다. 그런데 이러한 기술들은 운전자의 요구를 만족시키고 법적 요구사항에 부합하기 위해 고품질을 보장해야 하기 때문에, 이를 위한 체계적인 테스트 기법이 요구되고 있다.

하지만 차량용 소프트웨어의 디버깅 과정에서, 개발자가 이용할 수 있는 디버깅 정보의 부족으로 인해 결함 위치 추정에 막대한 비용과 시간이 소모된다. 이러한 문제의 근본적인 원인은 자동차 업계에서 차량용 전자장치의 개발, 테스트, 디버깅 과정에 대해 취하고 있는 OEM(Original Equipment Manufacturer) 방식에 있다. OEM 방식으로 인해 검사자와 개발자는 독립적으로 분리된 환경에서 테스트와 디버깅을 수행하기 때문에, 검사자와 개발자 사이의 정보 교환이 자유롭지 못하다. 즉 검사자는 HiL(Hardware-in-the-Loop) 시뮬레이션을 이용한 블랙박스 테스팅을 수행하고, 개발자는 블랙박스 테스팅의 결과와 같이 제한적인 정보를 이용하여 디버깅을 수행하기 때문에 결함 위치를 찾는데 막대한 비용과 시간이 소모된다.

이러한 문제를 해결하기 위해서는 블랙박스 환경에서 이용 가능한 정보인 메모리 갱신 정보를 이용하여 개발자에게 결함 위치 정보를 제공해줄 수 있는 기법이 필요하다[1]. 하지만 메모리에서 발생하는 갱신 정보의 양은 너무 방대하기 때문에, 메모리 맵을 이용하여 메모리를 분할하고 메모리 갱신 정보를 적용하여 메모리 상의 결함 후보 영역을 축소하는 기법을 제안한다.

2. 결함 후보 축소 기법

본 논문에서 제안하는 기법은 메모리 영역 중에서 데이터 섹션을 대상으로 한다. 그리고 메모리 정보는 CAN(Control Area Network)를 활용하여 프로그램의 실행 중에 특정 주기로 캡쳐하며, 각 시점마다 캡쳐하는 메모리 정보의 집합을 프레임(Frame)이라고 정의한다. 이 때, 메모리 정보는 각 주소에 있는 값(value)을 의미하고, 특정 프레임과 그 이전 프레임의 값을 비교하여 메모리 갱신 정보를 획득한다.

제안하는 기법에서는, 먼저 메모리 맵을 이용하여 데이터 섹션을 오브젝트 파일 단위로 분할한다. 이렇게 분할된 메모리 영역에 메모리 갱신 정보를 적용한다. 즉, 각 갱신 정보의 주소가 속하는 분할 영역을 확인하여, 특정 시점에서 각 분할 영역의 갱신

"이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No. NRF-2014M3C4A7030504)."

주소의 개수를 계산한다. 이 때, 갱신 주소의 개수가 절대적으로 많은 분할 영역은 현재 실행 중인 오브젝트 파일에게 할당된 영역으로, 갱신 주소가 0 개인 분할 영역은 현재 실행 중이 아닌 오브젝트 파일에게 할당된 영역으로, 그리고 갱신 주소의 개수가 평균 갱신 주소의 개수보다 작고 0 개보다 많은 영역은 결함 의심 영역으로 판단하고 결함 의심도를 증가시킨다. 이렇게 전체 프레임에 대해서 누적되어 계산된 결함 의심도를 기반으로 분할 영역들의 결함 의심 순위가 결정되며, 개발자는 결함 의심 순위가 높은 영역부터 우선적으로 디버깅을 할 수 있다. 이러한 과정은 그림 1과 같이 총 5단계로 구성되며, 각 단계에 대한 설명은 다음과 같다.

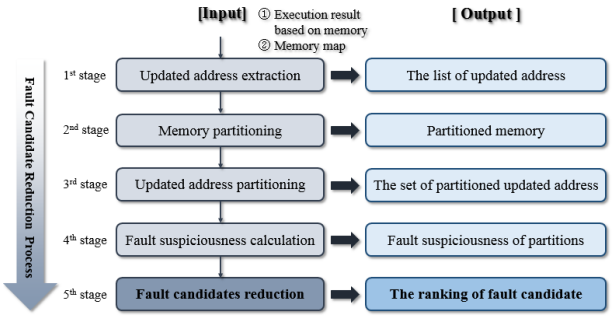


그림 1. 결함 후보 축소 과정

- 1st stage. Updated address extraction: 프로그램 실행 중에 획득한 메모리 정보를 이용하여, 각 시점에서 갱신이 발생하는 메모리 주소를 추출한다.
- 2nd stage. Memory partitioning: 메모리 맵을 이용하여 오브젝트 파일 단위로 메모리를 분할한다.
- 3rd stage. Updated address partitioning: 각 메모리 갱신 주소가 속하는 분할 영역을 결정한다. 즉, 각 분할 영역의 갱신 주소의 개수를 계산한다.
- 4th stage. Fault suspiciousness calculation: 메모리 정보를 획득한 각 시점에서 결함이 의심되는 분할 영역을 결정하고 해당 분할 영역의 결함 의심도를 증가시킨다. 즉, 결함 의심도는 결함 의심 영역으로 결정된 횟수를 의미한다.
- 5th stage. Fault candidates reduction: 각 분할 영역들에 대해 계산된 결함 의심도를 기준으로 결함 의심 순위 결정하고 결함 후보를 축소한다.

3. 실험 결과

제안하는 기법에서 대상으로 하는 메모리 정보는 10ms 의 주기로, 400 frame 만큼 캡쳐된다. 이렇게 획득한 메모리 정보는 메모리 맵에 의해 총 308 개의 오브젝트 파일 단위로 분할되고, 이 중에서 메모리 갱신이 한번 이상 발생하는 분할 영역의 개수는 67 개이다. 각 프레임마다 67 개의 분할 영역을 대상으로 결함 의심 영역을 결정하여 계산된 결함 의심도는 표 1과 같다. 표 1은 결함 의심도를 기준으로 상위 10 개의 오브젝트 파일들을 나타낸 것이다. 이 때, 실제 테스팅을 수행했던 검사자와 결함을 수정했던 개발자에게 #Object 58 에 결함 관련 주소들이 존재하는 것을 확인을 받았다. 결론적으로 전체 데이터 섹션의 크기는 25,228 byte 이고 표 1에 있는 데이터 섹션의 크기는 3,786 byte 이기 때문에, 우선적으로 디버깅할 수 있는 메모리 영역의 크기를 $(3{,}786 \div 25{,}228) \times 100(\%) = 15.01(\%)$으로 축소할 수 있었다.

표 1. 결함 의심도에 따른 결함 의심 순위

Ranking	Object File	Fault Suspiciousness
1	Object #8	399
2	Object #4	337
3	Object #49	333
4	Object #35	325
5	Object #59	210
6	Object #53	193
7	Object #10	186
8	Object #7	171
9	Object #58	156
10	Object #66	150

4. 결 론

본 논문에서는 HiL 시뮬레이션을 이용한 블랙박스 테스팅 환경에서 검사자가 차량용 소프트웨어를 테스트할 때, 결함이 발생한 경우에 한해서 결함 후보를 축소하는 기법에 대해 제안하였다. 본 기법은 메모리 맵과 메모리 갱신 정보를 이용하기 때문에 두 가지의 장점을 가진다. 첫째, 소스 코드를 이용할 수 없는 환경에 적용 가능하다. 둘째, 다양한 인터페이스를 가진 소프트웨어 테스트에 적용 가능하다.

향후에는 메모리 맵 없이, 스택 정보를 이용하여 메모리 분할을 하여 결함후보를 축소할 수 있는 방법에 대한 연구를 진행할 계획이다.

참고문헌

[1] Ki-Yong Choi, Jooyoung Seo, Seung Yeun Jang and Jung-Won Lee, “HiL Testing based Fault Localization Method using Memory Update Frequency”, The 10th KIPS International Conference on Ubiquitous Information Technologies and Applications, Vol.373, No.1, pp.765-772, 2015

IR 기반 결함위치추적을 위한 사용정보 영향력 분석

안준, 염창선, 김정호, 이은석

성균관대학교 전자전기컴퓨터공학과
경기 수원시 장안구 서부로 2066
{ahnjune, klausyoum, jeonghodot, leees} @skku.edu

요약: 소프트웨어의 결함 위치 식별을 위해 중요한 자료로 버그 리포트가 사용되고 있다. 버그 리포트에는 제목, 설명, 상태, 버전 정보, 결함 로그, 결함 이미지 등이 담겨 있으며, 제목과 설명에는 버그가 발생한 시나리오와 Stack-Trace 정보가 포함되어 있다. 또한 커밋 로그(Commit-Log)에는 결함 혹은 프로젝트 업데이트를 통해 갱신되는 파일로그, 설명 등이 포함 되어 있다. 본 논문에서는 이 정보들을 마이닝을 통해 결함의 위치를 추적하는 사용정보로 활용하려 한다. 특히 기존 정보검색 기반 결함위치 추적에서 사용된 버그 리포트와 커밋 로그의 영향력에 대한 분석을 통해 정보검색 기반 결함위치 추적에 사용되는 요소들의 상대적 중요성에 대한 실험적인 연구를 진행하였다. 실험을 위해 오픈소스 프로젝트(AspectJ, SWT, ZXing)를 이용하였고, 이들 프로젝트의 버그 리포트의 요약, 설명, Stack-Trace, 댓글 등 각각의 영향도를 확인했다.

구체적으로 본 실험을 통해 버그 리포트에 내포되어 있는 제목에서는 1.6%, 설명은 10.7%, Stack-Trace 7%, 댓글 9%의 영향력을 보였다. 또한 소스코드를 제외하고, 개발 중에 작성하게 되는 개발 문서와 버그 리포트의 영향력을 분석한 결과 제목 1%, 설명 7%, Stack-Trace 10%, 댓글 8%의 영향력을 보였다. 이러한 실험결과는 향후 효율적인 결함위치식별을 위한 전략수립에 활용 가능할 것이라 기대하고 있다.

핵심어: 정보검색, 결함위치 추적, 버그 리포트, 개발 문서, 전처리.

1. 연구배경

최근 소프트웨어 제품 개발은 개발 초기의 설계 및 구현의 비용보다 출시 이후 유지보수 단계에서 더욱 많은 비용이 발생한다[1]. 유지보수 단계의 소프트웨어 제품은 제품 출시 후 발견된 이슈/결함 내용이 개발팀에 리포트 된다. 개발팀은 리포트 받은 버그 리포트의 내용을 토대로 이슈 내용을 재현하고, 결함의 위치를 찾아 문제를 해결하기 위한 노력을 계속한다. 하지만, 하나의 버그 리포트를 해결하기 위해 소요되는 비용과 시간은 매우 크다. 또한, 대형 프로젝트의 경우 수많은 버그를 개발자가 적절하게 해결하기 또한 어렵다. 리포트된 버그의 해결 시간을 단축시킨다면 소프트웨어 제품의 유지보수 비용을 줄일 수 있고, 시장에서 발생된 결함을 빠르게 업데이트 함으로써, 제품의 품질을 크게 향상시킬 수 있다.

최근 버그 리포트의 내용을 정보검색 기법을 이용해 결함의 위치를 추적하는 연구들이 진행되고 있다[2,3]. 버그 리포트를 쿼리로 소스파일을 검색 대상의 문서 집합으로 간주하여 버그 리포트와 유사한 소스 파일을 찾는것 이다[4-9]. 여기에 이전 버그 리포트와의 유사성 분석 소스파일의 구조적 정보 활용, 버전 변경 이력분석, Stack-Trace 분석 등을 조합하여, 결함의 위치 추적의 정확도를 높이는 연구가 진행 되고 있다. 본 논문에서는 기존의 정보검색 기반 결함위치 추적에서 사용된 버그 리포트 내용과 커밋 로그의 영향력에 대한 분석을 통해 결함위치 추적에 사용되는 요소들의 상대적 중요성에 대한 실험적인 연구를 제안한다.

특히 저자들의 이전 연구인 정보검색 기반의 결함위치추적 시스템 “BLIA (Bug Localization Based on Code Change Histories and Bug Reports)”[10]에서 사용되었던 버그 리포트 정보(제목, 설명, Stack-Trace)와 커밋 로그가 결함위치 추적에서 차지하는 각각의 영향력을 심층 분석하는 연구 결과를 제시한다. 또한 이전 연구에서 사용되지 않았던 개발 문서와 버그 리포트에 결함을 수정하면서 나누는 대화인 댓글의 영향력을 확인한다.

본 논문의 구성은 다음과 같다. 제 2 장에서 정보검색 기반의 결함위치를 하기위해 사용되는 요소들에 대한 배경을 다루며, 제 3 장에서는 정보검색 기반 결함 위치 추적을 위한 사용정보 영향력 분석 방법에 대해 다룬다. 제 4 장에서 오픈소스 프로젝트 소개 및 실험에 대한 평가 방법에 대해 소개 한다. 제 5 장에서 소스코드와 개발 문서를 이용해 버그 리포트, 커밋 로그의 실험을 통해 요소들에 대한 평가

결과를 다루며, 제 6 장에서 영향력을 분석 한 결과를 논의하며, 제 7 장에서 향후 연구를 정리하며 맺는다.

2. 관련 연구

2.1 정보검색 기반 결함 위치 추적

정보검색 기반 결함 위치 추적연구에서는 버그 리포트와 커밋 로그를 사용한다. 일반적인 프로세스는 개발자가 버그 리포트를 전달 받고, 기존 연구들의 기법을 바탕으로 결함을 수정한다. 버그 리포트는 텍스트 정보(제목, 설명, Stack-Trace, 댓글)와 소스 코드 변경 이력인 Commit-Log 로 구성되며, 통합된 분석으로 결함 위치 추적의 정확도를 향상시킨다. 정보검색 기반 결함 위치 추적의 분석 과정은 다음과 같다. 첫 번째, 정보간 유사도 분석을 위해 버그 리포트의 텍스트 정보를 대상으로 소스 코드와 이전 버그 리포트 각각의 유사성을 분석한다. 이때, 소스 코드의 구조적인 정보(Abstract Syntax Tree)를 활용하여 정확도를 높인다. 두 번째, 버그 리포트에 Stack trace 정보가 있는 경우, 그 정보를 분석하여 의심도 점수를 계산한다. 세 번째, 이전의 소스 코드 변경 이력 및 버그 리포트 분석을 통해, 결함 의심 대상 리스트를 순위화한다.

앞서 얘기한 것과 같이, 버그 리포트를 받은 개발자는 일반적으로 버그 리포트의 내용을 기반으로 결함의 위치를 추적한다. 버그 리포트 내용에는 결함이 발생한 시간, 시나리오 및 제품의 버전 등의 정보가 포함되어 있고, 시나리오 상황에 따라 예외 상황이 발생한 경우는 해당 이슈 발생 시점의 Stack-trace 를 포함하고 있다. 개발자는 버그 리포트에 나온 이슈 파일부터 결함의 위치를 추적한다. 버그 리포트의 Stack-trace 는 개발자가 문제를 해결에 중요한 단서가 될 수 있다. Stack-trace 에 있는 method 와 소스 파일명을 토대로 결함이 발생한 코드 라인을 확인할 수 있다. 그리고, 개발자는 이전에 해결한 버그와 유사한 내용이 있는지 BTS(Bug Tracking System)에서 관련 결함을 찾아보기도 한다. 만약 유사한 버그가 확인되었다면, 해당 버그를 해결 하면서, 수정한 소스 파일들을 확인할 수 있다. 또한 형상관리시스템에서 수정한 이력들을 조회하여, Commit message 를 통해 최근에 수정한 이력 중에, 버그에 영향을 주었는지 확인 및 분석을 할 수 있다.
위에서 결함 위치추적을 위해, 분석 가능한 소스들을 정리하면 다음과 같다.

- 버그 리포트 내용(FixDate, OpenDate, 시나리오, Stack-trace, 댓글)
- 소스 파일
- 이전 해결된 유사 버그 정보
- 소스파일의 수정 이력(Commit Message)

2.2 버그 리포트 작성

버그 리포트는 결함을 수정하는 결정적인 도움을 준다. 버그 리포트 내에 작성되는 결함에 대한 설명, Stack-Trace 와 Log 등을 이용해 결함을 명확하고 빠르게 수정할 수 있도록 개발자에게 도을 준다.

또한 결함을 수정하고 해결하는 사후에 발생할 수 있는 동일한 결함의 예방 및 해결을 위해 버그 리포트를 작성하게 된다. 이러한 버그 리포트를 기록하고 관리하는 것이 BTS(Bug Tracking System)이다. BTS 는 Mantis, Bugzilla, Trac 등 많은 도구들이 있으며, 해당 도구를 이용해 버그 생성 및 버그가 해결되기까지의 일련의 과정들을 확인 할 수 있다.

버그 리포트에는 표 1 과 같이 다음과 같은 내용들을 작성할 수 있다. 다음 내용을 참고하여 개발자들은 버그를 식별하고 버그를 수정한다.

본 논문에서는 소프트웨어 결함과 직접적인 연관이 있는 버그 리포트를 이용해 버그와 연관된 소스 코드 파일을 찾게 되는 정보인 버그 리포트에 내포되어있는 내역들의 영향력을 평가하는 연구를 제안한다.

표 1 버그 리포트 형식

BugID	39626		
fixDate	03-8-28 6:38:00	openDate	03-7-4 5:26:00
Summary	thisAspectInstance compile problem with -1.8		
Description	imilar to other 'well known' local variables like thisJoinPoint, the eclipse 1.8 compiler is complaining that thisAspectInstance is not initialized when it is a variable that we (AspectJ) look after. The fix is the same as for thisJoinPoint et al - just speci java.lang.NullPointerException at lang.String.<init>(String.java:214) at AsmBuilder.visit(AsmBuilder.java:231) at AsmBuilder.visit(AsmBuilder.java:259) at …		
Comment	Changed the tabs so that it calculates the client area. This can't be done through one specific call. The area of the basic widget is taken, and then the area of the parent is taken. The resulting intersecting rectangle between the two , with the margins f		

2.3 소프트웨어 형상관리 시스템

소프트웨어 형상관리는 소프트웨어 생명주기 동안 발생하는 변경사항을 관리방법으로 형상관리를 통하여 요구사항을 만족시킬 수 있도록, 변경되는 요소들을 관리 하며, 개발 환경에서 소프트웨어의 변화에 대응할 수 있도록 소프트웨어 개발 및 변경되는 이력들을 관리하는 시스템이다.

형상관리 시스템에는 개발에 필요한 문서, 소스코드, 개발 도구 등 소프트웨어에서 개발되며 도출하게 되는 산출물들을 모두 포함하고 있다. 개발자는 결함이 발생했을 때, 버그 리포트의 내용을 바탕으로 의심되는 파일과 변경 이력을 확인하고 변경 이력의 내용이나 변경 코드 부분에서 의심스러운 부분이 있다면, 해당 부분을 위주로 중심으로 테스트 및 결함을 수정하게 된다[11, 12]. 이를 통한 수정이 어려운 경우, 개발자들은 형상 관리 시스템[13]으로 결함이 발생한 커밋 로그를 추적하며, 이런 시스템은 결함 위치 추적에 있어서 중요한 보조 역할을 한다.

2.4 개발 문서

소프트웨어의 대형화되고 복잡해 짐에 따라 대부분의 소프트웨어 개발은 다수의 개발자에 의해 공동개발이 진행되고 있다. 이런 상황에서 많은 기업들은 대부분의 인력 및 비용을 개발 및 테스팅에 소요하고 있으나, 그에 상응하는 문서화에는 어려움을 겪고 있다. 이 같은 문서화의 어려움은 소프트웨어 개발 중 발생할 수 있는 인력의 이동에 의해 심각한 문제를 야기할 수 있으며, 신규 개발자의 경우 타인의 소스코드를 분석해야 하는 어려움이 가중된다. 이러한 이유로 소스코드의 문서화는 선택이 아닌 필수사항이 되고 있다. 앞서 언급한 문제는 개발 문서 자동 생성을 통해 이전 개발자의 개발 흐름을 보다 쉽게 파악해 개발을 이어할 수 있다.

JavaDoc[14]과 Doxygen[15]은 소스코드의 문서화를 위한 지원도구로써 이러한 문제점의 완화를 위하여 활용될 수 있다. JavaDoc 과 Doxygen 은 문서화 작업을 위하여 개발자가 별도의 문서를 생성할 필요가 없으며, 소스코드에 정해진 태그를 활용하여 주석처리를 하는 것만으로 문서의 생성이 가능하도록 지원한다. 또한 Eclipse 에서는 이러한 문서화를 위한 많은 편의 기능을 제공하고 있어, 개발자의 문서화 부담을 완화하고 있다. 본 논문에서는 개발에 사용되었던 주석을 바탕으로 만들어지는 문서와 버그 리포트를 비교 분석해 정보검색 기반의 결함 위치 추적 연구의 성능을 향상시키려 한다.

2.5 정보검색 모델

본 논문에서는 검색 모델로 TF-IDF (Term Frequency - Inverse Document Frequency) 모델을 사용한다. TF-IDF 모델은 정보검색 및 텍스트 마이닝을 위해 문서의 단어간 상대적 중요도를 평가하기 위해 만들어진 모델이다. TF-IDF 의 점수를 통해 문서들 간의 순위를 결정할 수 있으며, 유사 문서들의 그룹을 찾는 문서군집화를 용이하게 한다. TF-IDF 점수가 크면 클수록 단어가 속해 있는 문서에 의미가 높다는 것이며, TF-IDF 점수를 통해 문서를 추출하는 척도로 사용할 수 있다.

$$TF = \frac{\mathrm{f(t,d)}}{\mathrm{maxf(w,d)}:w \in d} \quad (1)$$

$$idf(t,D) = \log \frac{|D|}{|d \in D : t \in d|} \quad (2)$$

$$TFIDFScore = \mathrm{tf(t,d)} \times idf(t,D) \quad (3)$$

식 1 의 TF 는 한 문서의 단어 출현 횟수를 나타내는 수식이며, 식 2 는 한 단어가 전체 문서들에서 출현하는지를 나타내는 수식이다. 식 3 은 TF-IDF 점수를 구하기 위한 식을 표현하고 있다. TF-IDF 의 점수를 구하기 위해 TF 값과 IDF 값을 곱해 문서의 가중치를 구하게 된다. 식 3 에서 TF 값은 문서 내 특정 단어의 출연 빈도를 의미하며, IDF 값은 문서 집합에 포함되어 있는 전체 문서를 특정 단어가 나타난 문서의 빈도로 나눈 것이다. 본 논문에서는 소스코드, 버그 리포트 그리고 커밋 로그로 구성된 문서들의 단어를 모두 추출해 문서들의 TFIDFScore 점수를 계산하며, 이를 위해 TF-IDF 모델을 사용해 실험한다.

3. 정보검색 기반 결함위치추적을 위한 사용정보 영향력 분석 방법

본 논문에서는 선행연구 [10]에서 사용된 버그 리포트(제목, 설명, Stack-Trace)와 커밋 로그 정보에 더해 개발 문서와 버그 리포트의 댓글 각각의 영향력을 분석한다. 이를 위해 TF-IDF 모델을 활용하여 각 요소들의 단어간의 유사성을 측정한다. 이때, TF-IDF 모델에 사용하기 위한 전체 요소들의 전처리 작업을 수행한다.

전처리 작업(Pre-Processing)은 정규화(Normalization), 불용어 제거(Stopword Removal), 어근화(Stemming)의 3 단계의 전처리 단계를 거치게

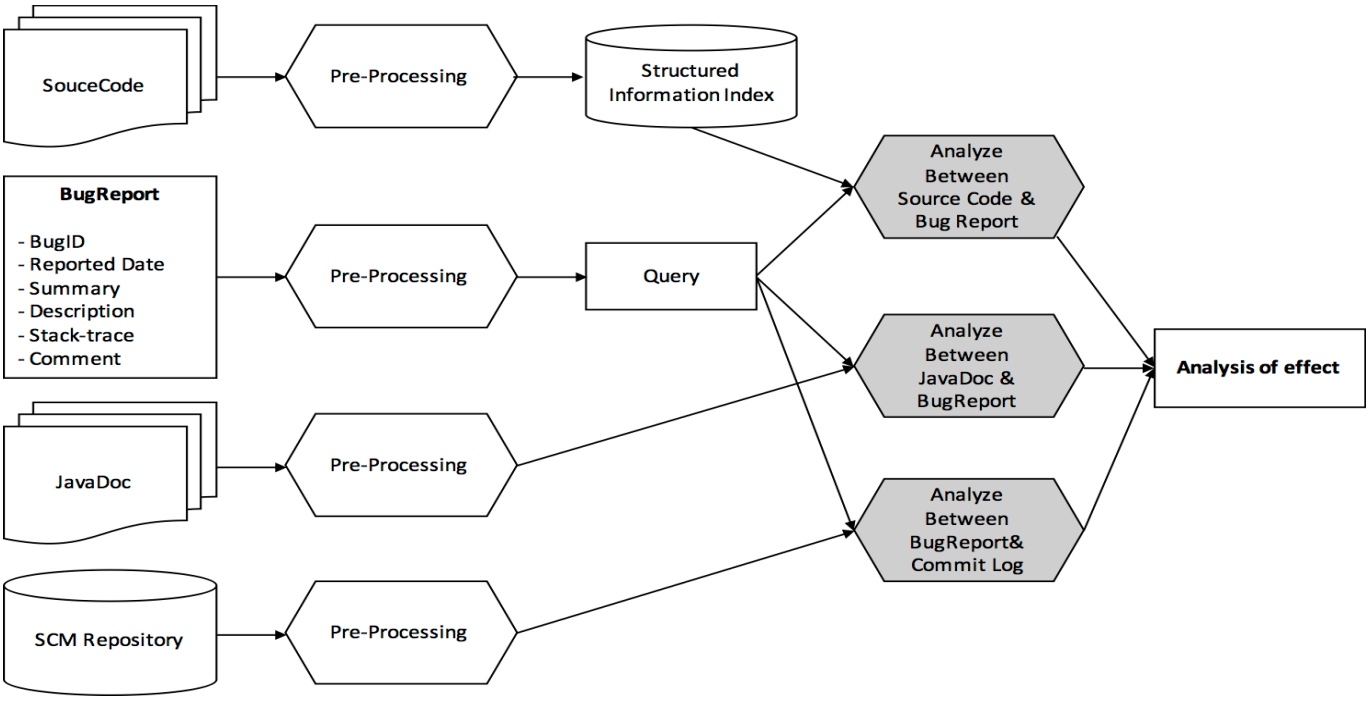


그림 1 IR 기반 결함위치추적을 위한 사용정보 영향력 분석 흐름

된다. 첫째, 정규화 작업은 "NullPointerException"과 같은 단어를 "Null", "Pointer", "Exception"으로 나누는 작업을 하며, 두번째, 불용어 제거 작업은 "a/an", "of", "as"와 같은 불필요한 던어들을 제거 한다. 마지막으로 어근화 작업은 각기 다른 단어들 "Computer", Computing"의 단어들을 "Compute"와 같은 형태로 만든다. 그림 1 과 같이 버그 리포트와 소스파일, 개발 문서, Commit-Log 들의 전처리 작업을 마치면, 정보검색(Information Retrieval) 분석을 위해, 파일내의 단어들을 추출하여 각 단어들의 정보를 생성한다. 소스 파일은 소스 파일의 내용과 구조적 정보(Abstract Syntax Tree)를 이용해 결함 위치 추적 정확도를 향상 시킨다. 버그 리포트는 소스 파일과 유사하게 전처리 과정을 통해, 쿼리화 한다. 버그 리포트(제목, 설명, Stack-Trace, 댓글)와 커밋 로그를 소스코드 파일, 개발 문서에 TF-IDF 모델을 이용해 계산한다. 소스코드파일을 통해 계산된 소스코드와 개발 문서를 이용해 버그 리포트와 커밋 로그의 각각의 요소들의 영향력을 확인한다.

4. 실험

버그 리포트(제목, 설명, Stack-Trace, 댓글), 커밋 로그, 개발 문서를 이용해 정보검색 기반 결함의 위치를 추적하기 위해 오픈소스 프로젝트를 이용해 실험 한다.

4.1 대상 프로젝트

표 2 오픈소스 프로젝트

프로젝트	설명	버전	버그 개수 (EA)	소스 파일 개수 (EA)
AspectJ	AOP 기반 자바 언어개발 프레임 워크	1.5.3	284	5,188
SWT	자바기반의 위젯 툴킷	3.659	98	738
ZXing	바코드 이미지 가공 라이브러리	1.6	20	391

표 2는 소스코드 파일을 이용해 버그 리포트, 커밋 로그의 영향력을 측정하기 위해 이전 연구에서 사용되었던 오픈소스 프로젝트 "AspectJ[16]", "SWT[17]", "ZXing[18]" 프로젝트를 사용한다.

4.2 평가항목

버그 리포트와 커밋 로그의 영향력을 측정하기 위

해 평균 정밀도(Average Precision)를 사용하였으며, 이를 통해 영향력을 측정한다.

$$AP = \frac{\sum_{r=1}^{N}(P(r)\times rel(r))}{\mathrm{N}} \quad (4)$$

식 4의 $P(r)$ 은 주어진 검색 결과에서의 정확도 값을 나타내며, $rel(r)$은 0과 1의 이진 값으로 만약 관련 있는 문서라면 1 아니면 0을 의미한다. 마지막으로 N은 전체 문서의 개수이다.

5. 논의

본 장에서는 표 2에서 제시한 대상 프로젝트를 이용해 정보 검색 기반 결함 위치 추적에 사용되는 요소들의 영향도 결과를 분석한다.

5.1 소스코드와 버그 리포트간 영향력 분석

본 실험은 이전 연구인 BLIA[10]에서 소스코드의 결함위치를 식별하기 위해 사용되었던 정보인 버그 리포트의 영향력을 분석한다. 표 3의 최소/최대 영향력 분석은 소스코드와 해당 결함을 담고 있는 버그리포트를 비교했을 때, 각각의 정보의 영향력을 나타내는 결과이다. 분석 결과 AspectJ 프로젝트에서 제목 1.6%, 설명 16.7%, Stack-Trace 7.1%, 댓글 9.4%의 평균 영향력을 가지고 있었으며, SWT 프로젝트에서는 제목 2.1%, 설명 6.2%, Stack-Trace 1.1%, Comment 9.7%의 평균 영향력을 가지고 있었다. 마지막으로, ZXing 프로젝트에서 제목 1.3%, 설명 8.7%, Stack-Trace 7.8%, 댓글 8.6%의 영향력을 가지고 있었다. 이로써 정보검색 기반 결함위치 추적 연구에서 중요한 요소로는 설명, 댓글, Stack-Trace, 제목 순 이였다.

실험을 통해 Stack-Trace 내용이 포함되어 있는 버그 리포트가 적은 것을 확인 할 수 있었으며, 결과는 아래와 같다. AspectJ 프로젝트의 경우 총 버그 리포트를 이용해 결함을 수정한 파일 1063개 의 소스 파일 중 141개만 Stack-Trace 정보가 포함되어 있는 것을 확인 할 수 있었으며, SWT 프로젝트에서는 결함을 수정한 파일 265개의 소스 파일 중 6개의 소스 파일에만 Stack-Trace 정보가 포함하고 있는 것을 확인 할 수 있었다. 마지막으로 ZXing 프로젝트에서는 결함을 수정한 파일 33개의 소스 파일 중 1개의 파일에만 Stack-Trace 정보가 포함되어 있었다. Stack-Trace 정보를 통해 정보 검색 기반 결함위치 추적 방법에서 효과적인 요소로 사용될 수 있지만, 버그 리포트 내에 Stack-Trace 내용이 적게 작성되는 것을 확인 할 수 있었다.

표 3은 각각의 프로젝트의 버그 리포트에 담긴 정보는 제목, 설명, Stack-Trace, 댓글의 영향력을 분석한 결과이며, 그림 2은 모든 프로젝트를 결과를 합쳐 평균적인 영향력을 분석한 결과이다.

표 3 소스코드와 버그 리포트를 이용한 영향력 분석

프로젝트	내용	최소	최대	평균
AspectJ	제목	0.0%	16.7%	1.6%
	설명	0.0%	**77.2%**	**10.7%**
	Stack-Trace	0.0%	24.7%	7.1%
	댓글	0.0%	40.4%	9.4%
SWT	제목	0.0%	8.9%	2.1%
	설명	0.0%	17.1%	6.2%
	Stack-Trace	0.0%	1.6%	1.1%
	댓글	0.0%	**22.1%**	**9.7%**
ZXing	제목	0.0%	9.4%	1.3%
	설명	0.0%	**15.7%**	**8.7%**
	Stack-Trace	0.0%	7.8%	7.8%
	댓글	0.0%	15.3%	8.6%

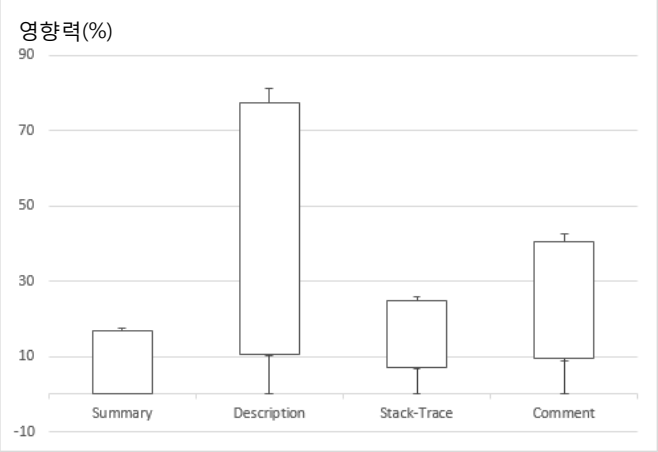


그림 2 소스코드와 버그 리포트의 영향력 표준 편차 그래프

5.2 개발 문서와 버그 리포트간 영향력 분석

본 실험은 소스코드를 제외하고 소스 코드 내에서 개발자가 개발도중에 작성하는 개발 문서 내용과 버그 리포트를 비교/분석해 영향력을 분석하였다. 표 4의 최소/최대 영향력 분석은 개발 문서와 해당 결함을 담고 있는 버그리포트를 비교했을 때, 각각의 영향력을 나태내는 결과이다. 실험을 통해 분석한 결과 AspectJ 프로젝트에서는 제목 1.4%, 설명

7.5%, Stack-Trace 10.1%, 댓글 8.6%의 평균 영향력을 나타냈으며, SWT 프로젝트에서는 제목 1.7%, 설명 12.5%, Stack-Trace 6.4%, 댓글 8.6%의 영향력을 나타냈다. 마지막으로 ZXing 프로젝트에서는 제목 1.2%, 설명 9.8%, Stack-Trace 5.2%, 댓글 5.5%의 평균 영향력을 나타냈다. 이전 연구와 유사하게 설명과 Stack-Trace, 댓글에서 의미 있는 결과를 나타냈다.

표 4 은 각각의 프로젝트의 개발 문서 내용과 버그 리포트에 담긴 제목, 설명, Stack-Trace, 댓글의 영향력을 분석한 결과이며, 그림 3 는 모든 프로젝트를 결과를 합쳐 평균적인 영향력을 분석한 결과이다.

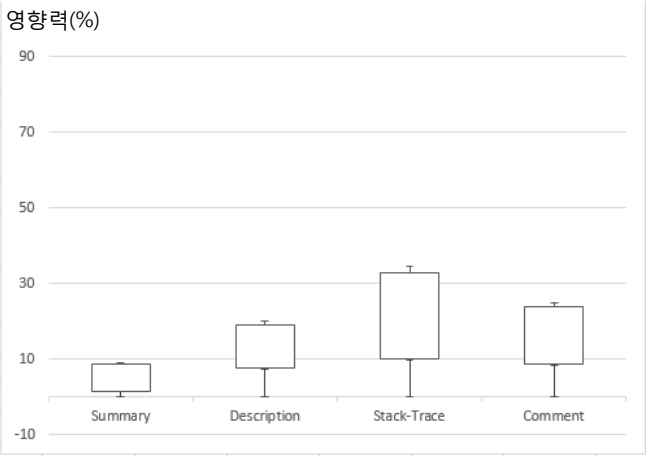


그림 3 개발 문서와 버그 리포트의 영향력 표준 편차

표 4 개발 문서와 버그 리포트를 이용한 영향력 분석

프로젝트	내용	최소	최대	평균
AspectJ	제목	0.0%	8.5%	1.4%
	설명	0.0%	19.1%	7.5%
	Stack-Trace	0.0%	**32.7%**	**10.1%**
	댓글	0.0%	23.7%	8.6%
SWT	제목	0.0%	10.8%	1.7%
	설명	0.0%	**21.4%**	**12.5%**
	Stack-Trace	0.0%	12.1%	6.4%
	댓글	0.0%	17.8%	8.6%
ZXing	제목	0.0%	7.8%	1.2%
	설명	0.0%	**15.6%**	**9.8%**
	Stack-Trace	0.0%	5.2%	5.2%
	댓글	0.0%	9.1%	5.5%

5.3 커밋 로그 영향력 분석

버그 리포트 내에는 많은 정보들이 포함 될 수 있지만, 반대로 버그 리포트를 작성하는 사람에 따라, 결함의 위치를 찾기 어려울 정도로 적은 내용이 담겨 있는 경우가 있다. 이 경우 정보검색 기반 결함 위치 추적에서 결함으로 의심되는 파일을 찾기 힘들다. 이런 약점을 고안하기 위해 사용되는 요소로 커밋 로그가 있다. 커밋 로그에는 추가/수정된 소스코드의 이력이 담겨 있다.

본 실험에서는 커밋 로그를 이용해 최근 Commit 에 [*bug*], [*fix*]의 내용이 담겨 있는 Commit 들을 정규 표현식으로 찾아 결함으로 의심되는 파일의 의심도를 계산해 영향력을 확인 하였다.

AspectJ 프로젝트의 8300 개 커밋 로그를 비교 분석한 결과 결함으로 의심되는 파일에 평균 9.2% 영향력을 나타낸다. 본 실험결과를 통해 표 5 와 같이 경험적으로 버그 리포트 내용이 미미할 경우 커밋 로그 분석을 통해 결함의 위치를 식별 할 수 있는 주요한 요소로 확인되었다.

표 5 커밋 로그 분석 영향력 분석 결과

프로젝트	최소	최대	평균
AspectJ	0.0%	25.9%	9.2%
SWT	0.0%	32.7%	11.3%
ZXing	0.0%	28.5%	9.4%

5.4 버그 리포트 내에 소스파일 이력 확인

표 6 버그 리포트 내에 소스코드 파일이름 존재 유무

프로젝트	AspectJ	SWT	Zxing
제목	20 개	27 개	0 개
설명	187 개	53 개	0 개
Stack-Trace	88 개	4 개	0 개
댓글	260 개	88 개	0 개

본 실험은 버그리포트를 작성할 때, 결함으로 의심되는 파일명을 테스터/개발자가 작성하는지 여부를 확인하기 위해 분석하였다. 대상 프로젝트 중 AspectJ 프로젝트의 경우 버그 리포트 내에 작성된 제목, 설명, Stack-Trace, 댓글 중 소스파일 이름이 1 회 이상 존재하는 요소는 총 1065 개 요소 중 346 회 발견 되었으며, 소스코드 파일 이름이 가장 많이 명시되어 있던 영역은 댓글, 설명, Stack-Trace, 제목 순 이였다. 이 같은 결과를 통해 정보검색 기반 결함 위치 추적을 수행하기 이전에 버그 리포트에 결함이 있는 파일의 이름이 존재한다면 위의 방법은 불필요한 방법으로 판단된다. 그러나 표 6 와 같이

프로젝트에 소스파일 이력으로 결함의 위치를 추적할 수 있지만, 프로젝트 특성에 따라 버그 리포트 내에 소스파일 이력이 적을 수 있다는 것을 확인 할 수 있었다.

5.5 Commit 일시와 결함의 상관분석

본 실험은 개발자가 어떠한 요일/시간에 Commit 을 했을 때 결함을 가장 많이 발생 시키는지 실험하였다. 앞서 말한 8300 개의 커밋 로그의 FixDate, openDate, 설명과 버그 리포트를 비교해 결함을 발생시켰던 소스코드파일을 찾아 실험을 진행하였으며, 결함을 발생시킨 Commit 의 "월", "요일", "시간"을 분석하였다. 실험결과 일반적인 평균 업무시간인 월요일에서 금요일에 균등하게 결함이 포함되어 있는 것을 확인하였으며, 업무시간인 9 시에서 16 시 사이에 결함이 균등하게 발생하는 것을 확인 할 수 있었다.

5.6 결함 위치를 식별하기 용이한 버그 리포트 형식 제안

앞서 버그 리포트 담긴 내용을 분석하였다. 본 장에서는 앞서 분석한 결과를 바탕으로 결함 효과적으로 식별할 수 있는 버그 리포트 형식을 제안한다. 최초 버그를 찾게되는 개발자/테스트들이 버그 리포트에 자세한 결함의 위치 정보를 작성한다면, 결함의 위치를 보다 쉽게 해결할 수 있다는 것을 앞서 실험한 결과에서 확인 할 수 있었다.

표 7 과 같이 제목에는 프로젝트명과 결함의 상태를 표시한다. 예를 들어 "Crash"또는"Interface"와 같은 단어로 결함을 수정하는 우선위를 결정할 수 있을 것이다. 설명에는 결함을 재연할 수 있는 자세한 시나리오와 Stack-Trace 정보, 의심되는 결함의 Class, Method 정보를 입력한다. 마지막으로 첨부파일에는 의심되는 결함의 로그파일정보와 인터페이스가 있는 경우 결함이 발생한 화면을 스크린 캡쳐를 이용해 이미지를 첨부해 결함을 빠르게 식별할 수 있도록 한다.

표 7 버그 리포트 형식 제안

제목	[Project Name] 결함의 상태
설명	결함의 상세 정보 1. 결함 재현 시나리오 2. Stack-Trace 정보 3. 의심되는 결함의 Class, Method 정보
첨부파일	Log-File, Screen-Shot

6. 결론

소프트웨어의 진화는 끊임 없이 되고 있으며, 그에 발맞춰 다양한 형태의 결함들이 발생되고 있다.

출시한 소프트웨어 제품에 대해 개발팀은 버그/이슈 관리 시스템을 사용자에게 제공하고 사용자나 테스트는 자신이 사용하면서 발견한 이슈 내용을 시스템에 등록한다. 이러한 버그 리포트는 결함 위치 추적으로 중요한 역할을 한다. 버그 해결 시간과 비용을 단축하기 위한 많은 연구가 진행되고 있다.

본 논문에서 정보검색 기반 결함위치 추적 연구에서 사용되는 요소들인 버그 리포트, 커밋 로그, 개발 문서의 영향력에 대해 실험적 연구를 통해 앞으로 정보검색 기반 결함위치 추적 연구에서 사용되는 요소들에 정리하였으며, 버그 리포트 내에 품질에 따라 결함을 찾을 확률이 높고 낮음에 대해 알게 되었다.

제 4 장의 5 가지의 다양한 실험을 통해 정보검색 기반 결함 위치 추적 연구에서 사용되지 않았던 요소들인 개발 문서와 버그 리포트의 댓글의 효과를 확인 할 수 있었다.

7. 향후 연구

이후 연구로, 우리는 다음과 같은 분야에서 연구를 진행하려 한다. 이미지 검색을 추가한 정보검색 기반 결함위치 추적을 진행한다. 본 실험을 통해 버그 리포트에 작성되는 텍스트들에 대한 정보의 영향력에 대한 확인을 하였다. 하지만 버그 리포트 내에 첨부파일에 결함이 발생한 시점에 대한 "화면캡쳐" 정보는 첨부되기도 한다. 이미지를 통해 결함 위치 추적의 정확도 향상에 기여하는 연구를 계획 중이다.

정보검색 기반 결함위치 추적 연구는 Class 레벨에서의 활발한 연구가 진행 되어 왔다. 향후에는 Method 레벨에서의 위치추적을 위해 버그 리포트, 소스코드파일, 커밋 로그를 통해 결함을 찾고자 한다. 또한 결함 위치 추적의 해결시간을 줄이기 위한 연구를 계획 중이다.

Acknowledgment

이 논문은 2015 년도 정부(교육과학기술부)의 재원으로 한국연구재단-차세대정보컴퓨팅개발사업의 지원을 받아 수행된 연구임(No. 2015045358)

참고문헌

[1] Ian Sommervile, Software Engineering, 10th Edition, 2015

[2] CD Manning, P Raghavan and H Schütze, "Introduction to information Retrieval", 2008

[3] Binkley, David, and Dawn Lawrie, "Information retrieval applications in

software maintenance and evolution.", Encyclopedia of Software Engineering, 2009
[4] Zhou, Jian, Hongyu Zhang, and David Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports", International Conference on. IEEE, 2012
[5] RK Saha, M Lease, S Khurshid, "Improving bug localization using structured information retrieval", Automated Software Engineering International Conference on. IEEE, 2013
[6] Wang, Shaowei, and David Lo, "Version history, similar report, and structure: Putting them together for improved bug localization", International Conference on Program Comprehension, 2014
[7] Davies, Sean, and Marc Roper, "Bug localisation through diverse sources of information", Software Reliability Engineering Workshops, 2013
[8] CP Wong, Y Xiong, H Zhang, "Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis", Software Maintenance and Evolution, 2014
[9] L Moreno, JJ Treadway, A Marcus, "On the Use of Stack Traces to Improve Text Retrieval-Based Bug Localization, Software Maintenance and Evolution, 2014
[10] Klaus Changsun Youm, June Ahn, Jeongho Kim, Eunseok Lee, "Bug Localization Based on Code Change Histories and Bug Reports", Asia-Pacific Software Engineering Conference, 2015
[11] Sisman, Bunyamin, and Avinash C. Kak, "Incorporating version histories in information retrieval based bug localization", *Mining Software Repositories, 2012*
[12] Tantithamthavorn, Chakkrit, "Mining A change history to quickly identify bug locations: A case study of the Eclipse project", Software Reliability Engineering Workshops, 2013
[13] GitHub, https://github.com/
[14] JavaDoc, http://www.oracle.com/technetwork/articles/java/index-jsp-135444.html
[15] Doxygen, http://www.stack.nl/~dimitri/doxygen/
[16] AspectJ Project, https://eclipse.org/aspectj/
[17] SWT Project, https://www.eclipse.org/swt/
[18] ZXing Project, https://github.com/zxing/

테스트케이스 재구성을 통한 결함위치식별 성능개선

김정호, 이은석

성균관대학교 정보통신공학부
경기도 수원시 장안구 서부로 2066
{jeonghodot, leees}@skku.edu

요약: 결함의 위치를 찾기 위해 가장 중요하게 활용되는 요소는 실패 테스트케이스가 지나간 정보이다. 결함 위치 식별 시, 이 정보 활용의 유의미함은 기확인 하였으며, 실제로 다수 결함위치식별 기법들이 이 정보를 이미 활용하고있다. 본 본문에서는 실패 테스트케이스의 가중치를 증폭시켜 보다 정확한 결함 위치 식별을 도와줄 수 있는 테스트케이스 그룹화기법을 제안한다. 또한, 테스트 효율성을 고려해 테스트케이스 필터링 기법을 제안하고 이들을 76 개의 알고리즘에 적용해 실효성을 검증한다.

핵심어: 테스트케이스 그룹화, 필터링, 결함 위치 식별, 스펙트럼, 실행 경로 추적.

1. 서론

대부분 스펙트럼 기반 기법은 의심도 계산 수식에서 실패 테스트케이스를 강조하지만 실패, 성공 테스트케이스를 한번에 적용 하는 것으로는 실패 테스트케이스의 고유 특성을 반영하기 어렵다. 이러한 이유로 인해, 새로운 스펙트럼 기반 기법 제안은 더 이상 무의미하고, 테스트케이스 재구성 기법 제안을 통해 테스트 정확성 및 효율성을 향상시키고자 한다. 본 논문에서는 실패 테스트케이스의 가중치를 증폭시켜 보다 정확한 결함 위치 식별을 도와줄 수 있는 테스트케이스 그룹화 기법을 제안한다. 또한, 테스트의 효율성과 정확성을 고려해 테스트케이스 필터링 기법을 제안한다.

2. 테스트케이스 재구성 기법

2.1 테스트케이스 그룹화 기법

결함위치식별 정확성 향상을 위해 실패 테스트케이스의 수만큼 그룹화하여 개별 계산을 먼저하고 통합하는 테스트케이스 그룹화기법을 제안한다.

NStmt = the Number of Statements
NFTC = the Number of Failed Test cases
NPTC = the Number of Passed Test cases
$S = \{s_1, s_2, s_3 \dots\ s_n\}$ = *Statements*
$F = \{f_1, f_2, f_3 \dots\ f_n\}$ = *Failed test cases*
$P = \{p_1, p_2, p_3 \dots\ p_n\}$ = *Passed test cases*
1: For (i=1, until NStmt){
2: ***For (j=1, until NFTC){***
3: $N_{cf} = Count.coveredTC(Count.failedTC(s_i, f_j, P))$
4: $N_{uf} = Count.uncoveredTC(Count.failedTC(s_i, f_j, P))$
5: $N_{cp} = Count.coveredTC(Count.passedTC(s_i, f_j, P))$
6: $N_{up} = Count.uncoveredTC(Count.passedTC(s_i, f_j, P))$
7: $s_{i_{suspicious\ ratio}}$=Similarity$\left(N_{cf}, N_{uf}, N_{cp}, N_{up}\right)$ *or*
Inverse(Distance$\left(N_{cf}, N_{uf}, N_{cp}, N_{up}\right)$)
8: }
9: $\boldsymbol{s_{i_{suspicious\ ratio}}}$**=Accumulate**$(s_{i_{suspicious\ ratio}})$
10: }
11:$s_{i_{ranking}}$=Rank$(S, s_i, s_{i_{suspicious\ ratio}})$

알고리즘 1 테스트케이스 그룹화 기법

알고리즘 1 은 테스트케이스 그룹화 기법을 설명한다. 기존 스펙트럼 기반 기법에 실패 테스트케이스의 수만큼 의심도를 반복 계산(2 행), 누적(9 행)하는 함수를 추가한다. 실패 테스트케이스 수만큼 그룹을 생성하고 각 그룹을 하나의 실패 테스트케이스와 모든 성공 테스트케이스로 구성한다. 모든 그룹의 의심도를 개별 계산하고 이들을 모두 합산해 최종 의심도를 구한다.

2.2 테스트케이스 필터링 기법

테스트의 정확성과 효율성 향상을 위한 테스트케이스 필터링 기법을 제안한다. 이는 전체 성공 테스트케이스 중, 의심도 정제를 도와줄 일부를 선택하는 기법이다.

NStmt = the Number of Statements
NFTC = the Number of Failed Test cases
NPTC = the Number of Passed Test cases

$S = \{s_1, s_2, s_3 \dots\ s_n\}$ = *Statements*
$F = \{f_1, f_2, f_3 \dots\ f_n\}$ = *Failed test cases*
$P = \{p_1, p_2, p_3 \dots\ p_n\}$ = *Passed test cases*

*1: **FilterStrategy = Distinct***
P = FilterStrategy({$p_1, p_2, p_3 \cdots p_n$})
2: For (i=1, until NStmt){
3: For (j=1, until NFTC){
4: $N_{cf} = Count.coveredTC(Count.failedTC(s_i, f_j, P))$
5: $N_{uf} = Count.uncoveredTC(Count.failedTC(s_i, f_j, P))$
6: $N_{cp} = Count.coveredTC(Count.passedTC(s_i, f_j, P))$
7: $N_{up} = Count.uncoveredTC(Count.passedTC(s_i, f_j, P))$
8: $s_{i_{suspicious\ ratio}}$=Similarity$(N_{cf}, N_{uf}, N_{cp}, N_{up})$ *or*
Inverse(Distance$(N_{cf}, N_{uf}, N_{cp}, N_{up})$)
9: }
10: $s_{i_{suspicious\ ratio}}$=Accumulate($s_{i_{suspicious\ ratio}}$)
11: }
12:$s_{i_{ranking}}$=Rank($S, s_i, s_{i_{suspicious\ ratio}}$)

알고리즘 2 테스트케이스 필터링 기법

알고리즘 2은 테스트케이스 그룹화 기법에 필터 함수를 추가한다. 필터링 전략은 중복되는 스펙트럼을 갖는 테스트케이스를 제외(1행)하는 방법을 선택한다.

3. 실험

3.1 테스트케이스 재구성 기법

테스트케이스 재구성 기법은 필터링 전략을 추가하여 확장이 가능하다. 전략에 따라 테스트 정확성과 효율성이 변하므로 전략 설정이 중요하다. 본 논문에서는 다음 3가지 필터 전략을 적용해 실험을 수행한다.

1. 중복된 성공 테스트케이스
2. 중복된 실패 테스트케이스
3. 중복된 실패, 성공 테스트케이스

표 1 테스트케이스 그룹화 및 필터링 기법 성능 비교

		Test case grouping + filtering			
Filtering strategy		Grouping only	Distinct PTC	Distinct FTC	Distinct F, PTC
Number of Outperformed methods		49	64	52	66
Outperformed methods		64.47%	84.21%	68.42%	86.84%
Number of Outperformed methods including no change		51	66	52	66
Outperformed methods including no change		67.11%	86.84%	68.42%	86.84%
Accuracy	Outperformed Median	2.00%	20.81%	2.13%	22.61%
	Outperformed Average	-12.32%	5.02%	-11.78%	6.99%
	Outperformed Max	51.77%	60.51%	51.99%	60.51%
	Outperformed Min	-537.88%	-456.88%	-540.87%	-452.91%
	Outperformed Deviation	76.85	65.97	77.49	65.19
Efficiency	Selected test cases by filtering	359,826 /359,826	105,959 /359,826	354,474 /359,826	100,607 /359,826
	Efficiency of test case filtering	0%	70.55%	1.49%	72.04%

4. 결론

본 논문의 실험 결과를 기반으로 새로운 스펙트럼 기반 기법 제안은 더 이상 무의미하고, 테스트케이스 재구성 기법을 통해 테스트 성능을 제고할 수 있다. 즉, 테스트케이스 재구성 기법만으로도 기존 최우수 기법들 보다 정확하고 효율적인 테스트가 가능하다. 최근 제안된 성능 우수 기법인 Naish1, 2[1], GP01-30[2], Heuristic1-3[3], Wong1-3[4], D star[5], Lex_Ochiai[6], 본 연구팀에서 제안한 SEM1-3[7]과 비교를 통해 성능을 확인하였다.

Acknowledgement

이 논문은 2015년도 정부(교육과학기술부)의 재원으로 한국연구재단-차세대정보컴퓨팅개발사업의 지원을 받아 수행된 연구임(No. 2015045358).

참고문헌

[1] Naish, Lee, Hua Jie Lee, and Kotagiri Ramamohanarao, "A model for spectra-based software diagnosis." ACM Transactions on software engineering and methodology (TOSEM) 20.3, Article 11, 2011.

[2] Yoo, Shin, "Evolving human competitive spectra-based fault localisation techniques." Search Based Software Engineering. Springer Berlin Heidelberg, pp.244-258, 2012.

[3] Wong, W. Eric, Vidroha Debroy, and Byoungju Choi, "A family of code coverage-based heuristics for effective fault localization." Journal of Systems and Software 83.2, pp.188-208, 2010.

[4] Wong, W. Eric, and Yu Qi, "Effective program debugging based on execution slices and inter-block data dependency." Journal of Systems and Software 79.7, pp.891-903, 2006.

[5] Wong, W. Eric, Debroy, V., Ruizhi Gao, and Yihao Li, "The dstar method for effective software fault localization." Reliability, IEEE Transactions on 63.1, pp.290-308, 2014.

[6] David Landsberg, Hana Chockler, Daniel Kroening, and Matt Lewis, "Evaluation of Measures for Statistical Fault Localisation and an Optimising Scheme." Fundamental Approaches to Software Engineering. Springer Berlin Heidelberg, pp.115-129, 2015.

[7] Jeongho Kim, Jonghee Park and Eunseok Lee, "A New Spectrum-based Fault Localization with the Technique of Test Case Optimization", Journal of Information Science and Engineering (JISE), Vol. 32, No. 1 Jan, pp.177-196, 2016.

효과적인 웹 애플리케이션 테스팅을 위한 DOM 관련 동적 데이터 흐름 분석 방법

김지훈, 고인영

한국과학기술원 전산학부
대전 유성구 대학로 291
{jihoon.kim, iko}@kaist.ac.kr

요약: 동적인 웹 애플리케이션 구축을 위해서는 클라이언트 쪽의 자바스크립트(JavaScript) 프로그램과 문서 객체 모델(Document Object Model, DOM) 노드 간의 빈번한 상호작용이 실행시간 중에 필요하다. 그런데 이러한 자바스크립트 프로그램과 DOM 노드의 상호작용은 일반적으로 오류 발생가능성을 높이고, 웹 애플리케이션을 분석하고 테스트하기 어렵게 한다. 본 연구에서는 자바스크립트 프로그램과 DOM 노드의 상호작용 과정에서 발생하는 데이터 흐름을 동적으로 분석하는 방법을 제안한다. 본 기법은 웹 애플리케이션을 수정하여 수행된 구문으로부터 데이터를 수집하여 그 흐름을 파악하고, DOM 과 관련된 데이터 흐름을 추출하여 DOM 노드와 상호작용하는 변수를 파악할 수 있도록 한다. 개발된 DOM 관련 데이터 흐름 분석 방법을 실제 웹 애플리케이션에 적용해 봄으로써 제안한 방법의 유용성을 보였다.

핵심어: 웹 애플리케이션, 동적인 데이터 흐름 분석, 문서 객체 모델

1. 서 론

웹 애플리케이션은 시간과 공간의 제약이 없는 웹의 장점을 바탕으로 활발히 사용되고 있다. 이에 따라, 웹 애플리케이션의 수요가 증가하여 다양한 사용자의 요구사항과 복잡한 기능을 갖는 시스템이 되고 있다. 그리고 웹 애플리케이션의 규모가 커지고 복잡해지면서 그 품질의 중요성은 높아지고 있다.

웹 애플리케이션 개발시에는 HTML 과 CSS 로 전체적인 웹페이지의 레이아웃 등을 정의하고, 스크립트 언어를 사용해 웹페이지 내 각 요소들의 동작을 정의하거나 사용자의 입력을 동적으로 반영한다.

자바스크립트(JavaScript)는 웹 애플리케이션을 개발하는데 가장 많이 사용되고 있는 언어이다[22]. 자바스크립트는 클라이언트(Client) 쪽에서 실행되며 사용자와의 더 나은 상호작용을 위해 문서 객체 모델(Document Object Model, DOM)[1]을 동적으로 조작한다. DOM 은 트리 형태로 구성되어있고, 각 노드를 제어하기 위한 Application Program Interface(API)를 제공한다. DOM API 는 XPath 나 DOM 노드의 위치지정자(Web Locator)를 사용하여 어떤 노드에 접근할지 지정한다. 위치지정자의 주체인 DOM 노드가 존재하면 그것을 사용하여 값을 읽거나 할당할 수 있고, 존재하지 않는다면 예외상황(Exception) 이 발생한다. 이와 같은 오류를 뒤늦게 발견하면 그 원인을 찾고 프로그램을 수정하는데 어려움을 겪게된다.

Ocariza et al. 에 의하면 자바스크립트 프로그램과 DOM 의 상호 작용과 관련된 오류가 웹 애플리케이션 오류의 65%에 달하며, 강한 영향력을 가지는 자바스크립트 오류의 80% 이상은 DOM 과 관련되어있다고 알려져 있다[4]. 그러나 매우 동적인 자바스크립트의 특성과 DOM API 의 다양성으로 인해 DOM 과 관련된 오류를 분석하기 힘들고 테스트하기 힘들다[2, 3]. 따라서, 웹 애플리케이션에서 자바스크립트 프로그램과 DOM 노드간의 상호작용과 관련한 분석과 테스트가 필요하다.

웹 애플리케이션 테스트 기법에 대한 연구는 기존의 C/C++, Java 등의 애플리케이션을 대상으로 하는 테스트 기법을 웹 애플리케이션에 적용하거나 웹 애플리케이션의 특성을 고려한 테스트 기법을 제시하고 있다. 그러나 대부분의 연구는 테스트 케이스생성 시 입력 값으로 DOM 노드를 사용하지 않거나 DOM 구조를 단순화시켜 고려하며 사용자의 이벤트를 생성하거나 입력 값을 생성하는데 초점이 맞춰져 있다.

본 연구의 주 목적은 자바스크립트 기반 웹 애플리케이션을 위한 데이터 흐름(Data Flow) 을 분석하여 DOM 과 관련된 데이터 흐름 관계를 추출하는 것이다. 프로그램의 실행 흐름에 따라 처리되는 데이터에 대한 정보는 테스트 케이스를 생성하는데 유용하다. 이를 통해 자바스크립트 기반의 웹 애플리케이션을 테스트하는 방법을 만들 수 있을 것이다. 본 연구에서는 Samegame, ToDoList 등의 2 개의 공개된 웹 애플리케이션을 대상으로 테스트를 통해 생성한 데

이터 흐름을 분석하여 DOM 과 관련된 데이터 흐름 관계를 추출하였다.

2 절에서는 제안 기법의 토대가 되는 DOM 과 자바스크립트의 상호작용과 데이터 흐름 분석에 대해 간략히 설명한다. 3 절에서는 웹 애플리케이션 테스트 기법과 동적인 데이터 흐름 테스트 기법에 관한 관련 연구를 소개하고 있고 4 절에서는 접근 방법을 설명하고 5 절에서는 접근 방법을 구현하여 적용해본 결과를 설명한다. 마지막으로 6 절에서는 결론과 향후 연구에 대해 설명한다.

2. 배경 지식

2.1 DOM 과 자바스크립트의 상호작용

DOM 은 HTML, XML 등의 문서를 위한 프로그래밍 인터페이스이다. DOM 은 트리 형태로 표현되며 각 노드들을 객체로 추상화하여 언어에 관계 없이 각 노드들을 다룰 수 있다. 웹 애플리케이션에서 HTML 은 웹 페이지가 로드 될 때 브라우저는 웹페이지의 레이아웃 등이 정의되도록 한다[24]. 웹 애플리케이션에서 DOM 은 스크립트언어로 제어할 수 있다. 웹 애플리케이션의 클라이언트 쪽은 DOM 을 제어하기 위해서 일반적으로 자바스크립트를 사용한다. 개발자는 DOM 을 제어할 때, XPath 나 위치지정자를 사용하여 제어할 DOM 노드를 지정한다.

```
1.  <html>
2.  <head> …. </head>
3.  <body>
4.      <div class = “container”>
5.      …….
6.      <li><input type=”checkbox”>
7.        <label> Pay Bills </label>
8.        <input type=”text”>
9.        <button class=”edit” onclick=”editTask()”>
                  Edit</button>
10.       <button class=”delete”>Delete </button>
11.     </li>
12.     ……
13. </body></html>
14. <script>
15. function editTask() {
16.   var listItem = this.parentNode;
17.   var editInput = listItem.querySelector(
          ‘input[type=text’];);
18.   var containsClass =
          listItem.classList.contains(‘edit’);
19.   if(containsClass) {
20.     label.innerText = editInput.value;
21.   } else {
22.     editInput.value = label.innerText;
23.   }
24.   listItem.classList.toggle(‘edit’);
25.   return listItem;
26. };
27. </script>
```

그림 1. TodoList 예제 코드

그림 1 은 DOM 노드들과 자바스크립트 프로그램의 상호작용을 나타내는 TodoList 웹 애플리케이션의 코드이다. 1 번째 줄부터 13 번째 줄은 HTML 로 DOM 노드들을 표현하고 15 번째 줄부터 26 번째 줄은 사용자의 이벤트에 의해 수행될 자바스크립트 코드이다. 예제에서 ‘editTask’ 함수는 9 번째 줄에 정의된 버튼이 눌러지면 호출 되며, 그 실행 흐름은 다음과 같다. 16 번째 줄은 ‘editTask’ 함수를 호출한 DOM 노드의 부모노드를 ‘listItem’ 변수가 참조하게 한다. 예제에서 ‘listItem’은 ‘<li>’ DOM 노드를 참조한다. 17 번째 줄은 ‘querySelector’ API 에 ‘input=[type=text]’ 위치지정자를 사용하여 8 번째 줄에 해당하는 DOM 의 참조자 (reference)를 ‘editInput’변수에 할당한다. ‘editInput’ 변수를 을 사용하여 20 번째 줄과 같이 다른 DOM 노드에 접근하거나 값을 할당할 수 있다. 18 번째 줄은 ‘listItem’의 자식노드(child) 중 ‘edit’이라는 클래스(class)를 가지고 있는지 확인하고, 클래스 존재의 유무에 따라 19 번째 줄의 분기문에 의해 DOM 노드에 할당되는 값이 바뀔 수 있다.

그림 1 의 예제와 같이 DOM 노드의 복잡한 구조와 DOM 노드에 할당된 값이 자바스크립트와의 상호작용으로 제어된다. DOM 에 접근하기 위해서 API 를 사용할 때, 올바른 Xpath 나 위치 지정자가 필요하며 DOM 노드의 관계를 사용하여 다른 DOM 노드에 추가적인 제어가 가능하다. 또한, DOM 노드의 참조자가 분기문에 사용되어 프로그램의 실행 흐름을 다양하게 만들 수 있다. 따라서 복잡한 구조의 DOM 노드의 관계와 DOM 노드에 할당된 값이 프로그램의 실행 흐름에 영향을 끼치기 때문에 웹 애플리케이션을 테스팅할 때 DOM 노드들과 자바스크립트 프로그램의 상호작용을 파악하는 것이 중요하다.

2.2 데이터 흐름 분석

데이터 흐름 분석은 주로 컴파일러가 프로그램 수행속도와 코드 크기의 최적화를 위해 사용해 왔으며 다양한 분석기법이 연구되었다. 데이터 분석을 하기 위해서 일반적으로 Control Flow Graph(CFG)가 필요하다. CFG 는 프로그램이 실행중에 수행될 수 있는 모든 경로를 그래프로 표현한 것으로최적화와 정적 분석에서 중요한 역할을 한다. CFG 에서 각 노드는 기본 블록(Basic Block)을 표현하고, 각 간선은 제어 흐름에서의 분기를 나타낸다. 대부분의 경우 흐름 그래프로 진입하는 블록(Entry Block)과 종료 블록(Exit

Block)을 지닌다.

데이터 흐름 분석에 대해서는 다양한 연구들이 수행 되었지만, 이 절에서는 본 연구와 관련이 있는 Reaching Definition 분석 방법에 대해 설명한다[25]. Reaching Definition 분석은 각 프로그램 지점에서 변수가 재정의 되지 않고 프로그램의 어느 지점까지 도달 가능한지 분석하는 것이다. 도달 가능성을 분석하기 위해 몇 가지 필요한 정의와 데이터 흐름 방정식은 다음과 같다.

Def: 할당문에 의해 정의된 변수
K: 이미 정의된 변수 중, 재정의된 변수의 집합
GEN(X): 기본 블록 X 내에서 정의된 변수의 집합
KILL(X): 기본 블록 X 내에서 재정의된 변수의 집합
IN(X): 기본 블록 X 의 시작지점에 도달 가능한 정의된 변수의 집합
OUT(X): 기본 블록 X 의 종료지점에 도달가능한 정의된 변수의 집합

$$Ops = \{Def\}$$
$$K = \{Def\ is\ overwritten\}$$
$$GEN(X) = OPS \cup (GEN(X) - K)$$
$$KILL(X) = K \cup (KILL(X) - Ops)$$
$$IN(X) = \bigcup_{p \in pred(X)} OUT(p)$$
$$OUT(X) = GEN(X) \cup (IN(X) - KILL(X))$$

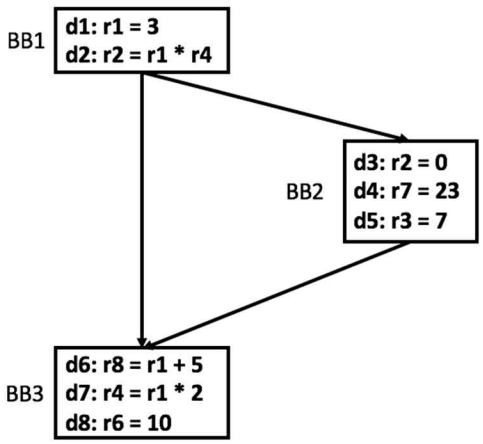


그림 2. Reachding Definition 분석을 위한 CFG

그림 2 는 Reaching Definition 을 설명하기 위한 CFG 로 각 노드는 기본블록을 나타내고 기본블록마다 정의된 변수들이 표시되어있다. 우선, 각 기본블록마다 GEN 과 KILL 을 구하고, 각 기본블록에 대해 IN 과 OUT 을 파악한다. 기본 블록 BB1 에서는 변수 r1, r2 가 정의된다. GEN(BB1)은 r1, r2 이고 KILL(BB1)은 재정의된 변수가 없기 때문에 존재하지 않는다. 이와 같은 과정을 BB2, BB3 에도 반복적으로 적용하여 모든 기본 블록에 대해 GEN 과 KILL 을 구한 후, 모든 기본 블록에 대해 IN 과 OUT 을 구한다. BB1 의 경우, 선행자가 없어 IN(BB1)은 존재하지 않으나 OUT(BB1)은 방정식에 의해 r1, r2 가 된다.

Reaching Definition 분석은 상수 전파(Constant Propagation), 루프 불변 코드 이동(Loop Invariant Code Motion) 등의 최적화 기법에 활용되며 데이터 흐름 테스팅에 경로 선택에 활용될 수 있다. 본 논문에서는 Reaching Definition 분석을 웹 애플리케이션에 적용하여 데이터 흐름 분석을 통해 DOM 과 관련된 데이터 흐름 관계 추출한다.

3. 관련 연구

3.1 웹 애플리케이션 테스트 기법

최근 웹 애플리케이션 테스트 기법에 대한 연구가 활발히 진행되고 있다. Artzi et al. 은 웹 애플리케이션을 대상으로 자동화된 테스트 프레임워크인 Artemis 를 제안하였다[5]. Artemis 는 임의의 이벤트 흐름을 생성하고 실행 한 후, 피드백을 기반으로 새로운 테스트 입력을 생성한다. 또한, 이벤트 흐름의 우선순위를 조절할 수 있는 기능을 제공한다. 그러나 랜덤 테스팅에 기반을 두고 있어 랜덤 테스팅과 비슷한 성능을 제공한다.

Saxena et al. 은 Kudzu 라는 자동화된 테스트 도구를 제안하였다[6]. Kudzu 는 임의의 테스트 입력을 생성하여 웹 애플리케이션의 이벤트 조합을 파악하고 문자열 연산이 빈번한 웹 애플리케이션의 특성을 고려한 복잡한 문자열의 심볼릭 실행(Symbolic Execution)을 통해 입력 값을 확인할 수 있다. 그러나 복잡한 문자열의 심볼릭 실행을 위해서는 시간이 오래 걸리는 단점이 있다.

Sen et al. 은 Concolic 테스트 및 동적 분석을 위한 Jalangi 라는 자동화 도구를 제안하였다[7]. Jalangi 는 선택적 기록-재생(Selective Record-Replay) 기법을 사용하여 프로그램의 수행을 쉐도우 값(Shadow value)과 함께 기록하고 재생한다. 쉐도우 값은 심볼릭 변수나 변수의 부가적인 정보를 포함하고 있어서 프로그램 분석에 도움을 준다. 이와 같은 기법을 사용하여 Jalangi 는 'null'이나 'undefined' 와 같은 일반적인 오류를 찾을 수 있다. 그러나 웹 애플리케이션보다는 자바스크립트 프로그램 자체의 분석과 테스트에 주로 초점이 맞춰져 있다.

Li et al. 은 클라이언트 쪽 웹 애플리케이션의 심볼릭 실행을 통해 자동으로 입력값과 이벤트를 생성하는 SymJS 도구를 제안하였다[23]. SymJS 는 자바스크립트와 이벤트 뿐만 아니라 DOM 과 브라우저 API 를 모델링하여 심볼릭 실행을 수행할 수 있다. 그러나 SymJS 는 DOM 을 단순히 정수나 문자열로 간주하여 DOM 이 가지는 복잡한 구조를 표현할 수 없다는 단점이 있다.

위의 연구들은 기존 테스트 기법을 자바스크립트 기반 웹 애플리케이션에 적용한 사례들 이다. 그러나 대부분 자바스크립트 프로그램 자체를 테스트하거나 이벤트를 생성하는데 초점을 두었고 DOM 의 복잡한 구조를 단순화하여 테스팅을 수행한다.

Liu et al. 은 데이터 흐름 테스트 기법을 웹 애플리케이션에 적용한 Web Application Test Model (WATM)을 소개 하였다[15]. Liu et al. 은 함수 수준부터 애플리케이션 수준까지 웹 애플리케이션을 다섯 단계로 분석했다. 그러나 페이지 단위의 링크를 분석 대상으로 삼아 최근 복잡한 웹 애플리케이션의 내부 실행 흐름을 파악하기 어렵기 때문에 이와 관련된 테스트 케이스 생성이 어렵다.

Jensen et al. 은 웹 애플리케이션의 서버와 클라이언트 사이의 상호작용을 테스트하기 위한 기법을 제안하였다[8]. Jensen et al. 은 서버 인터페이스 명세를 만들고 서버와 클라이언트 사이의 커뮤니케이션 패턴을 학습하여 모의 서버(Mock Server)를 만든다. 그러나 이 방법은 웹 애플리케이션의 내부 실행 흐름보다는 서버와 클라이언트의 커뮤니케이션을 테스트하는데 초점이 맞춰져 있다.

Mesbah et al. 은 크롤링(Crawling) 기반의 자동화된 테스트 도구인 Crawljax/ATUSA 를 제안하였다[9, 10]. Crawljax/ATUSA 는 웹 애플리케이션을 크롤링하여 임의로 이벤트와 관련된 DOM 노드를 찾아 이벤트를 발생시키거나 사용자로부터 주어진 입력을 사용하여 State-Flow-Graph (SFG)를 생성한다. SFG 는 웹 애플리케이션의 DOM 트리의 상태를 노드로, DOM 노드를 변경하는 이벤트를 간선으로 정의하여 그래프로 나타낸 것이다. Mirshokraie et al. 은 이를 회귀 검사[11], 뮤테이션 테스트 [12], 단위 테스트 케이스 생성 도구 [13] 등으로 확장하였다. 그러나 상태 폭발 문제(State Explosion Problem)를 야기할 수 있어 제한된 상태 범위 내에서만 테스팅이 가능하다.

Fard et al. 은 웹 애플리케이션의 자동화된 단위 테스트를 위해 필요한 복잡한 구조의 DOM 구조를 생성해주는 Confix 라는 도구를 제안하였다[14]. Confix 는 Concolic 테스팅 기법을 사용하여 DOM 의 복잡한 구조를 추론하고 구축한다. 그러나 DOM 구조를 이루는 각 DOM 노드들에 할당될 입력 값을 생성하지 못해서 완전한 테스트 케이스를 생성하지 못하는 한계가 있다.

3.2 동적인 데이터 흐름 분석

데이터 흐름 테스트 기법에 대한 연구는 꾸준히 있어왔다[16-18]. 최근에는 기존의 정적인 데이터 흐름 분석 기법으로 생성된 데이터 흐름의 정확도에 한계가 있어 테스팅에 어려움이 있었다. 이를 해소하기 위한 동적 데이터 흐름 분석 기법을 활용하여 테스팅에 적용한 연구가 진행되었다.

Denaro et al. 은 Java 프로그램를 대상의 동적인 데이터 흐름 테스트를 위한 Dynamic Reaching Definition Analysis(DReaDs) 라는 기법을 제안하였다[19]. Denaro et al. 은 정적 데이터 흐름 분석의 확장성을 높이기 위해 데이터 흐름을 추상화 하는 것이 정확한 분석에 한계가 있음을 언급하고 있다. 따라서, DreaDs 는 정확한 데이터 흐름 관계를 파악하기 위해 프로그램의 실행을 모니터링하고 동적으로 데이터 흐름 관계를 파악하였다.

Denaro 등은 DReaDs 를 확장하여 DynaFlow 라는 도구를 제안하였다[20]. DynaFlow 는 주어진 테스트 케이스로부터 DReaDs 이 생성한 데이터 흐름 관계를 사용하여 추가적인 테스트 케이스를 생성한다. 그 결과, 정적인 데이터 흐름 기법을 사용했을 때 보다 많은 데이터 흐름 관계를 찾아낼 수 있고 뮤테이션 기법을 사용하여 테스트 성능을 높일 수 있다.

본 연구는 Denaro 등의 연구를 웹 애플리케이션으로 확장하여 DOM 과 관련된 동적 데이터 흐름 관계를 생성한다.

4. DOM 관련 동적인 데이터 흐름 분석 기법

본 논문에서는 동적인 데이터 흐름 분석을 통해 웹 애플리케이션을 위한 DOM 기반의 데이터 흐름을 파악하는 방법을 제안한다. 이 방법은 (1) DOM 노드들과 상호작용하는 정보를 얻기 위한 프로그램 수정, (2) 웹 브라우저에서 웹 애플리케이션을 실행하고 테스트 케이스를 수행하는 과정, (3) 수집된 결과를 분석하여 데이터 흐름 관계를 파악하는 과정으로 이루어져 있다. 한다. 다음 각 절에서는 이러한 각 단계에 대해 상세히 설명한다.

4.1 프로그램 수정

프로그램 수정(Program Instrumentation)은 DOM 과 자바스크립트 코드의 상호작용하는 정보를 얻기 위해 기존의 코드에 분석에 필요한 코드를 추가하는 것을 말한다. 본 논문에서는 Fard et al. [14]이 사용한 프로그램 수정 방법에 따라 래퍼(Wrapper) 함수를 정의하여 기존 프로그램의 기능에 영향을 주지 않도록 하였다.

프로그램 수정은 자바스크립트 코드를 입력으로 받아 Rhino[21]라는 파서(Parser)를 사용하여 추상적 구문 트리(Abstract Syntax Tree, AST)로 변환한 후 이루어진다. AST 는 가장 소스 코드와 가까운 표현으로 소스 단계의 분석 및 수정에 용이하다. 그림 3 은 그림 1 의 ‘editTask’함수를 수정한 자바스크립트 코드의 예시이다.

```
1.  function editTask() {
2.      wrapper(“functioncall”,
            "console.log('Edit task...')",
            ["Edit task..."], ['Edit task...'], "editTask",
            console.log('Edit task...'));
3.      var listItem = wrapper(0,
            "var listItem = this.parentNode",
            [""], [], "editTask", this.parentNode);
4.      var editInput = wrapper(“varinit”, "var editInput
            =listItem.querySelector('input[type=text]')",
            [""], [], "editTask",
            listItem.querySelector('input[type=text]'));
5.      var label = wrapper(“varinit”, "var label =
            listItem.querySelector('label')",
            [""], [], "editTask",
            listItem.querySelector('label'));
6.                          ……
7.      wrapper(“functioncall”,
            "listItem.classList.toggle('editMode')",
            ["editMode"], ['editMode'], "editTask",
            listItem.classList.toggle('editMode'));
8.  }
```

그림 3. 수정된 자바스크립트 코드 예제

프로그램 수정을 통해 데이터 흐름 관계를 파악하기 위해 각 구문으로부터 얻어야 할 정보는 구문의 종류, 사용된 변수의 이름, 사용된 변수의 값, 수행되는 구문이 속한 함수 이름, 실제 수행될 구문이다. 수정해야할 구문의 종류는 다음과 같다.

1) 함수 호출(function call)
2) 배정문(assignment statement)
3) 변수 초기화(variable initialization)
4) 조건문(conditional expression)
5) 반환문(return statement)
6)

DOM 노드에 대한 값 변경이나 제어는 ‘getElementById’, ‘querySelector’, ‘appendChild’, ‘getElementsByTagName’ 등과 같은 API 호출을 통해 수행된다. 그림 1 의 17 번째 줄과 같이 인자로 input[type=’text’]와 같은 위치지정자를 실행 인자로 주어 DOM 노드에 접근할 수 있다. DOM 노드가 사용되는 API 함수 호출 구문은 데이터 흐름 관계를 파악하기 위해 중요한 정보이다. 배정문 및 변수 초기화는 DOM 노드에 할당 된 값을 사용되거나 DOM 노드나 할당된 값이 자바스크립트 별칭으로 사용될 수 있어 데이터 흐름에 중요한 역할을 한다. 조건문은 프로그램의 실행 경로를 달라 지게 할 수 있어 조건문의 결과에 따라 파악할 수 있는 데이터 흐름을 변화시킬 수 있다. 따라서 다양한 데이터 흐름 관계가 도출 될 수 있다. 반환문 또한 프로그램의 실행 흐름에 영향을 주기때문에 중요한 요소가 된다.

```
1.  var trace = [];
2.  function wrapper(type, stmt2str, vals, vars, scope,
    stmt) {
3.          trace.push({type:type,stmt2str:stmt2str, ,…} );
4.  }
5.  function getTracingData() {
6.          return trace;
7.  }
```

그림 4. 데이터 흐름 테스트에 필요한 정보를 모으고 반환하는 예제 코드

수정이 된 프로그램은 수행되면서 분석에 필요한 정보를 변수에 저장한다. 그림 4 의 2-3 번 줄은 프로그램 수행중 실행된 구문에 대한 정보를 1 번 줄에서 정의한 변수에 저장하는 함수이다. 5 번 줄은 프로그램 수행시 모은 정보를 반환하는 함수이다. 이는 테스팅 과정에서 수집된 프로그램 수행 결과를 전달하는데 사용된다.

4.2 테스팅

테스팅은 수정된 웹 애플리케이션을 웹 브라우저에서 실행하여 데이터 흐름 분석을 위한 데이터를 수집하기 위한 단계이다.

```
1: {stmt2str=document.getElementsByTagName('button'),
enFunc=, vals=[button], vars=['button'], type=5, stmt=[
[Element@4fcd18c -> unknown locator] }
2: {stmt2str=addButton =
document.getElementsByTagName('button')[0], enFunc=,
vals=[], vars=[], type=1, stmt=[ Element@4fcd18c ->
unknown locator]}
                        ……
22: {stmt2str=i < incompleteTasksHolder.children.length,
enFunc=, vals=[], vars=[], type=3, stmt=false}
23: {stmt2str=completedTasksHolder =
document.getElementById('completed-tasks'), enFunc=,
vals=[], vars=[], type=1, stmt=[Element@52e2a2d1 ->
unknown locator]}
24: {stmt2str=var i = 0, enFunc=, vals=[], vars=[], type=0,
stmt=0}
25: {stmt2str=i < incompleteTasksHolder.children.length,
enFunc=, vals=[], vars=[], type=3, stmt=true}
                        ……
33: {stmt2str=i < incompleteTasksHolder.children.length,
enFunc=, vals=[], vars=[], type=3, stmt=true}
```

그림 5. 수행된 구문에 대한 정보

그림 5 는 테스팅 단계에서 수집된 정보이다. ‘stmt2str’은 실행된 구문을 문자열로 변환한 것이고, ‘enFunc’는 실행된 구문을 포함하는 함수이다. ‘vals’와 ‘vars’는 분기문이나 반복문을 수행할 때 사용된 변수들을 저장한다. ‘type’은 수행된 구문의 종류이며 프로그램 수정단계에서 수정된 구문을 나타낸다.

'stmt'는 실행된 구문의 실제 결과 값을 가지고 있다.

구문과 관련된 정보를 수집하기 위해 웹 브라우저에서 웹 애플리케이션의 자바스크립트 함수들을 실행하여 각 함수마다 데이터 흐름을 분석한다. 각 함수의 수행은 QUnit 을 사용하여 수행한다. QUni[26]t 은 자바스크립트 단위테스트 프레임 워크로 jQuery[27]를 포함하여 일반적인 자바스크립트 코드까지 테스트 할 수 있다. QUnit 은 각 단위마다 실행환경을 생성하며 비동기 코드를 검증하는 기능도 제공한다. 또한, 테스트 결과를 웹 브라우저를 통해 시각적으로 확인할 수 있다. 이를 활용하여 함수단위로 수행된 프로그램의 수행 결과를 수집하여 데이터 흐름을 분석하는데 사용한다.

4.3 결과 분석

결과 분석 단계에서는 테스팅 단계에서 함수 단위로 수집된 결과를 분석하여 각 함수에 대한 데이터 흐름 관계를 추출한다.

데이터 흐름 관계는 Denaro et al. 이 개발한 DReaDs 기법을 사용하였다[19]. DReaDs 는 프로그램의 실행 경로에 따른 Reaching Definition 분석을 수행한다. Reaching Definition 분석은 어떤 변수가 재정의되기 전까지 사용되는 흐름을 관찰하여 어느 시점까지 변수가 유효한지 알아내는 것이다. DReaDs 는 기본 블록을 사용하지 않고 각각의 구문을 CFG 의 노드로 보고 분석한다. DReaDs 의 데이터 흐름 방정식은 아래와 같다.

$$NEXT = (PRIOR - KILL) \cup GEN$$

NEXT 와 PRIOR 는 각각 특정 시점에 실행된 구문을 기준으로 각각 이전에 실행된 구문과 다음에 실행된 구문의 Reaching Definition 을 나타낸다. GEN 은 정의된 변수가 프로그램의 마지막 구문이 수행될 때까지 재정의 되지 않은 변수의 집합이고 KILL 은 재정의된 변수의 집합이다. DReaDs 의 데이터 흐름 방정식을 사용하여 사용된 모든 변수의 Reaching Definition 을 동적으로 계산할 수 있다. 동적으로 계산된 Reaching Definition 은 $\{i_1, i_2, \dots, i_n\} \rightarrow \{D_1, D_2, \dots, D_n\}$ 으로 표현된다. i_* 은 순차적으로 수행된 구문을 나타내고, D_* 은 각 구문에 해당하는 Reaching Definition 이다.

본 논문에서는 동적으로 계산된 Reaching Definition 중, DOM 과 관련된 Reaching Definition 을 추출하여 DOM 과 관련된 연산을 수행할 때 어떤 변수가 영향을 미칠 수 있는지 분석한다. 분석결과는 테스트 케이스 생성이나 오류를 찾는데 도움을 줄 수 있다.

5. 시나리오 적용 및 결과 분석

본 논문에서는 웹 애플리케이션에 대하여 데이터 흐름 분석을 통해 DOM 과 관련된 데이터 흐름 관계를 추출하기 위하여 TodoList, Samegame 등 두 개의 공개된 웹 애플리케이션을 사용했다. ToDoList 는자바스크립트 기반의 간단한 작업목록을 관리하는 애플리케이션이고, Samegame 은 웹 기반의 게임 애플리케이션이다. 표 1 은 두 개의 웹 애플리케이션의 라인수, 함수의 수, DOM 연산을 포함하는 함수의 수를 나타낸다.

App	LOC	# of funs	# of DOM-related funs
TodoList	151	7	7 (100%)
Samegame	352	10	4 (40%)

표 1. 시나리오 적용을 위해 사용한 웹 애플리케이션의 특징

본 논문에서는 데이터 흐름 분석을 통해 DOM 과 관련된 데이터 흐름 관계를 추출하기 위해서 DOM 과 관련된 연산을 하는 함수들을 대상으로 테스팅을 수행하였다. 테스팅에 사용한 웹 브라우저는 Firefox 18.02 이고 Max OS X, Intel Core i5 2.7 GHz CPU 와 8GB 램을 장착한 장비에서 수행하였다.

5.1 프로그램 수정 실행 시간 측정

본 절에서는 프로그램 수정에 걸리는 시간과 프로그램 수정으로 인해 성능에 얼만큼 영향을 주는지 측정하였다.

우선, 웹 애플리케이션 수정을 10 번 수행하여 소요된 평균 시간을 측정했다. ToDoList 는 평균 256ms, Samegame 은 504ms 만큼 소요되었다.

App	TodoList	Samegame
Orig-time	635ms	74ms
Inst-time	690ms	96ms

표 2. 수정되지 않은 프로그램과 수정된 프로그램의 테스팅 시간

프로그램 수정이 프로그램에 성능에 얼만큼 영향을 주는지 측정하기 위해 구체적인 사용 시나리오를 설계하여 테스팅을 수행하였다. ToDoList 를 이용한 사례 연구에서는 할 일 추가/삭제/변경 기능을 순서대로 실행하는 사용 시나리오를 순서대로 실행하였다. Samegame 에서는 이벤트를 발생시킬 수 있는 버튼을 무작위로 누르는 시나리오를 실행하였다. 설계한 사용 시나리오를 수행하여 수정되지 않은 프로그램과 수정된 프로그램 실행에 소요된 시간을 비교함으로써 프로그램의 성능에 영향을 주는지 파악한다.

TodoList 가 Samegame 에 비해 많은 시간을 소요하는 이유는 DOM 과 관련된 연산이 빈번하게 일어나기 때문이다. 표 2 의 Orig-time 과 Inst-time 은 각각 수정되지 않은 프로그램과 수정된 프로그램이 테스팅에 소요한 시간을 나타낸다. TodoList 의 경우 55ms 의 차이를 보이고 Samegame 의 경우 22ms 의 차이를 보이기 때문에 수정 전 프로그램의 수행시간과 큰 차이를 보이지 않음을 알 수 있다.

5.2 동적인 데이터 흐름 관계 분석

TodoList 와 Samegame 등 두 개의 웹 애플리케이션에 대하여 DOM 관련 데이터 흐름을 추출하기 위해 DOM 과 관련된 연산을 포함하는 함수들에 대해 테스팅을 수행했다. 각 함수들을 테스트하기 위해 수동으로 테스트 케이스를 생성하였으며 테스트 케이스를 생성하는데 TodoList 의 경우 4 분, Samegame 의 경우 약 2 분 소요 되었다.

5.2.1 TodoList 사례 분석

TodoList 에 대해서는 수정된 일곱개 함수에 대하여 생성한 테스트 케이스를 수행하였다. 분석 결과로 총 99 개의 데이터 흐름 관계가 도출되었고, DOM 과 관련된 데이터 흐름은 94 개(94%)가 도출 되었다. TodoList 의 특성상, 복잡한 계산보다는 사용자의 간단한 입력 값을 DOM 노드에 할당하는 연산이 대부분이였기 때문에 DOM 관련 데이터 흐름이 대부분이였다. DOM 노드에 입력값을 할당하는 연산은 자바스크립트 변수에 DOM 노드 참조자를 할당한 후, 그 변수를 사용하여 DOM 노드와 상호작용한다. 즉, 변수에 저장된 참조자는 DOM 노드의 요소에 접근하며 새로운 값을 해당 요소에 할당한다. DOM 과 관련되지 않은 데이터 흐름 관계는 반복문의 카운터 변수의 초기화 및 증감문이다. 하지만 카운터 변수 역시 DOM 노드의 인덱싱에 사용될 경우 DOM 관련 데이터 흐름에 영향을 끼칠 수 있다.

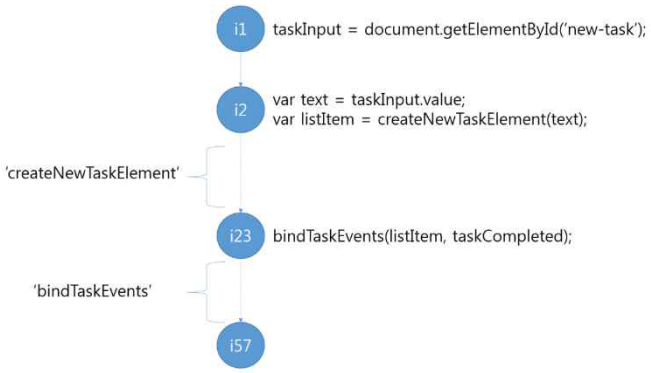


그림 6. TodoList 의 'addTask'함수의 실행 흐름

'addTask'함수로부터는 가장 많은 30 개의 데이터 흐름 관계를 도출 하였다. 왜냐하면 'addTask'함수는 수행시 'createNewTaskElement'와 'bindTaskEvents' 함수를 호출하여 새로운 데이터흐름을 생성하기 때문에 많은 데이터흐름이 발견되었다. 'createNewTaskElement'에서 자바스크립트를 사용하여 새로운 DOM 트리를 생성하고 'addTask' 함수에서 정의된 자바스크립트 변수 'listItem'에 'createNewTaskElement'함수에서 생성된 DOM 트리의 한 노드에 할당된 후, 'addTask' 함수로 DOM 트리가 할당된다. 또한, 'addTask' 함수가 호출하는 'bindTaskEvents' 함수에서 'createNewTaskElement' 함수 호출시 정의한 DOM 트리를 참조하여 'taskComplete' 함수를 바인딩한다. 이와 같이 여러 함수 호출에 걸쳐 DOM 노드와 자바스크립트 프로그램의 상호작용을 순차적으로 수행된 구문으로 나타내고 구문의 순서에 따른 데이터 흐름 분석을 통해 복잡한 계산 없이 데이터 흐름을 파악할 수 있다.

5.2.2 Samegame 사례 분석

Samegame 에 대해서는 수정된 네 개의 함수에 대하여 생성한 테스트 케이스를 수행하였다. 실행결과로 총 740 개의 데이터 흐름이 도출 되었고, DOM 과 관련된 데이터 흐름은 306 개(41%)가 도출 되었다. 네 개의 함수에서 740 개의 데이터 흐름이 생성된 이유는 반복문 때문이다. DReaDs 기법은 수행된 구문을 기본 단위로 GEN 과 KILL 을 계산하기 때문에 반복문의 카운터 변수에 따른 KILL 과 GEN 이 발생한다. 따라서 반복문의 반복횟수가 많아수록 데이터 흐름이 많아질 수 있으나 DOM 관련된 데이터 흐름은 상대적으로 적었다.

Samegame 은 DOM 과 관련된 데이터 흐름의 수가 가변적이다. 왜냐하면 'random' 함수에 의해 생성된 값에 의해 DOM 노드와 관련된 연산이 발생하지 않을 수 있기 때문이다. 'onBall' 함수는 DOM 노드를 클릭했을 때, 같은 값을 갖는 DOM 노드가 인접해 있는 경우 DOM 관련 데이터 흐름이 발생한다. 그러나 'random'함수에 의해 인접한 DOM 노드가 같은 값을 갖지 않도록 배치된다면 DOM 관련 데이터 흐름이 발생하지 않는다. 또한 같은 값을 가진 인접한 DOM 노드의 수에 따라 DOM 과 관련된 연산이 증가하거나 감소하는 가변성을 가지고 있다. 그러나 가변적인 요소가 있더라도 수행된 구문들을 분석하여 데이터 흐름 도출하여 DOM 과 관련된 연산이 있었다면 어떤 변수가 DOM 노드와 연산에 활용되는지 파악할 수 있다.

Samegame 의 DOM 관련 연산은 'document["img" + adj[n]].src = eval("off" + crayon + ".src")'과 같이 eval 구문을 사용하여 동적으로 할당될 값을 계산하거나 'document.scores.click.value'와 같이 복잡한 구

조의 참조을 사용한다. 이와 같은 경우도, 동적으로 수행된 구문의 결과를 사용하기 때문에 분석에 문제가 없고 관련된 오류를 쉽게 찾을 수 있다.

6. 결론 및 향후 연구

본 연구에서는 웹 애플리케이션을 대상으로 자바스크립트와 DOM 의 상호작용과 관련된 데이터 흐름을 추출하기 위해 프로그램을 수정하고 DReaDs[18] 기법을 적용하여 상호작용시 어떤 변수가 영향을 미치는지 분석했다. 그리고 제안한 방법의 핵심부분을 구현하고 시나리오를 마련하여 구현한 시스템을 시나리오를 기반으로 평가하였다.

본 연구가 기여한 부분은 다음과 같이 요약할 수 있다: (1) 웹 애플리케이션의 동적인 데이터 흐름 분석을 위해 프로그램을 수정하는 기법을 분석하고 적용했으며, (2) 웹 애플리케이션에 동적인 데이터 흐름 분석 기법을 적용하여 어떤 변수가 DOM 과 관련된 연산을 수행하는지 쉽게 파악할 수 있도록 하였다.

향후 연구로 본 접근 방법을 보다 규모가 큰 웹 애플리케이션에 적용하여 추가적으로 발생할 수 있는 문제 및 해결방안에 대해 분석하고, 웹 애플리케이션 테스팅을 위한 DOM 구조 생성 및 테스트 케이스 생성방안에 대해 연구할 계획이다.

Acknowledgement

본 연구는 민군겸용기술사업(Dual Use Technology Program) 지원을 받아 수행되었습니다 (UM13018RD1).

참고문헌

[1] Document Object Model(DOM), http://www.w3.org/DOM/, 27 Dec. 2015.

[2] S. H. Jensen, M. Madsen, and A. Moller, "Modeling the HTML DOM and browser API in static analysis of JavaScript Web Applications", *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, 2011.

[3] M. Madsen, B. Livshits, and M. Fanning, "Practical Static Analysis of JavaScript Applications in the Presence of Frameworks and Libraries", *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, 2013.

[4] F.Ocariza, K. Bajaj, K. Pattabiraman, and A. Mesbah, "An Empirical Study of Client-Side JavaScript Bugs", *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013.

[5] S. Artzi, J. Dolby, S. H. Jensen, A. Moller, and F. Tip, "A Framework for Automated Testing of JavaScript Web Applications", *in Proceedings of the 33rd International Conference on Software Engineering*, 2011.

[6] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song, "A Symbolic Execution Framework for JavaScript", *in Proceedings of the 2010 IEEE Symposium on Security and Privacy*, 2010.

[7] K. Sen, S. Kalasapur, T. Brutch, and S. Gibbs, "Jalangi: A Selective Record-Replay and Dynamic Analysis Framework for JavaScript", *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, 2013.

[8] C. S. Jensen, A. Moller, and Z. Su, "Server Interface Descriptions for Automated Testing of JavaScript Web Applications", *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, 2013.

[9] A.Mesbah, A. van Deursen, and D. Roest, "Invariant-based Automatic Testing of Modern Web Applications", *IEEE Transactions on Software Engineering*, 2012.

[10] A.Mesbah, E. Bozdag, and A. van Deursen, "Crawling Ajax by Inferring User Interface State Changes", *International Conference on Web Engineering*, 2008

[11] S. Mirshokraie, and A. Mesbah, "JSART: JavaScript Assertion-based Regression Testing", *12th International Conference on Web Engineering*, 2012

[12] S. Mirshokraie, A. Mesbah, and K. Pattabiraman, "Guided Mutation Testing for JavaScript Web Applications", *IEEE Transactions on Software Engineering*, 2014

[13] S. Mirshokraie, A. Mesbah, and K. Pattabiraman, "JSEFT: Automated JavaScript Unit Test Generation", *IEEE 8th International Conference on Software Testing, Verification and Validation*, 2015

[14] A. M. Fard, A. Mesbah, and E. Wohlstadter, "Generating Fixtures for JavaScript Unit Testing", *IEEE/ACM 30th International Confernece on Automated Software Engineering*, 2015.

[15] C. Liu, D. Kung, and P. Hsia, "An Object-Based Data Flow Testing Approach for Web Applications", International Journal of Software Engineering & Knowledge Engineering, 2001.

[16] F. E. Allen and J.Cocke, "A program data flow analysis procedure", Communications of the ACM, 1976.

[17] S. Rapps and E. J. Weyuker, "Selecting Software test data using data flow information", *IEEE Transactions on Software Engineering*, 1985

[18] P.M. Herman, "A Data flow Analysis approach to program testing", Australian Computer Journal 1976.

[19] G. Denaro, M. Pezze, and M. Vivanti, “On the Right Objectives of Data Flow Testing”, *IEEE 7th International Conference on Software Testing, Verification and Validation*, 2014.
[20] G. Denaro, A. Margara, M. Pezze, and M. Vivanti, “Dynamic Data Flow Testing of Object Oriented Systems”, *IEEE/ACM 37th International Conference on Software Engineering*, 2015.
[21] Mozilla Rhino, https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino, 27 Dec. 2015.
[22] Alexa, The Web Information Company, www.alexa.com, 27 Dec. 2015.
[23] G. Li, E. Andreasen, and I. Ghosh, “SymJS: Automatic Symbolic Testing of JavaScript Web Applications”, FSE, 2014.
[24] JavaScript technologies overview, https://developer.mozilla.org/en-US/docs/Web/JavaScript/JavaScript_technologies_overview, 27 Dec. 2015
[25] F. Nielson, H. R. Nielson, and C. Hankin, “Principles of Program Anslysis”, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.
[26] QUnit, https://qunitjs.com/, 27 Dec. 2015.
[27] jQuery, https://jquery.com/, 27 Dec. 2015.

회귀 시험을 위한 통계적 분석 기반 테스트 항목 우선순위화 기법

조영환*, 이은석**

* 성균관대학교 DMC 공학과
** 성균관대학교 컴퓨터공학과
경기도 수원시 장안구 서부로 2066
{yhwan.cho, leees}@skku.edu

요약: 회귀 시험은 시간이 지남에 따라 테스트 항목의 수가 지속적으로 증가하여 많은 비용과 시간을 필요로 하게 되고 이로 인해 시험 수행에 어려움을 갖게 된다. 이를 해결하기 위해 많은 테스트 항목 최적화 기법들이 제안되었다. 그 중 테스트 항목 우선순위화 기법은 테스트 항목의 순서를 조정하여 대상 시험이 보다 효과적으로 결함을 탐지할 수 있도록 하는 기법이다. 이러한 우선순위화 기법들 중에는 과거 시험 정보를 이용한 기법들이 있으며, 다른 기법들에 비해 보다 적은 비용으로 결함 탐지 능력을 높인다는 장점을 가진다. 본 논문에서는 과거 시험 정보를 이용한 기법을 기반으로 통계적 분석을 통하여 대상 프로젝트의 과거 시험 정보가 가지는 시간적 유효 범위와 테스트 항목들의 연관성을 정의하고 이를 활용한 테스트 항목 우선순위화를 통해 결함 탐지 능력을 높일 수 있는 기법을 제안한다.

핵심어: 테스트 항목 우선순위화, 회귀 시험, 과거 시험 정보, 통계적 분석

1. 서론

S/W 품질 관리를 목적으로 하는 여러 가지 시험 활동 중 하나인 회귀 시험은 새로운 기능의 추가나 버그의 수정 등으로 인한 소스 코드의 변경이 기존의 정상적인 동작에 나쁜 영향을 미치지 않는다는 것을 확인하는 시험이다. 이와 같은 목적을 달성하기 위해서는 기본적으로 소스 코드의 변경이 있을 때마다, 혹은 각 개발 단계를 새롭게 거칠 때마다 모든 테스트 항목들에 대해 확인을 해야 한다. 소스 코드의 변경이 늘어날 수록 회귀 시험을 위한 테스트 항목들도 늘어나게 되고, 그에 따라 회귀 시험에 필요한 비용과 시간도 증가하기 때문에 심각한 경우에는 회귀 시험을 정상적으로 진행하지 못하는 문제가 발생하기도 한다. 이와 같은 문제를 해결하기 위하여 지금까지 테스트 항목의 최소화, 테스트 항목의 부분적 선택, 테스트 항목의 우선순위화 등과 같은 최적화 기법들이 제안되었다[1]. 이 중 테스트 항목의 우선순위화 기법은 테스트 항목의 실행 순서 변경을 통해 많은 결함이 조기에 발견될 수 있도록 하는 방식의 최적화 기법이다. 이는 시험 초기에 더욱 더 많은 결함을 발견하도록 하여 임의적인 시험 종료나, 예기치 못하게 시험이 방해되는 상황에서도 큰 효과를 가질 수 있다는 장점을 가진다. 기존의 많은 테스트 항목 우선순위화 기법들은 소스 코드 정보나 테스트 항목이 갖는 정보를 이용하여 테스트 항목의 우선순위를 결정하였다[1]. 이와 더불어 과거 회귀 시험의 이력 정보에 기반한 우선순위화 기법들도 많이 연구되었는데, 과거에 발견된 결함 정보나 테스트 항목의 실행 결과를 사용하여 우선순위를 결정하는 방식으로 상당한 효과를 보인다는 것이 확인되었다[2-5].

본 논문에서는 이전에 실행된 회귀 시험의 이력 정보를 바탕으로 통계적 분석을 통해 과거 시험 정보의 유효 범위와 각 테스트 항목 간의 연관성을 결정하고 이를 바탕으로 테스트 항목을 우선순위화하여 결함 탐지 능력을 높이는 방법을 제안한다.

본 논문은 다음과 같이 구성된다. 2 장에서는 기존에 제안되었던 테스트 항목 우선순위화 기법들을 살펴보고 3 장에서는 통계적 분석에 기반한 테스트 항목 우선순위화 기법을 제시한다. 4 장에서는 실험 계획과 평가 방법에 대해 다루고, 5 장에서 결론을 맺는다.

2. 관련연구

기존에 제안된 테스트 항목 우선순위화 기법들은 다양한 정보를 바탕으로 테스트 항목의 실행 순서를 결정하게 되는데, 대표적인 정보로는 테스트 항목의 코드 커버리지(code coverage)가 있다. 테스트 항목이 포함하는 코드 커버리지를 측정하여 넓은 범위를 갖는 테스트 항목 순으로 우선순위를 할당하는 기법이 있으며, 각 테스트 항목이 포함하는 코드 커버리지의 중복 문제를 해결하기 위해 아직 포함되지 않은 범위 중에서 가장 넓은 범위를 갖는 테스트 항목의 우선순위를 높이는 기법도 있다[6]. 또한 자료 흐름(data flow)[7], 코드 변경(code change)[8]과 같은 다양한 정보에 기반한 우선순위화 기법들이 제안되었

으며, 각 테스트 항목의 비용, 결함의 심각도 등을 고려한 우선순위화 기법들도 연구되었다[1]. 그리고 과거 결함 탐지 정보, 과거 테스트 항목 실행 정보와 같이 과거 시험 정보를 이용한 기법들도 많이 제안되었는데, 이 연구들은 코드 커버리지나 코드 변경과 같이 소스 코드에 대한 분석에 의존하지 않아 비용이 상대적으로 적게 요구된다는 장점을 가지고 있다. 그리고 지금까지 많은 연구들이 그 효과를 실험적으로 입증하여 'proven performer'로 표현되기도 한다[9].

과거 시험 정보를 이용한 테스트 항목 우선순위화 기법 중 하나로 그림 1 과 같이 테스트 항목 세트의 실행에 대하여 특정 크기로 설정된 윈도우를 기준으로 테스트 항목의 과거 결함 발견 여부와 실행 여부를 가지고 우선순위화 하는 기법이 제안되었다[2]. 연구 [2]는 특정 시간 내에 실패한 테스트 항목의 우선순위를 높이는 동작과 또 다른 특정 시간 내에 실행된 테스트 항목의 우선순위를 낮추는 동작을 통해서 앞으로 실행될 테스트 항목의 우선순위를 결정하는 방법이다. 이 방법은 실제로 미국 IT 업체 Google 에서 제공하는 data set 을 통해 실험적으로 그 효과를 입증하였다. 그리고 해당 실험 결과에서는 여러 윈도우 크기에 따라 그 효과가 각기 다르게 나타났으며 이를 통해 윈도우 크기의 결정이 결과에 상당한 영향을 미친다는 것을 보여주고 있다.

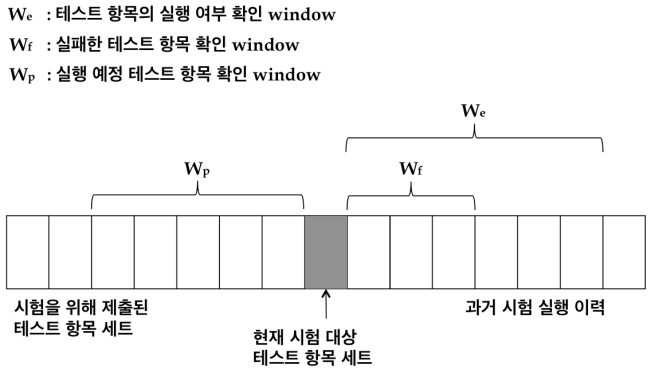


그림 1 테스트 항목 세트 실행 windows

과거 발견된 결함 정보와 각 테스트 항목이 실행된 시간적 거리에 기반하여 가중치를 계산하고 이를 통해 우선순위를 결정하는 기법도 제안되었다[3]. 이 기법은 과거에 수행되었던 시험 정보 중에서 테스트 항목들의 성공, 실패 여부를 가지고 식 (1)와 같이 실패 행렬(Failure Matrix)을 구성하고 식 (2)과 같이 시험의 순서를 기준으로 현재 시점으로부터 가장 가까운 3 개의 버전에 대해서 가중치를 부여한다. 그리고 식 (3)과 같이 두 값의 연산을 통해 우선순위 값을 결정하는 방식을 취하고 있다. 그리고 테스트 항목이 수행되는 시간을 고려하여 우선순위의 결정을 위한 값을 식 (4)와 같이 변경하여 최종적으로 사용한다.

과거 시험 정보 중에서 테스트 항목의 직전 실행 여부를 기준으로 확률에 기반하여 랜덤하게 테스트 항목을 선택하고, 선택된 테스트 항목에 대해서 최근에 수행되지 않은 테스트 항목 순으로 우선순위를 부여하여 회귀 시험을 진행하는 방법도 제안되었다[4]. 그리고 이와 유사한 방법으로 과거 시험에서 사용된 테스트 항목의 우선순위와 함께, 전체 결함 대비 각 테스트 항목들이 탐지한 결함의 비중, 그리고 각 테스트 항목들이 실행되지 않은 시간을 기준으로 우선순위를 정하는 방법도 제안되었다[5].

ω : 가중치
MF : 실패 행렬
ps : 각 테스트 항목의 우선순위
Te : 각 테스트 항목 별 실행 시간

$$MF_{m,n} = \begin{pmatrix} -1 & 1 & \cdots & -1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & 1 & \cdots & -1 \end{pmatrix} \quad (1)$$

$$\omega_i = \begin{cases} 0.7, & \text{if } i = 1 \\ 0.2, & \text{if } i = 2 \\ 0.1, & \text{if } i \geq 3 \end{cases} \quad (2)$$

$$ps_i = \sum_{i=1\ldots m} MF[i,j] * \omega_i \quad (3)$$

$$ps_i = \begin{cases} t+1 & Te_i \geq T_{\max} \\ ps_i + \dfrac{Te_i}{T_{\max}}, & \text{otherwise} \end{cases} \quad (4)$$

ω: 가중치
MF : 실패 행렬
ps : 각 테스트 항목의 우선순위
Te : 각 테스트 항목 별 실행 시간

지금까지 언급한 과거 시험 정보에 기반한 연구들은 각각 자신들이 정한 시간적 범위 내의 과거 시험 정보를 활용하여 다음 테스트 항목 우선순위화에 활용하였다. 이렇게 고정된 기준의 정보 활용은 대상 프로젝트와 시험의 특성에 따라 변화하는 결함 발생 현상에 대응할 수 없으므로 결함 탐지 능력의 향상을 저하시키는 원인이 된다. 그리고 테스트 항목의 실행 여부와 성공, 실패 여부만으로 앞으로 결함이 발견되기 쉬운 테스트 항목(error-prone test cases)을 찾는 것이기 때문에 과거 발생 결함이나 결함 탐지 기록을 바탕으로 예상하기 어려운 결함의 발생에는 대응하기 어려운 한계를 가지고 있다.

3. 통계적 분석에 기반한 테스트 항목 우선순위화

앞서 2 장에서 소개한 과거 시험 정보를 바탕으로 한 기법들은 사용되는 정보가 가지는 시간적 유효 범위에 대해서는 각기 다른 기준을 가지고 효과를 주장하고 있으며, 연구 [2]에서는 과거 시험 정보의 적용 범위에 따라 그 효과가 차이를 보인다는 것을 실험으로 확인하였다.

본 논문에서는 이전에 실행된 회귀 시험의 결과를 바탕으로 그 정보의 시간적 유효 범위를 측정하여 이후 테스트 항목 우선순위화의 효과를 높일 수 있는 기법을 제안한다. 그리고 과거 시험 정보를 바탕으로 예상하기 어려운 결함의 탐지를 위하여 각 버전들 간의 시험 결과 정보가 가지는 연관성을 통계적으로 분석하여 테스트 항목 우선순위화에 활용하는 방법도 함께 제안한다.

3.1. 과거 시험 정보의 유효 시간 결정

본 논문에서 주장하고자 하는 과거 시험 정보의 유효성 정의 기법은 영역 특수적(domain-specific)인 발견법(heuristics)에 기반하고 있다. 다시 말해서 각 테스트 항목의 결함 발생 분포와 빈도는 각 프로젝트가 갖는 여러 가지 특성에 의해서 결정된다고 보는 것이다. 따라서 과거 시험 정보를 통계적으로 분석하면 동일 프로젝트 내에서는 과거에 실패한 테스트 항목이 다시 실패할 수 있는 확률을 구할 수 있으며, 이를 이용하여 테스트 항목의 우선순위를 조정하면 보다 높은 결함 탐지 능력을 가질 수 있다. 본 논문에서는 식 (5)와 같이 과거 결함을 탐지한 테스트 항목이 특정 시간 내에 임계값보다 높은 확률로 결함을 탐지했다면 그 시간을 결함이 다시 발생할 수 있는 유효한 범위로 결정하고 이를 통해 해당 테스트 항목의 우선순위를 조정하는 기법을 제안한다. 여기서 측정된 유효 범위 값은 회귀 시험 결과 정보를 바탕으로 계속적으로 갱신된다.

$$HEB = k, \quad \sum_{i=1}^{k} P(FT_{\alpha+i} \mid FT_{\alpha}) > \gamma \qquad (5)$$

a: 최초 실패 테스트 항목 index
γ : 임계값
HEB : 유효 범위 *(Historical Effective Bound)*
FT : 실패한 테스트 항목

3.2. 테스트 항목의 연관성 분석과 활용

과거 시험 정보를 이용한 테스트 항목 우선순위화 기법들은 다른 기법에 비해 시간과 비용이 적게 요구되므로 지속적인 통합 개발 환경에서 그 활용성과 효과가 높다고 할 수 있다. 지속적으로 코드의 수정이 일어나는 개발 환경에서는 기존에 발생한 결함을 고치기 위해 상당 수의 수정 사항이 계속적으로 발생한다. 따라서 과거 시험 결과 중 결함 탐지 여부를 바탕으로 테스트 항목 간의 연관성을 분석하면 실패한 테스트 항목에서 검출된 결함의 수정으로 인해 발생하는 다른 결함을 예측할 수 있다. 따라서 본 논문에서는 결함 탐지 여부를 대상으로 통계적 분석을 통하여 테스트 항목 간의 연관성을 측정하여 우선순위화에 사용하는 방법을 제안한다. 연관성은 식 (6)과 같이 각 테스트 항목 간의 최대 상호정보량(Mutual Information)을 가지는 테스트 항목의 쌍을 구하는 방식으로 측정된다. 측정된 값에 따라 이전에 실패한 테스트 항목과 연관성이 높다고 결정된 테스트 항목들은 다음 회귀 시험 시 높은 우선순위를 가지고 수행된다.

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right) \qquad (6)$$

4. 실험 계획 및 평가 방법

제안된 기법을 평가하기 위해 아래와 같은 실험을 진행할 계획이다.

첫 번째로, 과거 정보의 유효 시간을 결정하여 적용하는 것이 테스트 항목 우선순위화의 결함 탐지 능력을 개선시킨다는 것을 확인하기 위하여 각 테스트 항목 별로 누적된 과거 결함 탐지 횟수를 고려하여 우선순위를 결정하는 값을 계산한다. 결정된 유효 시간에 따라 우선순위 값을 재조정하는 방법으로 제안된 기법의 효과를 측정한다. 그리고 이를 유효 시간을 고려하지 않은 결과와 비교하여 평가한다.

두 번째로, 테스트 항목 간의 연관성을 분석하여 활용하는 방식의 효과를 확인하기 위해서 이전 시험에서 실패한 테스트 항목과 연관성이 높게 측정된 테스트 항목의 우선순위를 높이는 실험을 진행한다. 여기서 우선순위를 결정하는 값은 가중치에 따라 최종적으로 결정되는데 연관성의 정도에 따라 가중치를 다르게 부여하는 방식으로 진행한다. 그리고 상호정보량으로 연관성을 측정하는 것과 더불어 다른 측정 방법으로도 진행하여 비교, 분석할 계획이다. 다른 측정 방법으로는 상관계수(Correlation Coefficient)를 구하여 측정하는 방법과 직전 시험 결과 정보를 바탕으로 현재 결함 발생을 예측한다는 측면에서 은닉 마르코프 모델(Hidden Markov Model)을 사용할 계획이다.

실험 대상으로는 계속적으로 개발이 진행 중인 오픈 소스 프로젝트 중에서 코드의 수정이 활발하게 일어나는 프로젝트를 선정하여 진행하도록 한다.

$$APFD = 1 - \frac{TF_1 + TF_2 + ... + TF_m}{nm} + \frac{1}{2n} \quad (7)$$

n : 전체 테스트 항목 수
m : 결함을 탐지한 테스트 항목 수
TF : 결함을 탐지한 테스트 항목의 위치

제안된 기법의 효과를 평가하기 위한 방법으로는 평균 결함 탐지율(Average Percentage of Fault Detected, APFD)을 사용한다. 평균 결함 탐지율은 식 (7)에서 볼 수 있듯이 테스트 항목 세트에서 결함을 발견한 테스트 항목의 위치에 따라 0 에서 1 사이의 값을 가지며 높을수록 결함을 조기에 탐지하는 능력을 가진 것으로 판단한다. 이 방법은 Rothermel 에 의해 제안되었으며[6], 이후 테스트 항목 우선순위화 기법의 평가에 널리 쓰이고 있다.

5. 결론

본 논문에서는 과거 시험 정보에 기반한 테스트 항목 우선순위화 기법의 결함 탐지 능력을 높이기 위해 통계적 분석을 통해 과거 시험 정보의 유효 시간과 테스트 항목들 간의 연관성을 분석하고 이를 활용하여 테스트 항목의 우선순위를 조정하는 방법을 제안하였다. 향후에는 실험을 통해 제안된 기법의 효과를 확인하고 이를 바탕으로 회귀 시험에 필요한 한정된 자원과 시간을 고려하여 테스트 항목 우선순위화의 효율성을 높일 수 있는 방법을 추가로 제안할 계획이다.

Acknowledgements
이 논문은 2015 년도 정부(교육과학기술부)의 재원으로 한국연구재단-차세대정보컴퓨팅개발사업의 지원을 받아 수행된 연구임(No. 2015045358).

참고문헌

[1] Yoo, S., & Harman, M. "Regression testing minimization, selection and prioritization: a survey." Software Testing, Verification and Reliability 22.2, pp.67-120, 2012.
[2] Elbaum, S., Rothermel, G., & Penix, J. "Techniques for improving regression testing in continuous integration development environments." Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, pp.235-245, 2014.
[3] Marijan, D., Gotlieb, A., & Sen, S. "Test case prioritization for continuous regression testing: An industrial case study." Software Maintenance (ICSM), IEEE International Conference on. IEEE, pp.540-543, 2013.
[4] Kim, J. M., & Porter, A. "A history-based test prioritization technique for regression testing in resource constrained environments." Software Engineering (ICSE), Proceedings of the 24rd International Conference on. IEEE, pp.119-129, 2002.
[5] Fazlalizadeh, Y., Khalilian, A., Azgomi, M. A., & Parsa, S. "Prioritizing test cases for resource constraint environments using historical test case performance data." Computer Science and Information Technology (ICCSIT), 2009 2nd IEEE International Conference on. IEEE, pp.190-195, 2009.
[6] Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. "Test case prioritization: An empirical study." Software Maintenance (ICSM), IEEE International Conference on. IEEE, pp.179-188, 1999.
[7] Hutchins, M., Foster, H., Goradia, T., & Ostrand, T. "Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria." Proceedings of the 16th international conference on Software engineering. IEEE Computer Society Press, pp.191-200, 1994.
[8] Jiang, B., Zhang, Z., Chan, W. K., & Tse, T. H. "Adaptive random test case prioritization." Automated Software Engineering (ASE). 24th IEEE/ACM International Conference on. IEEE, pp.233-244, 2009.
[9] Harman, M. "Making the case for MORTO: Multi objective regression test optimization." 2011 Fourth International Conference on Software Testing, Verification and Validation Workshops. IEEE, pp.111-114, 2011.

인수 (Acceptance) 테스트 자동화 실무 사례

신현일, 조성민, 최근호

에스코어
경기도 성남시 분당구 황새울로 258 번길 31
{hi25.shin, sungmin79.jo, gunho14.choi}@samsung.com

요약: 소프트웨어 개발에서 테스트 자동화의 중요성이 커지고 있다. Scrum 과 같이 개발/테스트를 주기적으로 반복하는 경우 회귀테스트의 역할이 중요하므로 더욱 테스트 자동화가 필요하다. 본 논문에서는 1 년여간의 인수 테스트 자동화 경험을 바탕으로 인수 테스트 자동화에서 고려해야 할 요소들을 설명한다. 테스트 자동화 설계, 테스트 수행 속도 개선, Flaky 테스트 제거, 테스트 실패 원인의 빠른 파악에 대해 경험한 바를 공유하고 관련 다른 방법들을 소개한다. 또한, 테스트 자동화 도구로 사용한 JUnit 과 Jenkins 의 활용 사례를 소개한다.

핵심어: 테스트 자동화, JUnit, Jenkins

※ 프로젝트 보안 때문에 실제 사례를 일반화하였습니다.

1. 서 론

소프트웨어 개발에서 품질과 개발 생산성의 향상을 위해 테스트 자동화가 점차 중요해 지고 있다. 특히, Scrum 처럼 개발/테스트를 반복하거나 지속적 통합/배포 (Continuous Integration/Delivery)를 하려면 반드시 테스트 자동화가 필요하다. 본 논문에서는 1 년여간의 인수 테스트 자동화 경험을 공유한다. 장기간 회귀테스트를 반복하면서 테스트 수행 속도를 높이고 Flaky 테스트 [1]가 없도록 하는 것이 중요하였다. 느린 테스트와 Flaky 테스트는 꼭 피해야 하는 테스트 유형이다 [2]. 또한, 테스트 실패를 빨리 분석하기 위해 테스트 코드를 읽기 쉽도록 개선하였고, 테스트 이름과 실패 메시지를 의미 있게 작성하여 테스트 코드를 살펴보지 않아도 테스트가 왜 실패하였는지를 알아낼 수 있었다.

본 논문에서는 인수 테스트 자동화 경험을 바탕으로 인수 테스트 자동화에서 고려해야 할 요소를 4 가지로 소개한다. a. 테스트 자동화 설계에서 고려해야 할 사항과 b. 테스트 수행 속도를 높이고 c. Flaky 테스트를 방지하는 방법, d. 테스트 실패 원인을 빨리 파악할 방법을 소개한다. 또한, 테스트 자동화 도구로 사용한 JUnit [3]과 Jenkins [4]에서 위 사항들을 어떻게 처리하였는지를 설명한다.

2. 테스트 환경

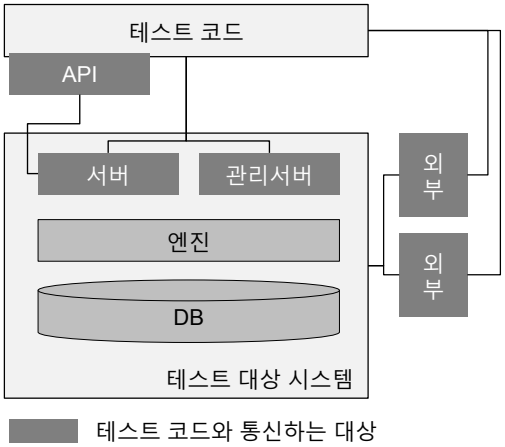


그림 1. 테스트 환경

테스트 대상 시스템은 API (Java Library), 서버, 관리서버, 엔진, DB 로 구성되어 있다 (그림 1). 테스트 대상 시스템의 기능은 API, 서버, 관리서버를 통해 사용할 수 있고, 외부시스템과 상호작용하는 기능도 존재한다. API 가 Java 로 개발되어 테스트 코드도 JUnit 으로 작성하였다. 테스트 코드와 테스트 대상 시스템은 다음과 같이 통신한다.

- API : 테스트 코드에서 import 하여 직접 호출
- 서버, 외부 시스템: 특정 프로토콜 (JMS: Java Message Service 등) 사용
- 관리서버: HTTP 사용 (REST API)

3. 테스트 자동화 설계

테스트 자동화도 하나의 시스템이기 때문에 개발초기에 설계가 중요하다. 프로젝트 성격과 테스트 엔지

니어의 경험 등을 고려하여 테스트 자동화 도구를 선택하고, Layered Architecture 로 설계하는 것이 필요하다.

3.1 테스트 자동화 도구

테스트 자동화 도구는 JUnit 과 같은 xUnit Framework 와 Cucumber [5], JBehave [6] 등의 BDD (Behavior-Driven Development) [7] 도구 중에서 선택할 수 있다. 각 방법의 특징이 표 1 에 나타나 있다.

표 1. 자동화 도구 특징

도구	특징
xUnit Framework	- 개발언어를 알면 쉽게 사용할 수 있다. - 요구사항 변경 시 Acceptance Criteria, 테스트 코드 모두를 수정해야 한다.
BDD 도구	- 새로운 도구를 배워야 한다. - Acceptance Criteria 를 별도 도구에서 관리할 필요가 없다. - Acceptance Criteria 자체가 Executable Specification [5 (p.6)]이 된다. - 테스트 도구의 기술적 지식을 요구하지 않기 때문에 non-technical 인력도 테스트를 쉽게 추가할 수 있다.

3.2 Layered Architecture

테스트 대상 시스템 변경에 의한 테스트 코드 변경을 최소화하고 테스트 코드 Readability 를 위해서 테스트 대상 시스템과 통신하는 로직을 테스트 코드에서 분리하는 것이 필요하다. 이를 위해 그림 2 와 같이 Layered Architecture 를 사용하는 것이 좋다 [8 (p.191)].

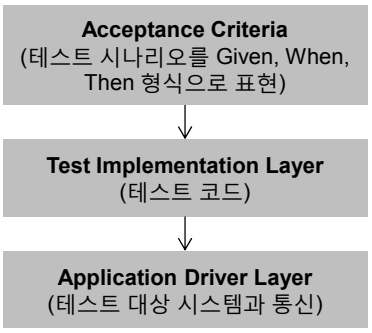


그림 2. Layers in Automated Tests

본 프로젝트에서는 표 2 와 같이 테스트 코드와 테스트 대상 시스템과 통신하는 로직을 구성하였다. 그림 3 에 간단한 테스트 예가 나타나 있다. 테스트 대상 시스템과 데이터를 주고받는 부분을 Util 클래스 (Application Driver Layer)에서 처리함으로써 테스트 코드를 간단히 작성할 수 있었고 테스트 코드의 의도를 쉽게 파악할 수 있었다.

추가로 테스트 코드에는 Readability 를 위해 복잡한 로직, if/for/switch 등의 분기 로직을 포함하지 않는 것이 좋다 [9 (p. 282), 10].

표 2. Layered Architecture 예시

Layer	package (Java)	설명
Test Implementation Layer	ft	User Story 기반 테스트
	long	시간 오래 걸리는 테스트
	external	외부시스템 기능 테스트
	rt	Smoke Tests, 성능테스트
	failover	Failover 테스트
	etc	기타 테스트
Application Driver Layer	common	통신하는 대상 별로 Util Class 를 생성함 AccountUtil BatchUtil RemoteShellUtil ExternalUtil ContentUtil APIUtil ServerUtil

```
// 테스트 코드

// 계정 A 가 계정 B 에 데이터를 전송하고 계정 B 로 데이터를 확인한다.
String accountA = AccountUtil.getAccount("11");
String accountB = AccountUtil.getAccount("12");
APIUtil.sendData(accountA, accountB, "this is a message.");
ContentUtil[] contents = APIUtil.readData(accountB, 1); // 비동기
assertThat("this is a message.", contents[0].getMessage());
```

```
// APIUtil class

// 데이터 개수가 count 일 때까지 polling 방식으로 확인한다.
// await(): 5 장 Awaitility 참고
ContentUtil[] readData(String account, int count) {
    try {
        await().until(readCallable(account, count));
        return getContents(account);
    }catch(ConditionTimeoutException timeout) {
    }
}

Callable<Boolean> readCallable(String account, int count) {
    int actual = getContentsCount(account);
    if actual > count, 예외 발생;
    if actual = count, return true;
    if actual < count, return false;
 }
```

그림 3. 테스트 코드와 Util 코드 예시

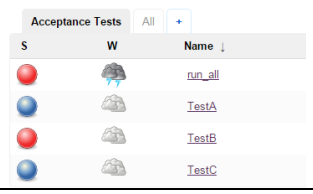
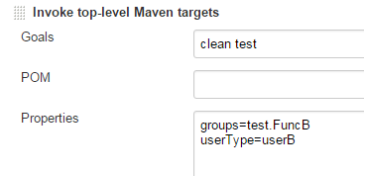
4. 테스트 수행 속도 높이기

테스트 수행 속도를 높이려면 병렬 수행이 가장 효과적이다. 병렬 수행을 위해서는 테스트가 atomic 해야 한다 [8 (p.205)]. 즉, 테스트를 동시에 수행하더라도 서로 간에 영향을 주지 않아야 하며, 어떠한 경우에라도 테스트를 (다른 테스트의 실행 없이) 단독으로 실행할 수 있어야 한다.

4.1 테스트 병렬 수행

병렬 수행 방법 4 가지를 소개한다.

A. 테스트를 그룹핑하고 Jenkins 활용한 병렬 수행
- 테스트들을 그룹핑 하고 Jenkins 에서 여러 개의 Test Job 을 생성하여 병렬로 수행한다. 본 프로젝트에서는 그림 4 와 같이 Category [11]를 사용하여 테스트들을 그룹핑하였고, Build Flow [12]를 사용하여 여러 개의 Test Job 을 동시에 실행하였다. 이 방법의 단점은 테스트가 많아지면 Category, Test Job 도 증가하여 관리의 부담이 생긴다는 점이다.
- 주의사항: 반드시 순차적으로 수행되어야 테스트들은 같은 Category 에 포함해야 한다. 예를 들어 동시에 사용되면 문제가 발생하는 기능이 있다면 이 기능의 테스트들은 한번에 하나씩 실행되어야 한다.

Jenkins Test Jobs	
TestB job 의 maven 실행 옵션 - test.FuncB Category 테스트 수행	

test.FuncB Category 에 포함된 테스트	@Category({FuncB.class})
	public class SomeTest {
	…
	}

run_all 스크립트 (Build Flow)	// TestA, TestB, TestC 를 동시에 실행 parallel ({ build("TestA") }, { build("TestB") }, { build("TestC") })

그림 4. Jenkins 활용한 병렬 수행

B. 테스트 클래스의 모든 메소드를 병렬로 수행
- JUnit 에서는 tempus-fugit [13]를 이용하여 특정 테스트 클래스 내 모든 메소드를 병렬로 수행할 수 있다. 그림 5 의 FuncTest 를 실행하면 test1, test2, test 가 동시에 같이 수행된다.

```
import
com.google.code.tempusfugit.concurrency.ConcurrentTestRunner;

@RunWith(ConcurrentTestRunner.class)
public class FuncTest {
    @Test public void test1() { …}
    @Test public void test2() { …}
    @Test public void test3() { …}
}
```

그림 5. 테스트 메소드 병렬 수행

C. Maven 설정을 활용한 병렬 수행
- Maven 설정을 사용하여 클래스 단위 또는 메스드 단위 등으로 전체 테스트를 병렬로 수행할 수 있다 [14]. 모든 테스트의 atomic 를 보장할 수 있으면 이 방법으로 간단히 병렬 수행이 가능하다.

```
<!-- 메스드 단위로 병렬 수행하고 동시에 5 개의 쓰레드를 사용한다. -->
<build>
   <plugins>
     <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-surefire-plugin</artifactId>
       <version>2.19</version>
       <configuration>
         <parallel>methods</parallel>
         <threadCount>5</threadCount>
       </configuration>
     </plugin>
   </plugins>
 </build>
```

그림 6. 병렬 수행 Maven 설정 (pom.xml)

D. 테스트 환경을 분리하여 병렬로 수행
- 테스트 환경을 2 개 이상 운영하여 기능테스트, 성능테스트 등을 병렬로 실행한다. 기능테스트와 성능테스트는 같은 환경에서 동시에 실행할 수 없으므로 각각의 테스트 환경에서 수행함으로써 전체 수행 시간을 줄일 수 있다.

4.2 테스트 간 의존성 없애기: 테스트 계정 관리 사례

테스트가 서로 영향을 주지 않기 위해서는 같은 테스트 계정이 동시에 사용되지 않아야 한다. 이상적으로는 테스트 실행 시 unique 한 테스트 계정을 생성하여 사용하면 의존성을 쉽게 없앨 수 있으나 그렇지 못한 경우도 있다. 계정 생성 기능이 제공되지

않거나 계정 생성에 시간이 걸리는 경우 테스트 계정을 미리 만들어 두고 테스트 코드에서 사용해야 한다. 본 프로젝트에서는 다수의 테스트 계정을 미리 생성하고 다음의 원칙으로 사용하여 테스트 계정으로 인한 의존성을 없앨 수 있었다.

- 하나의 테스트에서 최대 200 개의 테스트 계정만 사용할 수 있고, 테스트 계정을 1 ~ 200 의 숫자로 표현한다. 예를 들어 String accountA = AccountUtil.getAccount("11") 이다 (그림 3).
- prefix 를 매개변수로 전달하여 테스트 실행 시점에 실제 테스트 계정이 결정된다. 그림 4 처럼 mvn -DuserType=userB 로 테스트를 실행하면 AccountUtil.getAccount("11")의 실제 테스트 계정은 userB. 11 이 된다.
- Test Job 에 각기 다른 prefix 를 할당한다. 그림 4 의 TestA, TestB, TestC 는 서로 다른 userType 을 가진다.
- 동일한 테스트를 Jenkins 또는 테스트 엔지니어 PC 에서 자유롭게 수행할 수 있도록 테스트 엔지니어는 개인 전용 prefix 를 사용하여 테스트를 수행한다.

4.3 비동기 기능 테스트 고려사항

비동기 기능 테스트에서는 대기 (wait) 시간이 필요하므로 테스트 수행 시간이 길어지게 된다. 비동기 기능 테스트에서 수행 시간을 줄일 수 있는 몇 가지 방법을 소개한다.

- 동기화된 방법 활용: 스케줄링이 포함된 기능을 테스트 하는 경우 해당 기능을 셸 스크립트 (ssh 접근)로 직접 호출하여 대기 시간을 없앨 수 있다. 예를 들어 1 분 주기로 동작하는 기능을 테스트 하는 경우 최대 1 분을 기다려야 하지만 해당 기능을 실행하는 셸 스크립트를 호출하여 대기 시간을 피할 수 있다.
- 병렬 수행: 대기 시간을 피할 수 없는 경우에는 유사한 테스트들을 모아서 병렬로 수행 (tempus-fugit 사용)한다. 예를 들어 변경된 값을 조회하기 위해 cache 업데이트 (10 초 주기로 cache 가 업데이트 된다.)를 기다려야 한다면 이러한 테스트들을 동시에 수행하여 테스트 수행 시간을 단축할 수 있다.
- Timeout 관리: 비동기 기능 테스트 시 테스트 환경의 성능과 부하 등을 고려하여 timeout 을 적절하게 설정해야 한다. Timeout 이 너무 작으면 Flaky 테스트가 발생할 수 있고, 너무 많으면 테스트 실패 시 테스트 수행 시간이 길어진다.

5. Flaky 테스트 제거

Flaky 테스트 (Unreliable Test)란 테스트 대상 시스템의 문제가 아니라 테스트 코드 또는 테스트 방법의 이슈로 간헐적으로 실패하는 테스트를 말한다. Flaky 테스트가 존재하면 테스트 신뢰성이 떨어져서 실제 결함이 있는 테스트 실패를 놓칠 수도 있고 (테스트 실패가 있어도 Flaky 라 가정하고 다시 실행하여 성공하면 이전 실패를 조사하지 않는다.), 어떤 부분이 문제인지 파악하기 위해 디버깅을 불필요하게 수행해야 한다. 테스트 자동화에서는 Flaky 테스트가 없도록 관리하는 것이 중요하다.

Flaky 테스트의 주요 원인으로는 Async Wait, Concurrency, Test Order Dependence 가 있다 [15]. 본 프로젝트에서는 다음과 같은 방법으로 이러한 문제들을 회피하였다.

- Async Wait: 비동기 기능 테스트 시 Awaitility [16]를 활용하였다 (그림 3).
- Concurrency: 같은 계정이 동시에 사용되지 않도록 하였고 (4.2 참고), 동시 수행이 불가능한 테스트들을 동일한 Category 에 포함시켜 순차적으로 실행되도록 하였다.
- Test Order Dependency: 테스트 실행 순서에 상관없도록 테스트를 작성하였다.

Test Order Dependency 이슈는 shared state 가 올바르게 초기화 (cleanup) 되지 않아서 발생하기도 한다 [15]. 테스트 간에 같은 자원 (예, 테스트 계정, 시스템 설정)을 사용하는 경우 초기화가 올바르지 않으면, 같은 자원을 사용한 이전 테스트 수행으로부터 영향을 받아 다음 테스트가 실패할 수 있다. 본 프로젝트에서도 shared state 가 제대로 초기화되지 않아 발생한 Flaky 테스트가 많았다.

5.1 테스트 초기화 보완

초기화되지 못한 부분들 때문에 테스트 결과가 기대값 (Expected Result)과 다르게 나타나서 실패한 경우가 많았다. 테스트 초기화 문제를 없애기 위해서는 테스트가 실패할 수 있는 모든 조건/상황을 파악하여 테스트 초기화에 포함시켜야 한다. 많은 요소들이 테스트 결과에 영향을 줄 수 있으므로 다양한 상황을 고려해야 한다 [1]. 예를 들어 scp (Secure Copy) 기능을 테스트 하는 경우 scp 기능이 정상적이지만 테스트가 실패할 수 있는 다양한 상황이 표 3 에 나타나 있다. userA, /some/folder/fileB 가 다른 테스트에서도 사용된다면 이러한 상황이 발생할 수 있을 것이다.

또한, 초기 상태를 검증하는 코드를 추가하는 것이 좋다 [8 (p.206)]. 테스트 시작 부분 (JUnit 같은 경우 @Before)에 초기 상태를 검증하는 코드 (assert)를 추가하여 초기화가 잘못된 상태에서 테스트가 실행되는 것을 방지 할 수 있고, 초기 상태 문제점을 빨리 찾을 수 있었다.

표 3. scp 기능이 실패할 수 있는 다양한 상황

테스트 시나리오	scp fileA userA@remote:/some/folder/fileB
테스트가 실패할 수 있는 상황	- userA locked - userA 의 password 유효기간이 지났음 - userA 의 disk quota 가 full 인 상태 - /some/folder 가 존재하지 않음 - /some/folder/fileB 가 있지만 write permission 이 없음 - remote 가 변경된 후 제일 처음 접속 시 나타나는 security alert - 설정에 의한 user disabled ('DenyUsers userA' in /etc/ssh/sshd_config)

5.2 Flaky 테스트 관리

Flaky 테스트 관리에 도움이 되는 방법을 소개한다. Maven 에서 rerunFailingTestsCount 옵션 [17]을 사용하면 실패한 테스트가 자동으로 n 번 반복 실행된다. n 번 중 한 번이라도 성공하면 최종 결과가 성공으로 나타난다. 이 옵션을 이용하면 Flaky 인지 판단하는 데 도움될 것이다. Flaky Test Handler [18]를 같이 이용하면 Jenkins 에서 Flaky 테스트 목록을 별도로 조회할 수 있다.

Facebook 에서는 Flaky 테스트인지 자동으로 판단하는 시스템을 개발하여 사용한다 [19]. 이 시스템을 통해, 자동화 테스트가 등록되면 다양한 환경에서 해당 테스트가 반복적으로 수행되고, 모든 테스트가 성공해야 회귀테스트 대상에 포함된다. 반복 수행 중 실패가 발생하면 테스트 코드 작성자에게 테스트 실패가 전달되고 회귀테스트 대상에 포함되지 않는다. 이렇게 함으로써 검증된 테스트만 회귀테스트에 포함되어 Flaky 테스트가 발생할 가능성이 줄어든다.

6. 테스트 실패 원인 파악

본 프로젝트에서는 스프린트마다 시스템을 통합하였는데 통합 검증을 위한 회귀테스트에서 많은 테스트가 실패하는 일이 자주 발생하였다. 통합을 신속하게 완료하기 위해서는 어떤 테스트가 실패하였고, 또 어떤 부분에서 실패하였는지를 빨리 파악하는 것이 중요하였다. 테스트 실패 분석에 도움이 되는 Assertion 라이브러리와 Test Naming 을 소개한다. 또한 동일한 원인으로 다수의 테스트가 실패하는 경우 이를 빨리 파악할 수 있는 방법을 제시한다.

6.1 Assertion 라이브러리

Assertion 라이브러리는 JUnit 기본 assertion 메소드의 확장 기능을 제공한다. Assertion 라이브러리를 사용하면 assert 문장의 가독성을 높일 수 있고, 실패 메시지를 보다 상세히 나타나게 할 수 있고, 복잡한 assert 를 간단하게 작성 할 수 있다 (그림 7, 표 4). JUnit 에서 사용할 수 있는 도구로는 FEST Assertions [20], Truth [21], Hamcrest [22] 등이 있다.

테스트 코드	assertTrue("hello".contains("goodbye"));	assertThat("hello").contains("goodbye");
실행 결과	java.lang.AssertionError	java.lang.AssertionError: expecting:<'hello'> to contain:<'goodbye'>

그림 7. 실패 메시지 비교

표 4. FEST assertThat 예시

Example	에러 메시지
String[] list = {"hello", "goodbye", "bye", "you", "hello"}; assertThat(list).as("All should be unique.").doesNotHaveDuplicates();	java.lang.AssertionError: [All should be unique.] found duplicate(s) <['hello']> in <['hello', 'goodbye', 'bye', 'you', 'hello']>
String[] actual = {"hello", "goodbye", "bye", "you"}; String[] expected = {"hello", "goodbye", "bye", "me"}; assertThat(actual).containsAll(Arrays.asList(expected));	java.lang.AssertionError: expecting: <['hello', 'goodbye', 'bye', 'you']> to contain: <['hello', 'goodbye', 'bye', 'me']> but could not find: <['me']>
String[] list = {"hello", "goodbye", "bye", "me"}; assertThat(Arrays.asList(list)).hasSize(4).contains("bye").doesNotContain("me");	java.lang.AssertionError: expecting <['hello', 'goodbye', 'bye', 'me']> not to contain <['me']> but found <['me']>

6.2 Test Naming

어떠한 테스트인지를 알려주는 가장 첫 번째 수단이 테스트 class/method 이름이다. 표 5 와 같이 이름을 구체적으로 지으면 테스트 코드를 살펴 볼 필요 없이 테스트 시나리오를 파악 할 수 있을 것이다 [9, 23]. 테스트 이름에 시나리오와 기대값을 포함하는 것이 좋다.

표 5. 테스트 class/method 이름 예시 [9 (p.250)]

Class Name	Test Methods
OrderTest	constructorShouldThrowExceptionForNullUser()
	constructorShouldCreateValidOrderForValidUser()
PasswordValidat	shouldNotAcceptDictionaryBasedPasswords()
	shouldNotAcceptPasswordWithoutDigits()

orTest	shouldNOtAcceptShortPasswords()
Booking System	shouldAllowToBookTaleForOneHourAndMore()
	shouldDisallowToBookForLessThanHour()
	shouldAllowToCreateRecurrentReservation()

6.3 다수의 실패 파악

다수의 테스트가 실패하면 같은 원인으로 인한 실패일 수 있다. 특히, 인수 또는 End-to-End 테스트에서는 테스트 환경이나 기본 기능에 오류가 있으면 많은 테스트가 실패할 수 있다 [24]. 이 경우 같은 원인으로 실패하는 테스트를 파악할 수 있어야 나머지 개별적으로 실패하는 테스트를 쉽게 식별할 수 있다. 같은 원인으로 실패한 테스트의 실패 메시지가 비슷할 가능성이 높기 때문에 실패 메시지를 기준으로 실패한 테스트를 정렬하는 것이 도움된다. 이를 위해 JUnit 플러그인에 비슷한 실패 메시지를 가진 테스트 실패를 그룹핑하는 기능을 추가하였다 [25]. 이 기능을 이용하면 그림 8 과 같이 실패 메시지를 기준으로 실패한 테스트들을 확인할 수 있다.

Group Failed Tests By Error Message for Test Results

Error Message	Count ↑
expected:<1> but was:<0>	11
should have 132 total tests expected:<132> but was:<1007>	11
404 Not Found for http://localhost:38580/job/test-remoteapi/1/testReport/org.twia.vendor/VendorManagerTest/testCreateAdjustingFirm/api/xml?depth=-10	5
junit.framework.AssertionFailedError: null	2
Table text is missing the test duration	1
expecting many passes expected:<131> but was:<999>	1
xpath value should match for /testResult/passCount expected:<[4]> but was:<[999]>	1
java.lang.NullPointerException: null	1

그림 8. 실패 테스트 그룹핑 기능

7. Jenkins 활용

본 프로젝트에서는 테스트 수행과 테스트 Dashboard 를 위해 Jenkins 를 활용하였다.

7.1 테스트 수행과 테스트 Dashboard

우선, Jenkins 에서 수행한 테스트로만 결함을 보고하였다. 실패한 테스트의 로그와 수행시간이 기록된 Jenkins 링크를 결함 내용에 포함하여 결함 보고를 간편하게 할 수 있었다. 또한, 결함 발생 여부의 객관성을 유지할 수 있었다. 개인 PC 에서의 테스트로 결함을 보고하는 경우 시간이 지나 재현되지 않으면 실제 결함이 발생했었는지 확인하기 어려울 수 있다.

다양한 테스트의 수행을 제어하기 위해 Build Flow 를 사용하였다. 그림 9 에 Smoke Test (8.1 참고) -> 기능테스트 -> 성능테스트 -> Failover 테스트를 순차적으로 실행하는 스크립트가 나타나 있다. Smoke Test 에서는 가장 기본적인 기능을 테스트하였고, Smoke Test 가 실패하면 후속 테스트들을 진행하지 않았다. Smoke Test 가 성공하면 기능/성능/Failover 테스트를 순차적으로 (앞의 테스트가 실패하더라도) 실행하였다. 기능테스트에서는 그림 4 와 같이 또 다시 다수의 테스트가 병렬로 실행된다.

전체 테스트 결과 (성공, 실패 개수)는 그림 10 과 같이 Jenkins Dashboard [26]로 확인하였다. 실패한 테스트를 모아서 보려면 테스트 결과 (JUnit xml 파일)를 합치는 것이 필요하다 ([25]와 같이 실패 테스트를 그룹핑 하려면 테스트 결과를 모아야 한다). 이를 위해 Copy Artifact [27]를 이용하였다.

```
// 매개변수
// - test_set: 테스트 환경
// - ver: 테스트 대상 시스템 버전 (브랜치)


// Smoke Test 수행
if(params["RunSmokeTest"] == "true") {
  build("SmokeTest",
        SET: test_set,
        AWAIT_TIMEOUT:"30",
        VERSION:ver)
}

// 기능테스트 수행
if(params["RunFunctionalTest"] == "true" && test_set == "CITOSPRINT") {
   ignore(FAILURE) {
      b_it = build("FunctionalTest", SET:test_set,
                   AWAIT_TIMEOUT:await_timeout, VERSION:ver)
   }
   finalResult = b_it.result
}else if(params["RunFunctionalTest"] == "true" && test_set == "CITODEV") {
   ignore(FAILURE) {
      b_it = build("FunctionalTest_root", SET:test_set,
                   AWAIT_TIMEOUT:await_timeout, VERSION:ver)
   }
   finalResult = b_it.result
}

// 성능테스트 수행
if(params["RunPerformanceTest"] == "true" && test_set == "CITOSPRINT") {
   ignore(FAILURE) {
      b_perf = build("PerformanceTest", SET:test_set,
                   AWAIT_TIMEOUT:await_timeout, VERSION:ver)
   }

   if(finalResult == SUCCESS) {
      finalResult = b_perf.result
   }
}

// 장애테스트 수행
if(params["RunFailoverTest"] == "true") {
   ignore(FAILURE) {
      b_failover = build("FailoverTest", SET:test_set,
                         AWAIT_TIMEOUT:await_timeout, VERSION:ver, REPEAT_COUNT:"1")
   }

   if(finalResult == SUCCESS) {
      finalResult = b_failover.result
   }
}

build.result = finalResult

```

그림 9. Build Flow 스크립트 예시

Test Statistics Grid

Job	Success #	%	Failed #	%	Skipped #	%	Total #
test-score-etc	31	100%	0	0%	0	0%	31
test-score-long	26	100%	0	0%	0	0%	26
test-score-policy	22	96%	0	0%	1	4%	23
test-score-rt	38	100%	0	0%	0	0%	38
test-score-sprint-1	35	97%	0	0%	1	3%	36
test-score-sprint-2	37	95%	0	0%	2	5%	39
test-score-sprint-3	68	91%	0	0%	7	9%	75
test-score-sprint-4	30	83%	0	0%	6	17%	36
test-score-sprint-5	133	91%	1	>0%	12	8%	146
test-score-sprint-6	83	98%	0	0%	2	2%	85
test-score-sprint-7	41	91%	0	0%	4	9%	45
test-score-sprint-8	43	80%	0	0%	11	20%	54
test-score-sprint-9	49	84%	1	2%	8	14%	58
test-score-sprint-ens	27	100%	0	0%	0	0%	27
test-score-stage-1	17	74%	2	9%	4	17%	23
test-score-stage-2	25	100%	0	0%	0	0%	25
test-score-stage-3	3	75%	0	0%	1	25%	4
Total	708	92%	4	>0%	59	8%	771

그림 10. Test Dashboard 예시

테스트 기간 중 특정 테스트 케이스에 의해 시스템 장애가 발생할 수 있다. 이러한 상황을 빨리 파악하기 위해 Smoke Test 를 활용하였다. 시스템 장애가 발생하면 가장 기본적인 기능을 확인하는 Smoke Test 가 실패할 확률이 높았다. Jenkins 를 이용하여 10 분 단위로 Smoke Test 가 실행되도록 스케줄링하였고, 실패 시 이메일이 발송되게 하였다.

7.2 테스트 분석

Defect Prevention 을 위해 다양한 분석이 필요하다. 다만, Jenkins 에는 테스트 분석 기능이 많지 않다. 몇 가지 도구를 활용하여 어떤 테스트가 많이 실패하는지 정도는 파악이 가능하다.

- Test Results Analyzer Plugin [28] : 테스트 별 성공/실패 History 를 보여준다. 어떠한 테스트에서 실패가 많이 발생했는지를 알 수 있고, 이를 활용하면 Weak Point 를 찾을 수 있을 것이다. 본 프로젝트에서는 테스트 개수와 수행횟수가 많아지면서 분석 속도가 느려지고 UI 가 불편하여 활용하지는 못하였다.
- Test Stability Plugin [29] : 테스트 별 Pass Percentage 를 알려준다.
- UnitTH [30] : 다양한 테스트 분석 (Trends, 개별 테스트 성공률 등)을 제공한다. Jenkins 와 연동되지는 않는다.

8. 기타 고려사항

테스트 자동화에 도움되었던 몇 가지 방법을 추가로 소개한다.

8.1 Deployment Test and Smoke Test

테스트 기간에는 코드 수정사항이 계속 발생하여 새로운 버전을 주기적으로 배포하였다. 새로운 버전이 배포된 후 잘못된 코드 수정으로 인해 시스템 장애가 발생하거나 배포 과정에 문제가 있어 테스트할 수 없는 상황이 자주 발생하였다. 이러한 문제를 빨리 발견하는 것이 중요하여 배포 후 Deployment Test 와 Smoke Test 를 차례대로 수행하였다.

Deployment Test 에서는 설치와 실행이 올바르게 되었는지, 테스트 환경에 문제가 없는지를 확인하였다. 구체적으로는 모든 프로세스가 실행되었는지, 필요한 포트가 정상적으로 열려 있는지, 네트웍 또는 CPU 가 비정상적으로 사용되지 않는지, 디스크 공간에 여유가 있는지 등을 점검하였다. 이러한 항목에 문제가 있으면 테스트가 실패할 확률이 높고, 테스트 실패를 분석하는데 시간이 다소 소요될 수 있다.

Smoke Test 에서는 가장 기본적인 테스트를 수행하였다. 기본 기능, 테스트에서 공통으로 사용하는 기능, 테스트 초기화에 필요한 기능을 점검하였다. 빠른 수행을 위해 병렬로 수행 (tempus-fugit 이용) 하였다.

8.2 Deployable 브랜치

배포 이슈를 줄이기 위한 다른 방법으로 배포가 성공한 코드를 별도 브랜치 (Deployable 브랜치)로 관리하여 실제 테스트 환경 배포에 이용하였다. 배포 테스트 환경을 구축하여 주기적으로 최신 코드로 빌드/배포/Deployment Test/Smoke Test/회귀테스트를 수행하여 회귀테스트까지 성공한 코드를 Deployable 브랜치로 저장하였다. 단, 테스트 환경의 차이로 인해 배포 테스트 환경에서는 성공하였더라도 실제 테스트 환경에서는 문제가 발생할 수 있다.

8.3 비기능 테스트 자동화

비기능 요소 중 성능과 신뢰성이 중요하였기에 성능 테스트와 Failover 테스트를 자동화하였다. 다만, 비기능 테스트는 자동화하기 어려운 부분이 많아 정교한 테스트는 수작업으로 진행하였고 자동화가 가능한 시나리오 위주로 자동화하였다. 테스트 환경의 성능을 고려하여 부하를 설정하였고, 타이밍 이슈가 있으므로 같은 테스트를 반복해서 수행하였다.

성능 테스트 자동화의 목적은 성능을 실제 측정하기보다는 부하를 발생시켜 테스트 환경이 안정적인지를 판단하기 위해서였다. 부하를 발생시켜 모든 request 가 정상적으로 처리되었는지, 부하가 완료된 후 시스템이 정상적으로 살아있는지를 주로 점검하였다. 간접적으로는 Performance 플러그인 [31]을 이용하면 테스트 수행 시간 trend 를 알 수 있어 이전 성능 테스트와 응답시간을 비교할 수 있다 (부하가 동일해야 한다). 테스트 환경마다 성능이 다르므로 부하, timeout 등을 성능 테스트 실행 시 매개변수로 전달하는 것이 필요하였다.

Failover 테스트 자동화에서는 동일한 역할의 2 개의 서버 (그림 1 의 테스트 환경에서 엔진 부분에 여러 종류의 서버가 존재함) 중 하나가 다운되어도 서비스가 계속 가능한지를 검증하였다. 기본 시나리오는 다음과 같다. a. 부하를 특정 시간 동안 발생, b. 부하가 발생하는 중 한 대 서버 (프로세스)를 강제 다운 (ssh 로 접속하여 kill 사용), c. (optional) 부하가 발생하는 중 다운 된 서버를 재실행, d. 특정 시간 후 모든 request 가 올바르게 처리되었는지 확인. 또는, 부하가 발생하는 도중 같은 서버 2 대 모두를 다운시키고 다시 실행하는 시나리오도 포함하였다.

8.4 테스트 반복

특정 타이밍에 발생하는 결함이 있는 경우 타이밍

을 의도적으로 재현하기 어려우므로 테스트 반복을 통해서 결함 재현율을 높일 수 있다. 특정 테스트 메소드를 반복하고 싶은 경우 tempus-fugit 의 RepeatingRule, ConcurrentRule 을 활용하면 쉽게 구현할 수 있다. ConcurrentRule 을 사용하여 동시에 수행하는 경우 테스트 의존성이 없도록 하는 것이 필요하다. 예를 들어 테스트 코드에서 특정 테스트 계정을 고정하여 사용하지 않고 테스트 실행마다 다른 테스트 계정을 사용하도록 하였다. 참고로, tempus-fugit 를 사용하는 경우 몇 번을 반복하더라도 하나의 테스트로만 결과에 나타나고, 수행 도중 실패가 발생하면 즉시 테스트가 종료된다.

9. 결 론

본 논문에서 인수 테스트 자동화 시 고려해야 할 요소들로 테스트 자동화 설계, 테스트 수행 속도 높이기, Flaky 테스트 없애기, 테스트 실패 분석을 위한 방법을 설명하였고, JUnit 과 Jenkins 활용 사례를 공유하였다.

향후에는 비기능 테스트 자동화, 결함 방지를 위한 테스트 분석을 개선할 예정이다.

참고문헌

[1] TotT: Avoiding Flakey Tests. http://googletesting.blogspot.kr/2008/04/tott-avoiding-flakey-tests.html

[2] Ankit Mehta. Move Fast & Don't Break Things. In GTAC 2014.

[3] JUnit. http://junit.org/

[4] Jenkins. https://jenkins-ci.org/

[5] Matt Wynne, Aslak Hellesoy (2012). The Cucumber Book: Behaviour-Driven Development for Testers and Developers. Pragmatic Bookshelf.

[6] JBehave. http://jbehave.org/

[7] BDD. http://behaviour-driven.org

[8] Humble, Jez; Farley, David (2011). Continuous Delivery: reliable software releases through build, test, and deployment automation. Pearson Education Inc.

[9] Tomek Kaczanowski (2013). Practical Unit Testing with JUnit and Mockito.

[10] TotT: Don't Put Logic in Tests. http://googletesting.blogspot.kr/2014/07/testing-on-toilet-dont-put-logic-in.html

[11] JUnit Category. https://github.com/junit-team/junit/wiki/Categories

[12] Build Flow Plugin. https://wiki.jenkins-ci.org/display/JENKINS/Build+Flow+Plugin

[13] tempus-fugit. http://tempusfugitlibrary.org/

[14] Parallel Test Execution in Maven. http://maven.apache.org/surefire/maven-surefire-plugin/examples/fork-options-and-parallel-execution.html

[15] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov. An empirical analysis of flaky tests. In FSE 2014.

[16] Awaitility. https://code.google.com/p/awaitility/

[17] Rerun Failing Tests. https://maven.apache.org/surefire/maven-surefire-plugin/examples/rerun-failing-tests.html

[18] Flaky Test Handler Plugin. https://wiki.jenkins-ci.org/display/JENKINS/Flaky+Test+Handler+Plugin

[19] Roy Williams. Never send a human to do a machine's job: how Facebook uses bots to manage tests. In GTAC 2014.

[20] FEST Assertions. https://github.com/alexruiz/fest-assert-2.x

[21] Truth. https://google.github.io/truth/

[22] Hamcrest. http://hamcrest.org/JavaHamcrest/

[23] TotT: Writing Descriptive Test Names. http://googletesting.blogspot.kr/2014/10/testing-on-toilet-writing-descriptive.html

[24] TotT: Just Say No to More End-to-End Tests. http://googletesting.blogspot.kr/2015/04/just-say-no-to-more-end-to-end-tests.html

[25] JUnit Plugin. https://github.com/jenkinsci/junit-plugin/pull/29

[26] Dashboard View. https://wiki.jenkins-ci.org/display/JENKINS/Dashboard+View

[27] Copy Artifact Plugin. https://wiki.jenkins-ci.org/display/JENKINS/Copy+Artifact+Plugin

[28] Test Results Analyzer Plugin. https://wiki.jenkins-ci.org/display/JENKINS/Test+Results+Analyzer+Plugin

[29] Test Stability Plugin. https://wiki.jenkins-ci.org/display/JENKINS/Test+stability+plugin

[30] UnitTH. http://junitth.sourceforge.net/

[31] Performance Plugin. https://wiki.jenkins-ci.org/display/JENKINS/Performance+Plugin

입력 커버리지를 활용한 효율적인 동적 기호 실행 탐색 기법

김윤호, 김문주

KAIST 전산학부
대전광역시 유성구 대학로 291
kimyunho@kaist.ac.kr, moonzoo@cs.kaist.ac.kr

요약: 본 논문에서는 동적 기호 실행 기법을 사용해서 효율적으로 버그를 찾기 위해 입력 커버리지를 활용한 동적 기호 실행 탐색 기법을 제안한다. 입력 커버리지란 주어진 프로그램 실행 경로를 실행할 수 있는 입력 값의 전체 가능한 입력 값 대비 비율을 나타낸다. 입력 커버리지가 낮은 프로그램 실행 경로는 개발자가 평소 생각하기 힘든 코너 케이스일 가능성이 높기 때문에 제안하는 탐색 기법은 입력 커버리지가 낮은 프로그램 실행 경로를 우선 탐색하여 코너 케이스 버그를 효과적으로 찾을 수 있다. 3 개의 SIR 벤치마크 프로그램을 대상으로 실험한 결과 제안하는 탐색 기법이 깊이 우선 탐색 기법 대비 동일한 수의 테스트 케이스를 생성했을 때 1.4~1.6 배 더 많은 소프트웨어 버그를 찾을 수 있었다.

핵심어: 소프트웨어 테스팅(software testing), 동적 기호 실행(dynamic symbolic execution), 입력 커버리지(input coverage)

1. 서론

반복문을 사용하는 복잡한 프로그램의 버그를 효율적으로 찾기 위해서 빠른 시간에 분기 커버리지 달성도를 높이기 위한 동적 기호 실행[1] 탐색 기법들이 제안되었다[2-3]. 아직 실행하지 않은 프로그램 구문을 실행할 것으로 예상되는 테스트 케이스를 생성하여 구조 커버리지를 효과적으로 높이고 아직 실행하지 못한 구문에 있는 버그를 찾을 가능성을 높이는 것이다. 하지만 단순히 실행하지 못한 프로그램 구문을 실행한다고 소프트웨어 버그가 발견되는 것이 아니고 특정한 조건을 만족하는 실행 경로에서만 소프트웨어 버그가 발견되기 때문에 구조 커버리지 달성도를 빠른 시간에 높이는 테스트 케이스가 더 효율적으로 버그를 찾을 수 있는 것은 아니다.

본 논문에서는 소프트웨어 버그를 효율적으로 찾기 위해 입력 커버리지를 활용한 동적 기호 실행 탐색 기법을 제안한다. 소프트웨어 버그 중 사용자가 평소 잘 실행하지 않는 프로그램 실행 경로에서 발생하는 코너 케이스 버그는 일반적인 소프트웨어 사용 환경에서 발생하지 않기 때문에 개발자가 테스트 과정에서 발견하기 어렵다. 특히 코너 케이스를 실행할 수 있는 프로그램 입력 값 수는 전체 가능한 입력 값 수에 비해 극히 적기 때문에 일반적인 소프트웨어 사용이나 랜덤 테스팅 등으로는 탐색하기 어렵다. 제안하는 기법은 프로그램 실행 경로를 실행할 수 있는 입력 값의 수를 측정하고 실행할 수 있는 입력 값의 수가 더 적은 프로그램 실행 경로를 우선해서 탐색한다. 따라서 일반적으로 많이 실행되는 실행 경로 대신 일반적인 사용이나 개발자 테스트에서 실행하기 어려운 코너 케이스를 먼저 탐색할 수 있다. SIR 벤치마크의 3 개 프로그램을 대상으로 적용한 결과 제안하는 탐색 기법이 깊이 우선 탐색 기법 대비 동일한 수의 테스트 케이스를 생성했을 때 1.4~1.6 배 더 많은 소프트웨어 찾았다.

본 연구는 한국연구재단 한-아프리카 협력기반조성사업(NRF-2014K1A3A1A09063167), 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성 지원사업(IITP-2015-H8501-15-1012), 초소형, 고신뢰(99.999%) OS 와 고성능 멀티코어 OS 를 동시 실행하는 듀얼 운영체제 원천 기술 개발(R0101-15-0081)의 지원으로 수행되었음

2. 동적 기호 실행 기법

동적 기호 실행 기법은 동적 분석과 정적 분석 기법을 결합하여 테스트 케이스를 자동으로 생성하는 기법이다. 테스트를 생성할 대상 프로그램과 초기 테스트 케이스가 주어지면 주어진 테스트 케이스를 실행하고 실행된 프로그램 경로를 분석하여 분기문에서 어떤 조건이 수행되었는지 나타내는 경로 제약 조건식을 생성한다. 테스트 실행이 종료되면 생성된 경로 제약 조건식을 분석하여 이전에 실행되지 않은 새로운 프로그램 경로를 실행할 수 있는 테스트 케이스를 생성한다. 모든 프로그램 실행 경로를 테스트하거나 사용자가 지정한 종료 조건을 만족하면 동적 기호 실행 기법이 종료된다.

3. 입력 커버리지를 활용한 효율적인 동적 기호 실행 탐색 기법

본 장에서는 입력 커버리지와 제안하는 효율적인 동적 기호 실행 탐색 기법을 설명한다.

```
1 /* x, y, z: 심볼릭 변수 */
2 int max(int x, int y, int z){
3   if (x>=y){
4     if (x>=z)
5       return x;
6     else return z;
7   }else if (y>=z)
8     return y;
9   else return z;}
```

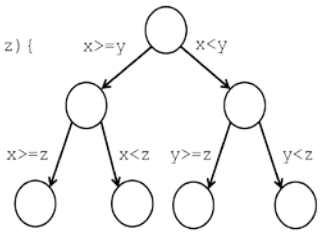


그림 1 분기문 3 개를 갖는 예제 프로그램 및 실행 경로 ([4]에서 발췌)

3.1 입력 커버리지

입력 커버리지는 프로그램의 실행 경로가 얼마나 많은 서로 다른 입력 값을 테스트하는지를 나타내는 커버리지 지표이다. 예를 들어 x, y, z 의 입력 값이 모두 1~10 사이인 그림 1 의 프로그램에서 x=y=z=1 인 테스트 케이스를 실행할 때 생성되는 경로 제약 조건식 (x>=y)∧(x>=z) 만족하는 입력 값은 모두 385 개로 전체 입력 값 1,000 개 중 38.5% 의 입력 커버리지를 갖는다고 볼 수 있다.

3.2 입력 커버리지를 활용한 동적 기호 실행 탐색 기법

소프트웨어 버그를 찾을 수 있는 테스트 케이스를 효율적으로 생성할 수 있도록 동적 기호 실행의 탐색 휴리스틱 알고리즘을 개발하였다. 동적 기호 실행에서 새로 생성할 테스트 케이스가 실행할 실행 경로의 예상 동적 커버리지를 계산하고 예상 동적 커버리지가 낮은 실행 경로를 먼저 실행하도록 테스트 케이스를 생성하였다. 따라서 제안하는 기법을 사용하면 일반적인 입력 값으로 실행하기 힘든 코너 케이스 실행 경로를 먼저 테스트할 수 있다.

4. 실험

제안하는 기법이 실제 버그를 효율적으로 탐색하는지 확인하기 위해 SIR 벤치마크[5]의 3 개 프로그램을 대상으로 실험을 수행하였다. 소프트웨어 버그 탐지 효율을 비교하기 위해 프로그램 변이 분석 (mutation analysis) 기법을 적용하여 소프트웨어 버그를 삽입하고 SIR 벤치마크의 기존 테스트 케이스의 90% 이상이 탐지 가능한 변형 프로그램을 제거하여 코너 케이스 버그를 삽입하였다. 표 1 은 실험에 사용한 프로그램과 프로그램의 크기, 테스트 수 및 변형 프로그램 수를 나타낸다.

제안하는 탐색 기법은 CREST-BV[4] 동적 기호 실행 도구를 사용하여 구현하였으며 CRES-BV 에서 제공하는 탐색 기법인 깊이 우선 탐색 기법과 제안하는 탐색 기법을 비교하였다.

표 2 는 테스트 케이스 생성 횟수를 최대 1,000, 2,000, 3,000 개로 제한했을 때 탐지한 어려운 변형 프로그램의 수를 나타낸다. 테스트 케이스 생성 횟수를 1,000 개로 제한했을 때의 replace 실험 결과를 제외하면 제안하는 탐색 기법이 깊이 우선 탐색 기법보다 더 많은 변형 프로그램을 탐지하였다. 테스트 케이스 생성 횟수를 1,000, 2,000, 3,000 으로 제한했을 때 평균적으로 제안하는 탐색 기법이 깊이 우선 탐색 기법보다 1.4, 1.6, 1.5 배 더 많은 변형 프로그램을 탐지할 수 있었다.

표 1. 실험 대상 프로그램의 LOC, 입력값 제한 및 테스트 케이스 수

이름	LOC	테스트 수	변형 프로그램 수
printtoken1	725	4130	5,624
printtoken2	569	4115	2,391
replace	563	5542	2,116

표 2. 제한된 테스트 케이스 생성 횟수 내에서 탐지된 어려운 변형 프로그램 수

이름	탐색기법	1000	2000	3000
printtoken1	제안기법	1,083	1,809	2,741
	깊이우선	608	1,107	1,605
printtoken2	제안기법	873	1,293	1,518
	깊이우선	640	933	1,117
replace	제안기법	386	1,106	1,607
	깊이우선	486	671	1,098
평균	제안기법	780.7	1,402.7	1,955.3
	깊이우선	578.0	903.7	1,273.3

5. 결 론

본 논문에서는 입력 커버리지를 활용한 소프트웨어 버그를 효율적으로 탐지하는 동적 기호 실행 탐색 기법을 제안하였다. 3 개의 SIR 벤치마크 프로그램을 대상으로 실험한 결과 제안하는 탐색 기법이 깊이 우선 탐색 기법 대비 동일한 수의 테스트 케이스를 생성했을 때 1.4~1.6 배 더 많은 소프트웨어 버그를 찾을 수 있었다.

참고문헌

[1] Cadar et al., “Symbolic Execution for Software Testing in Practice-Preliminary Assessment”, ICSE, 2011

[2] H. Seo and S. Kim, “How We Get There: A Context-Guided Search Strategy in Concolic Testing”, FSE, 2014

[3] J. Burnim and K. Sen, “Heuristics for Scalable Dynamic Test Generation”, ASE, 2008

[4] 김윤호 외 2 인, “CREST-BV: 임베디드 소프트웨어를 위한 Bitwise 연산을 지원하는 향상된 Concolic 테스팅 기법”, 정보과학회논문지: 소프트웨어 및 응용, Vol.40, No.2, 2013

[5] Do et al., “Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and Its Potential Impact”, ESE, Vol.10, No.4, 2005

계층적 구조를 고려한 소프트웨어 아키텍처 편차 측정

김희주, 이기성, 김준석, 이찬근

중앙대학교 창의 ICT 공과대학 컴퓨터공학부
서울특별시 동작구 흑석로 84
{heeju120, goory, zeldababo, cglee}@cau.ac.kr

요약: 프로그램의 원활한 유지보수를 위해 최근에 소프트웨어 리파지토리 정보로부터 아키텍처를 자동 복원 하는 연구가 많이 진행되고 있다. 또한 이를 이용한 아키텍처 품질평가 다양한 지표가 제안되었다. 본 연구는 계층 구조를 고려한 스플릿-자카드 거리(Split-Jaccard Distance) 기법을 사용하여 소프트웨어 아키텍처 편차를 측정한다. 이 기법은 복원 아키텍처와 패키지 구조간 차이를 평가하며, 이와 관련하여 다양한 오픈 소스 프로젝트를 대상으로 아키텍처 품질을 평가하여 실증적인 적용 결과를 분석하였다.
핵심어: 계층적 구조 비교, 스플릿-자카드 거리, 아키텍처 품질평가

1. 서론

소프트웨어 아키텍처 정보는 개발 초기에 명시적으로 작성되고 시스템의 변화에 따라 적절히 갱신되어야 한다. 그 중요성에도 불구하고 명시적인 소프트웨어 아키텍처와 설계 자료가 부재하거나 갱신되지 않아 확장 및 유지보수 활동에서 아키텍처 및 설계 원칙에 위반되는 경우가 많다.

최근 소프트웨어공학 커뮤니티에서는 소프트웨어의 소스 코드에 다양한 기법을 적용하여 대규모 소프트웨어의 자동적 분석과 이해를 돕는 이론과 도구에 대한 연구가 진행되었다. 그 중 소프트웨어 아키텍처 모듈-뷰 복원 연구는 다양한 소프트웨어 리파지토리 정보로부터 자동적으로 아키텍처 모듈-뷰를 복원하는 것을 목적으로 한다. 이와 같은 고 수준의 직관적 아키텍처 정보를 통해 소프트웨어 개발자 및 유지보수 엔지니어들은 보다 용이하게 해당 소프트웨어에 대한 이해와 분석을 할 수 있다.

본 연구에서는 복원된 모듈-뷰와 패키지 구조를 비교하여 아키텍처 품질 측정방법으로 활용하고자 한다. 기존의 유사 연구는 평면구조의 아키텍처만을 대상으로 했던 반면 본 연구는 계층적 구조를 반영하여 보다 소프트웨어 특성을 반영하는 장점을 가진다. 또한 제안하는 방법의 실증적 평가를 위해 오픈 소스 프로젝트를 대상으로 사례연구를 진행하였다.

2. 관련 연구

2.1 아키텍처 모듈-뷰 복원

아키텍처 모듈-뷰란 구현된 유닛들을 기능적으로 유사한 모듈들로 묶어 표현한 것이다. 모듈-뷰를 통해 복잡한 구조의 소프트웨어를 그림 1 처럼 비슷한 역할의 클래스와 파일들을 묶어 이해하기 쉽게 표현할 수 있다.

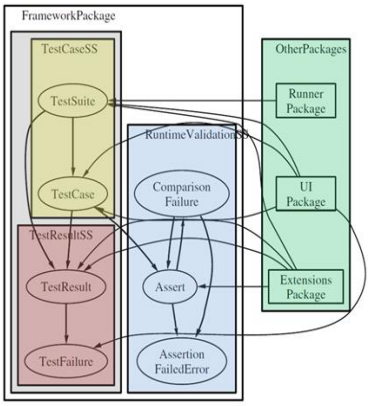


그림 1 복원된 아키텍처 모듈-뷰

소프트웨어 리파지토리에서 다양한 정보를 활용하여 해당 소프트웨어의 모듈-뷰를 만들어 내는 것을 소프트웨어 아키텍처 모듈-뷰 복원이라고 한다. 이러한 복원을 통해 소프트웨어 개발자 및 유지보수 엔지니어들은 소프트웨어 아키텍처를 보다 직관적이고 쉽게 이해, 분석을 할 수 있다.

복원 기법으로는 Bunch[1], ACDC[9], WCA[3], LIMBO[4] 등이 있는데, 본 연구에서는 Bunch 를 사용한다. Bunch 는 언덕 오르기 알고리즘을 사용한다.

아키텍처 복원 기술과 평가 기법을 활용하여 소프트웨어 품질을 측정하는 연구가 있었다. Zhu 등[7]은 MojoFM 을 이용한 SiMo 라는 새로운 품질평가 지표를 제안하였다.

그러나 해당 연구에서는 소프트웨어의 계층성을 고려하지 못하고 있다. 따라서 우리는 사전 연구에서 제안된 스플릿-자카드 거리[2]를 이용하여 복원된 뷰와 패키지 구조간 계층적 비교를 수행한다.

2.2 스플릿-자카드 거리

LEE 등[2]은 소프트웨어의 계층적 구조를 고려한 비교 메트릭으로 스플릿-자카드 거리를 제안하였다. 스플릿-자카드 거리는 두 개의 비정렬(Unordered) 트리간 차이점을 측정한다. 계층적 구조를 고려하기 위하여 스플릿[6] 개념을 사용하는데, 스플릿은 두 노드의 공통된 조상들 중 두 노드와 가장 가까운 노드를 의미한다. 스플릿-자카드 거리는 두 트리의 모든 노드 쌍을 스플릿 관점에서 차이를 계산해 이를 0 과 1 사이로 정규화하여 1 에서 해당 수치를 뺀 거리 메트릭이다. 본 연구는 이 메트릭을 이용하여 소프트웨어

의 품질 평가를 위해 소프트웨어 아키텍처의 편차 측정을 제안한다.

3. 소프트웨어 아키텍처 편차 측정

품질 평가를 위한 소프트웨어 아키텍처 편차 측정의 개요는 다음과 같다.

- **1 단계: 구조적 정보 추출**

Understand[8]를 이용하여 소프트웨어 리파지토리에서 파일간 의존성을 추출한다. 추출한 데이터는 노드와 엣지로 표현되며 각 노드는 자바파일, 엣지는 상속, 메소드 호출, 변수 참조 등에 의해 정의된다.

- **2 단계: 아키텍처 모듈-뷰 복원**

Understand 로 얻은 결과를 이용하여 Bunch 를 통해 유사한 노드끼리 클러스터링을 수행한다. 이 경우, 응집도와 결합도를 기반으로 클러스터링이 수행되며, 결과는 비정렬 트리로 저장된다.

- **3 단계: 소프트웨어 품질 평가**

복원 결과와 패키지 구조간 스플릿-자카드 거리를 측정한다. 결과값으로 나오는 수치를 통해 소프트웨어 품질을 평가할 수 있다. 도출된 거리가 0 에 수렴할수록 두 트리가 유사함을 의미하며, 패키지 모듈성 관점에서 아키텍처 품질이 좋음을 의미한다.

4. 사례 연구

우리는 앞서 제안한 편차 측정 방법을 오픈 소스 대상으로 실험하였다. 프로젝트 별로 계산된 스플릿-자카드 거리 결과는 그림 2 와 같다.

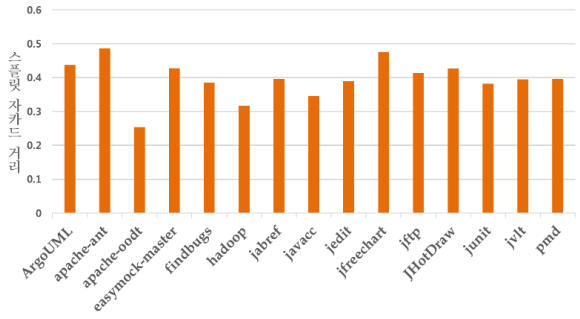


그림 2 패키지 구조와 복원 결과간의 스플릿-자카드 거리 측정 결과

실험 결과, Apache-oodt 와 Hadoop 의 품질이 제일 좋은 것으로 나타났다. 품질이 가장 좋지 않은 것은 약 0.4870 의 값을 가지는 Apache-ant 로 나타났다. 또한, 단일 프로젝트를 대상으로 다음과 같은 15 개의 버전 별로 실험을 수행해보았다.

결과적으로 Apache-ant 1.6.3 과 1.7.1 에서 품질이 좋아진 것을 알 수 있었다. 전체적인 관점으로 보았을 때, 이 프로젝트는 진화하면서 미세하게 거리가 커지고 있어 프로젝트 규모가 점점 커지면서 구조적 품질저하가 일어나고 있는 것으로 관측된다.

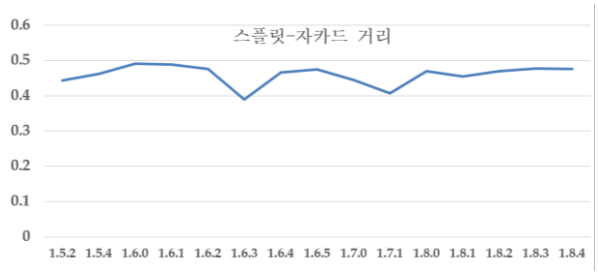


그림 3 버전 별 스플릿-자카드 거리 그래프

5. 결론 및 향후 연구

우리는 본 연구에서 소프트웨어의 계층적 구조를 고려한 아키텍처 편차를 측정하였다. 스플릿-자카드 거리를 이용하여 패키지 구조와 복원 결과를 비교하였고, 품질평가 지표로 사용하였다. 또한 사례 연구를 진행하여 실제 소프트웨어에서의 적용 가능성을 확인하였다. 향후 연구로는 계층적 복원을 지원하는 다양한 클러스터링 알고리즘을 도입하고자 한다.

감사의 글

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 서울어코드활성화사업(R0613-15-1205), 대학 ICT 연구센터육성 지원사업(IITP-2015-H8501-15-1012)과 한국연구재단 기초연구사업(과제번호 NRF-2014-005519)의 지원을 받았습니다.

참고문헌

[1] B. S. Mitchell, et al.., "On the automatic modularization of software systems using the Bunch tool," IEEE Transactions on Software Engineering, Vol. 32, No. 3, pp. 193-208, 2006.

[2] K.-S. Lee, et al., "Split-Jaccard Distance of Hierarchical Decompositions for Software Architecture," IEICE TRANSACTIONS on Information and Systems, Vol. E98-D, no. 3, pp. 712-716, 2015.

[3] O. Maqbool, et al., "The weighted combined algorithm: a linkage algorithm for software clustering," Proc. of the Conference on Software Maintenance and Re-engineering(CSMR), pp. 15-24, 2004.

[4] P. Andritsos and V. Tzerpos, "Information-theoretic software clustering," IEEE Trans. on Software Engineering, Vol. 31, no. 2, pp. 150-165, 2005.

[5] P. Jaccard, "The distribution of the flora in the alpine zone," New Phytologist, vol.11, no.2, pp.31-60, 1912.

[6] Q. Zhang, et al., "Split-Order Distance for Clustering and Classification Hierarchies," Scientific and Statistical Database Management, Vol. 5566, pp. 517-534, 2009.

[7] T. Zhu, et al., "Monitoring Software Quality Evolution by Analyzing Deviation Trends of Modularity Views," Proc. of Working Conference on Reverse Engineering(WCRE), pp. 229-238, 2011.

[8] Understand, http://www.scitools.com

[9] V. Tzerpos, et al., "ACDC: an algorithm for comprehension-driven clustering," Proc. of Working Conference on Reverse Engineering(WCRE), pp. 258-267, 2000.

[10] Y. Tzerpos, et al., "MoJo: A distance metric for software clusterings," Proc. IEEE 6th Working Conference on Reverse Engineering, pp. 187-193, 1999.

산업용 로봇 소프트웨어에서 재사용성 확보를 위한 옵저버 패턴 적용

하상범, 유철중

전북대학교 소프트웨어공학과
전북 전주시 덕진구 백제대로 567
{sangb-ha, cjyoo}@jbnu.ac.kr

요약: 전통적으로 생산용 기계에 비해 산업용 로봇은 유연하게 적용할 수 있도록 설계된다. 산업화의 가속화에 따라 많은 산업용 로봇들은 인간의 일을 대체할 산업 생산에 적용되고 있다. 현재 산업용 로봇은 서로 다른 벤더에서 생산한 구성요소들을 조립하여 생산하는데, 구성요소들을 제어하고 상호작용하기 위해서는 벤더에서 제공하는 라이브러리를 이용해 산업용 로봇 소프트웨어를 개발한다. 하지만 산업용 로봇 소프트웨어는 현재 특정한 규격 없이 개발되고 있어 재사용성의 부족한 실정이다. 앞의 산업용 로봇 소프트웨어의 문제점을 해결하기 위해 공학적인 설계 방법이 필요하다. 따라서 본 논문에서는 특정한 규격 없이 개발되는 산업용 로봇소프트웨어에 구성요소 상호작용을 구현하는데 있어 옵저버 패턴을 적용하여 재사용성을 확보하도록 한다.

핵심어: 산업용 로봇, 산업용 로봇 소프트웨어, 디자인 패턴, 옵저버 패턴

1. 연구배경

전통적인 생산용 기계에 비해 산업용 로봇은 유연하게 적용할 수 있도록 설계된다[1]. 산업화의 가속화로 점점 더 많은 로봇들은 인간의 일을 대체할 산업 생산이 적용 되고 있다[2]. 현재 산업용 로봇의 각 구성요소들은 서로 다른 벤더에서 생산된 카메라, 모터들이 조립되며 벤더들에서 제공하는 라이브러리를 통해 각 구성요소들을 제어한다. 산업용 로봇 소프트웨어는 각 벤더에서 제공하는 라이브러리를 이용해 각 구성요소를 제어할 수 있도록 개발된다.

하지만 산업용 로봇 소프트웨어는 현재 특정 규격 없이 개발되어 재사용성 부족한 실정으로 산업용 로봇 소프트웨어 설계와 구현에 있어 공학적인 설계 방법이 필요하다[3]. 따라서 본 논문에서는 산업용 로봇 소프트웨어에서 구성요소 상호작용을 구현하는데 있어 옵저버 패턴을 적용하여 재사용성을 확보한다. 옵저버 패턴은 객체의 상태변화에 따라 의존관계가 있는 객체의 메소드들이 호출 되는 것으로 산업용 로봇에서 구성요소들간 상호작용을 구현에 적용하기 적합하다.

본 논문은 다음과 같이 구성되어 있다. 2 장에서는 디자인 패턴에서 옵저버 패턴과, 옵저버 패턴이 적용된 사례에 대해 다루고, 3 장에서는 본 논문에서 제안하는 산업용 로봇 소프트웨어 옵저버 패턴 적용에 대해 논의하고 4 장에서는 결론을 맺는다.

2. 옵저버 패턴과 관련 연구

옵저버 패턴의 의도는 객체 사이에 일대 다의 의존 관계를 정의 해두어, 어떤 객체의 상태가 변할 때 그 객체의 의존성을 가진 다른 객체들이 그 변화를 통지 받고 자동으로 갱신 될 수 있게 만든다[4].

그림 1 은 일반적인 옵저버 패턴의 구조이다.

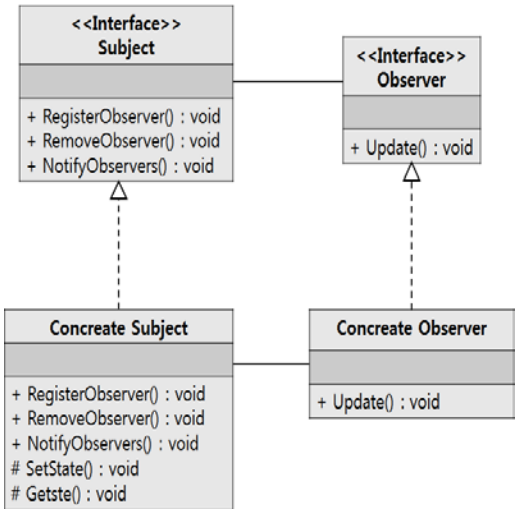


그림 1. 옵저버 패턴 구조

Subject 클래스에서 Observer 객체를 등록, 제거할 수 있는 리스트를 가지고 있으며, 모든 Observer 에 Subject 의 상태를 갱신하는 Update()를 호출하는 메소드를 가지고 있다. 그리고 Subject 와 Observer

는 각 구체화 할 수 있도록 상속 받는 각각의 Concrete 클래스가 있으며 세부적인 상태에 대해 추가 구현이 가능하다.

소프트웨어의 기존 문제점을 해결하기 위해 옵저버 패턴을 적용한 여러 연구가 진행 되었는데, Xinyun Liu 의 연구팀에서는 프로그램의 상태 변화에 있어 자동적으로 알려주는 WatchDog 설계를 위해 옵저버 패턴과 유한 상태머신을 이용한 메소드를 도출하는 기법을 제시하였고[5], Farzin Karimi 의 연구에서는 소프트웨어 제품의 디자인이나 모델의 정확성 또는 건전성을 확인하는데 사용하는 가상환경인 테스트벤치 환경에 옵저버 패턴을 적용시켜, 동적으로 객체에 새로운 제약조건을 추가할 수 있도록 제시 하였다[6].

위의 연구들은 옵저버 패턴을 이용하여 상태 변화의 관련 기능들을 구현하는 동시에 재사용성을 확보하였다. 따라서, 본 논문에서는 산업용 로봇의 구성요소들 간 상태변화에 대해 갱신이 되어야 하는 기능들을 옵저버패턴을 적용해 재사용성을 확보하도록 한다.

3. 옵저버 패턴 적용

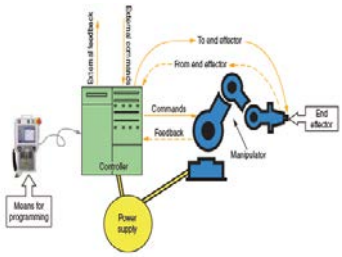


그림 2. 산업용 로봇 기본 구성요소

그림 2 는 산업용 로봇 기본적인 구성요소를 분류한 것이다. 각 구성요소는 모터(Motor), 머니퓰레이터(Manipulator), 동력공급장치(Power Supply), 말단장치(End Effector), 제어기(Controller)로 나눠진다. 각 분류된 구성요소들은 옵저버 패턴에서 Subject 로부터 상속받아 구현되고, 옵저버는 각 구성요소가 상태 변화가 감지가 되었을 때 상태 변화에 맞춰 갱신, 상호작용에 관련된 기능들로 구현된다.

또한 구성요소들은 옵저버 패턴을 Subject 로 상속받아 구현하고, 구성요소들을 제어하기 위해 내부적으로는 외부라이브러리를 멤버 변수로 가지고 있고, 메소드들이 정의 되어야 한다. 아래 코드는 Motor 클래스를 구현한 코드로 구현언어는 C#을 사용하였다.

```
//모터 제어를 위한 모터 클래스
public class Motor
{
    private VenderMotor mVenderMotor;
    private Thread mThreadReadPosition;

    public Motor()
    {
       mVenderMotor = new VenderMotor();
       mThreadReadPosition =    new Thread
                  (ThreadReadPosition());
        mThreadReadPosition.Start();
    }

    public void Move(double position)
    {
        mVenderMotor.move(position);
    }

    public void Stop()
    {
        mVenderMotor.stop();
    }

    public void Return()
    {
        mVenderMotor.init();
    }

    public void ThreadReadPosition()
    {
       double currentPos = 0;
       while(true)
       {
            currentPos =
                 mVenderMotor.getCurrentPostion();
       }
    }
}
```

앞의 코드상에서 멤버로 있는 VenderMotor 는 산업용 로봇에 부착되어 있는 모터를 제어하기 위한 외부 라이브러리다.

구현된 구성요소에 옵저버 패턴을 적용을 위해 구성요소 클래스들은 옵저버 패턴에서 Subject 인터페이스를 상속을 받는다. 또한 실시간으로 감지하여 구성요소의 상태변화를 갱신을 위한 기능들을 Observer 인터페이스에서 상속 받아 구현한다. 구성요소 갱신에 관련된 메소드는 Observer 로부터 상속받은 메소드 Update()안에 정의한다.

아래 그림 4 는 Motor 의 현재 위치가 특정 위치에 도달하였을때, 다른 구성요소인 카메라에게 명령을 내릴 수 있

는 기능을 옵저버 패턴을 적용하여 나타낸 클래스 구조다.

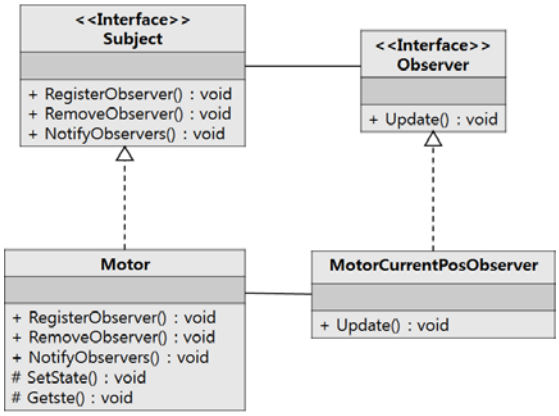


그림 4. 옵저버 패턴 적용된 Motor 클래스와 예시

그림 4 의 Subject 로부터 상속받아 구현한 Motor 클래스를 코드를 C#으로 나타내면 다음과 같다.

```
//옵저버 패턴의 subject 를 상속한 Motor 클래스
public class Motor : Subject
{
    Private VenderMotor mVenderMotor;
    Private Thread mThreadReadPosition;
     //쓰레드를 이용하여 모터온도 체크

    Public Motor()
    {
       mVenderMotor = new VenderMotor();
       addObserver(new
                    MotorCurrentPosObserver(this));
    //옵저버 추가

        mThreadReadPosition =
             new Thread
                  (ThreadReadPosition());
        ThreadReadPosition.Start();
    }

     public void Move(double position)
    {
        mVenderMotor.move(position);
    }

    public void Stop()
    {
        mVenderMotor.stop();
    }

    public void Return()
    {
        mVenderMotor.init();
    }

    public void ThreadReadPosition()
    {
       double currentPos = 0;
       while(true)
       {
            currentPos =
                 mVenderMotor.getCurrentPostion();

            If( True )
            {
                 NotifyObservers();
                 //등록된 옵저버에게 알림
                 …
            }
       }
    }
}
```

위에 구현된 Motor 클래스는 Motor 객체 생성시 Motor 구성요소를 제어하기 위해 VenderMotor 의 멤버 변수를 생성하고, Subject 를 상속받은 Motor 클래스이므로 Motor 의 현재위치가 특정위치에 도달하였을때, 다른 구성요소인 카메라에게 명령을 내릴 수 있도록 MotorCurrentPosObserver 를 리스트에 추가하고 생성한다.

그리고 또한 현재위치를 계속해서 Motor 외부 라이브러리부터 받아와야 하므로 별도의 쓰레드를 두어 ThreadReadPosition() 메소드 안에서 현재위치를 확인한다. 만약 현재위치가 특정한 위치에 도달하였다면 등록된 옵저버들에게 상태변화를 NotifyObservers() 메소드를 호출한다.

Observer 로부터 상속받은 MotorCurrentPosObserver 클래스를 C#으로 나타내면 다음과 같다.

```
Public class MotorCurrentPosObserver: Observer
{
   private Motor mMotor;
   private Camera mCamera;

   public MotorCurrentPosObserver (Motor motor,
Camera camera)
   {
       mMotor = motor;
       mCamera = camera;
   }

   //Motor 클래스로부터 호출됨
   Public override void Update()
```

```
  {

      mMotor.Stop();
      mCamera.Shot();
  }
}
```

MotorCurrentPosObserver 는 Observer 로부터 상속을 받아 구현하고, 멤버로는 Motor 와 Camera 를 가지고 있다. 이는 Motor 클래스로부터 현재위치가 특정한 위치가 되었을 때 호출이 되면 모터가 정지를 하고 카메라는 사진을 찍어야 하기 때문이다.

4. 결론

본 논문에서는 산업용 로봇 소프트웨어에 상태 변화가 발생 했을 때 즉각적인 응답과 재사용성 확보하기 위해 디자인패턴 중 옵저버 패턴을 이용하여 구현하였다.

산업용 로봇 소프트웨어 개발에서 특정한 규격이 없이 개발되는 문제점을 옵저버 패턴을 이용해 해결했을 뿐만 아니라 재사용성을 확보하였다.

향후 연구로는 산업용 로봇 소프트웨어에서 옵저버 패턴 적용 시 구성요소로부터 옵저버를 도출함에 있어 명확하지 않았다. 이러한 문제점을 해결하기 위해서 각 객체의 상태와 함께 객체 상태를 변화를 유발하는 이벤트와 동작을 정의하는 상태차트 다이어그램을 이용하여 명확하게 옵저버를 도출하는 기법에 대한 연구가 필요하다.

사사

본 논문은 중소기업청에서 지원하는 2015 년도 산학연협력 기업부설연구소 지원사업(NO.C0267942)의 연구수행으로 인한 결과물임을 밝힙니다.

참고문헌

[1] U.Hagn, M. Nickl, S. Jorg, G.Passing, T. Bahls, A. Nothhelfer, F Hacker, L. Le-tien, A. Albu-Schaffer, R. Konietschoke, and G. Hirzinger, “The DLR MIRO: A versatile lightweight robot for surgical applications,” Industrial Robot, vol. 35, no.4 pp. 324- 336, 2008.

[2] G. Ma, G-Y shen. “The presnet situation and developing trend of industrial robot”. Modular Machine Tool & Automatic Manufacturing Techique. 48-50, 2002(3).

[3] H. Bruyninckx. “Open Robot Control Software: The OROCOS Project”. Proceedings of the 2001 IEEE International Conference on Robotics & Automation, 21-26, 2001

[4] Liu, Xinyun, Shihang Chen, and Wenqiang Song. "A design and implementation of watchdog based on observer pattern and finite state machine." Reliability, Maintainability and Safety (ICRMS), 2014 International Conference on. IEEE, 2014.

[5] Liu, Xinyun, Shihang Chen, and Wenqiang Song. "A design and implementation of watchdog based on observer pattern and finite state machine." Reliability, Maintainability and Safety (ICRMS), 2014 International Conference on. IEEE, 2014.

[6] Karimi, Farzin. "Applications of decorator and observer design patterns in functional verification." High Level Design Validation and Test Workshop, 2008. HLDVT'08. IEEE International. IEEE, 2008.

상호 정보량을 이용한 소프트웨어 아키텍처 모듈-뷰 복원 평가

허민재, 이찬근

중앙대학교 컴퓨터공학부
서울 동작구 흑석로 84
elfel.cau@gmail.com, cglee@cau.ac.kr

요약: 기존의 상호 정보량(Mutual Information)은 패턴인식, 클러스터링 비교, 등에 사용되어 왔다. 그러나 소프트웨어 아키텍처 복원 평가에는 매우 제한적으로 사용되었다. 현재까지 기존 연구들은 소프트웨어 아키텍처 모듈-뷰 복원 평가에서 MoJo 계열을 보편적으로 사용하고 있다. 본 연구에서는 MoJo 계열과 상호 정보량의 각 특성에 대해 기술하고 이 특성을 바탕으로 MoJo 계열과 함께 상호 정보량의 소프트웨어 아키텍처 모듈-뷰 복원 평가 실험을 진행한다. 이를 통해 상호 정보량이 MoJo 계열과 함께 사용할 수 있는 평가 메트릭임을 제안한다.

핵심어: 소프트웨어 아키텍처 모듈-뷰, 복원 평가, 패턴 인식, MoJoFM, 상호 정보량

1. 서 론

소프트웨어 아키텍처 복원 연구는 문서화가 잘 되어있지 않고 변화가 잦은 소프트웨어를 분석하기 위해 진행되어 왔다. 이러한 복원 연구는 다양한 방법이 제시되고 있으며[1]. 복원된 소프트웨어 명세는 개발 및 유지보수에 기여하고 있다. 그 중에서도 모듈-뷰 복원에 사용되는 방법 역시 다양한 알고리즘이 있으며 각각의 알고리즘마다 강점을 나타내고 있다. 이 강점을 살펴보고 소프트웨어의 특성에 따라 사용될 알고리즘을 나눌 때 사용하는 것이 모듈-뷰 복원 평가 방법이다. 기존의 소프트웨어 아키텍처 모듈-뷰 복원 평가는 서로 다른 알고리즘으로 복원된 소프트웨어 즉, 클러스터링 된 소프트웨어들의 결과를 어떻게 평가할 것인가에 초점이 맞춰져 있다. 일반적으로, 정답지(Ground truth)와 클러스터링 알고리즘을 통해 복원한 소프트웨어 결과의 유사도를 비교하는 방법으로 사용된다. 이런 측정 방법들에는 정밀도(precision)와 재현율(recall), EdgeSim, MoJo 계열 등을 이용한 방법이 있다[2]. 이들은 소프트웨어의 특성을 반영한 것으로 엔티티들의 집합인 결합도(Coupling), 응집도(Cohesion)를 반영하고 있거나 엔티티간의 거리를 기반으로 유사도를 측정한다. 엔티티란 소프트웨어를 구성하는 정보 단위로 복원 의도에 따라 패키지, 클래스, 메소드, 함수 등으로 정해진다. 본 논문에서는 각 클래스를 엔티티로 취급한다. 이처럼 다양한 측정 방법이 있지만 현재 가장 널리 사용되는 측정 메트릭은 MoJo 계열의 MoJoFM 이 대부분이다. 그러나 조금 더 다양한 측정 메트릭을 도입하려는 연구가 진행되고 있으며, 본 연구에서는 정보 검색(Information Retrieval), 패턴 인식 등에서 사용되고 있는 상호 정보량을 이용한 모듈-뷰 복원 평가 메트릭을 제안한다. 상호 정보량은 두 집합간의 유사도를 다루고 있으며 이는 클러스터링 결과 유사도를 비교하는 것과 맥락을 같이 한다고 판단하여 연구를 진행 하였다. 본 논문의 기여는 다음과 같다.

- 소프트웨어 아키텍처 모듈-뷰 평가 영역에서 제한적으로 사용되었던 상호 정보량을 복원 평가 메트릭으로 사용할 수 있음을 보인다.
- 가장 널리 쓰이고 있는 MoJoFM 과 상호 정보량의 연관성을 보이고, 상호 정보량이 복원 평가에서 어떤 강점을 보이는지 보인다.

본 논문의 구성은 다음과 같다. 2 절에서 MoJo 계열과 상호 정보량의 관련 연구에 대해 기술하며, 3 절에서 MoJoFM 과 상호 정보량을 이용한 복원 평가를 진행하고 상호 정보량이 모듈-뷰 복원 평가 메트릭으로 사용될 수 있음을 기술한다. 마지막 4 절에서 결론과 향후 연구 방향을 기술한다.

2. 관련 연구

2 절에서는 본 연구의 비교에서 사용된 MoJo 계열과 상호 정보량에 대해서 다룬다. 1 절에서 언급한 바와 같이 MoJo 는 다양한 시리즈로 제안되어 MoJo 계열이라고 불리운다. 상호 정보량 역시 MI 로 단순히 사

용되기도 하지만 다양한 시리즈로 제안되어 있다. 이 절에서는 소프트웨어 특성을 가장 잘 반영하는 MoJo 계열의 MoJo 와 MoJoFM 에 대해 기술하고, MI 의 특징과 적용 분야에 대해 기술한다.

2.1 MoJoFM

MoJoFM 은 V. Tzerpos 가 제안한 MoJo 의 확장 메트릭이다. MoJo 의 원리는 소프트웨어의 특성을 반영한 거리 기반 측정 메트릭이다. 두 클러스터 결과 간의 유사도를 엔티티의 이동(Move)과 결합(Join)의 횟수가 가장 적을수록 유사한 클러스터 결과라고 반영한다. MoJo 의 수식은 다음과 같다[3].

$$\boldsymbol{MoJo}(\boldsymbol{A},\boldsymbol{B}) = min(mno(A,B), \mathrm{mno}(B,A)) \quad \cdots\cdots \quad (1)$$

수식 (1)에서 A 와 B 는 비교하고자 하는 클러스터링 결과이며 mno(A,B)는 A 와 B 의 클러스터간 이동이 최소값인 경우를 말한다. 그러나 MoJo 의 경우 A 에서 B 또는 B 에서 A 의 이동 횟수 결과 중 값이 적은 결과만을 반영하며, MoJo 의 최대값이 제한되지 않았다. 그렇기 때문에 그들은 보다 개선된 MoJoFM(MoJo eFfectiveness Measure)을 추후 제안했다[4]. 수식은 다음과 같다.

$$\boldsymbol{MoJoFM}(\mathrm{M}) = \left(1 - \frac{mno(A,B)}{max(mno(\forall A,B))}\right) \times 100\% \quad \cdots\cdots \quad (2)$$

MoJoFM 은 이동과 결합을 사용한다는 점에서 MoJo 와 다르지 않지만 $mno(\forall \mathrm{A}, \mathrm{B})$를 나눠줌으로써 MoJo 에서 발생할 수 있는 발산 문제를 해결하였다. 이는 임의의 클러스터링 결과와 B 에 대한 mno 최대값 즉, $mno(\mathrm{A},\mathrm{B})$ 보다 값이 적은 경우가 발생할 때 이를 나누어 정규화 하였다. MoJoFM 은 새롭게 제안되는 다양한 소프트웨어 복원 알고리즘[5, 6] 등의 평가에 사용되고 있다. 특히 정답지와의 비교에 가장 대표적인 측정 메트릭으로 사용된다.

2.2 상호 정보량(Mutual Information)

상호 정보량은 정보 이론, 정보 검색(IR), 패턴 인식분야에서 사용되었다. 일반적으로 사용되는 상호 정보량의 원리는 다음과 같다. A 와 B 두 개체가 있을 때, A(또는 B)의 정보량이 B(또는 A)의 정보량에 전달되는 정도이다. 이때, A 와 B 가 서로 유사성이 없는 독립된 개체라면 0 으로, 완벽히 같은 종속된 개체일 때 1 로 나타난다. 상호 정보량의 수식은 다음과 같다.

$$\mathrm{MI}(\mathrm{A},\mathrm{B}) = \sum_{i=1}^{R}\sum_{j=1}^{C} P(i,j)\log\frac{P(i,j)}{P(i)P\prime(j)} \cdots\cdots \quad (3)$$

이때, P(i)는 클러스터 A_i에서 i 요소가 발생될 확률, $P'(j)$는 B_j에서 j 요소가 발생할 확률을 말한다. MI 이는 이러한 A, B 두 개체 전체의 합을 반환하므로 패턴 인식 분야를 예로 들었을 때, 손상된 이미지의 복구 등에 사용되며 대표적으로 의학 분야에서 사용되고 있다[8]. 최근에는 MI 를 이용한 소프트웨어 클러스터링 연구[9]도 제안이 되고 있어, 다양한 분야에서 상호 정보량이 사용됨을 알 수 있다.

그러나 앞서 기술하였듯 A 와 B 의 정보량 전달을 통해 유사성을 도출하는 상호 정보량은 MoJoFM 과 유사한 특징을 가지고 있지만 클러스터링 알고리즘에 적용 했을 뿐, 소프트웨어 아키텍처 모듈-뷰 복원 평가 메트릭으로는 매우 제한적으로 사용되고 있다[7].

3. 실험 및 평가

이 절에서는 2.2 절에서 기술한 바와 같이 MoJoFM 과 상호 정보량의 특징을 데이터로 관찰하고 MoJo FM 과 함께 또는 특정 부분에서 상호 정보량을 사용할 수 있음을 보인다.

3.1 실험 데이터

표 1 실험에 사용된 네 개의 오픈소스 프로젝트

	Apache_oodt 0.2	ArchStudio	Bash 1.14.1	Hadoop_core 0.19.0
Entities	875	412	126	627

실험에 사용되는 데이터는 MoJoFM 과 MI 의 특성을 관찰하기 위해 정답지가 존재하는 네 개의 오픈소스 프로젝트로 진행하였다. 이때 표 1 의 엔티티는 각 오픈 소스 프로젝트의 클래스 개수이다. 또한 각 오픈소스 소프트웨어의 클러스터링 알고리즘은 Border flow[10], BunchNAHC[5], BunchSAHC[5], Gephi[11] 을 사용하였다. 각각의 알고리즘은 그래프 클러스터링, 구조적 소프트웨어 클러스터링을 기반으로 한다.

3.2 실험 결과 및 분석

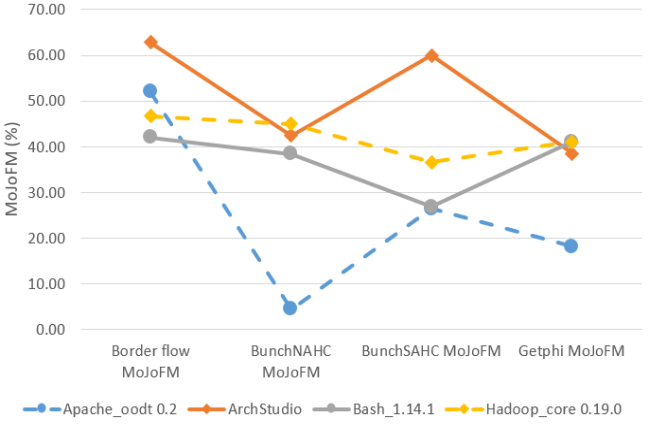


그림 1. 네 개 프로젝트의 MoJoFM 복원 평가 결과

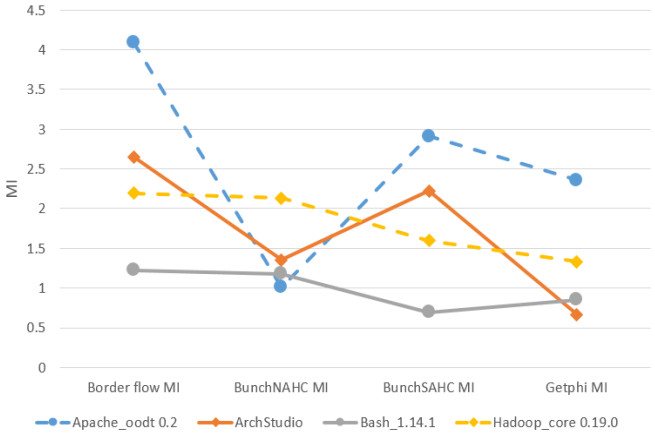


그림 2. 네 개 프로젝트의 MI 복원 평가 결과

우선 그림 1 의 MoJoFM 복원 결과를 보았을 때 가장 우수한 복원 결과는 62.86%의 Borderflow 를 이용한 ArchStudio 이다. 그리고 그림 2 의 MI 결과를 보았을 때 가장 우수한 복원 결과는 4.1 로 Borderflow 를 이용한 Apache_oodt 임을 알 수 있다. 하지만 그림 1 과 그림 2 의 결과, 각 프로젝트의 복원율 변동은 비슷하지만 가장 좋은 복원 알고리즘은 다르게 반영하는 것을 알 수 있다. MI 의 경우 수식 (3)에서 보는 것처럼 모든 요소의 합 즉 각 클러스터에 속한 엔티티가 전달되는 양의 총 합이므로 표 1 의 엔티티 개수에 따라 평가되는 것을 알 수 있다. 따라서 MI 의 정규화 식인 NMI(Normalized Mutual Information)을 통해 같은 방법으로 결과를 도출 하였다.

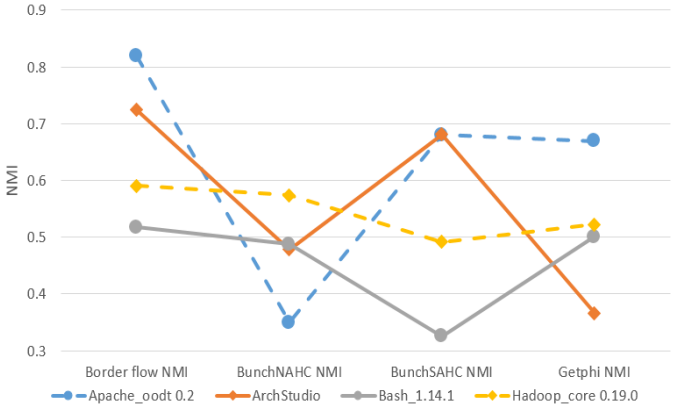


그림 3. 네 개 프로젝트의 NMI 복원 평가 결과

NMI 를 이용한 결과는 MI 를 이용한 결과보다 평가 결과를 MoJoFM 과 비슷하게 반영하고 있음을 알 수 있다. 그러나 여전히 MoJoFM 에서 가장 복원이 잘 되었다고 반영한 결과와는 차이가 있음을 볼 수 있다. 따라서 해당 문제를 살펴보기 위해 다음과 같은 추가 실험을 진행하였다.

1) 하나의 알고리즘(Agglomerative Hierarchical)을 이용하여 클러스터의 개수가 1 개부터 엔티티의 개수 n 까지 가지는 클러스터링 결과 추출
2) 두 개의 프로젝트에 대해 1)의 과정을 진행 후, n 개의 결과를 정답지와 비교하여 MoJoFM 과 MI 시리즈의 특성 분석

이 실험에 사용된 데이터는 Apache_oodt 0.2 와 Hadoop core 0.19.0 이며 각각의 엔티티는 1016 개, 606 개이다. 먼저, Apache_oodt 0.2 의 결과는 그림 5 와 같다. MoJoFM 은 특정 클러스터 개수까지 복원 결과가 좋음을 반영하다가 일정 수치 이후에 감소하는 것을 볼 수 있다. 그러나 MI 와 NMI 는 지속적으로 증가한다.

$$\mathrm{NMI}(A,B) = \frac{I(A;B)}{\sqrt{H(A)H(B)}} \cdots\cdots (4)$$

이것은 NMI 의 수식 (4)가 주어진 A 와 B 의 요소만을 관측하며 MoJoFM 과 같이 일반화된 문제를 해결하지 못하였기 때문에 발생한다. 그런 반면, AMI 는 그림 4 와 같이 MoJoFM 과 마찬가지로 일정 수치까지 상승 후 MoJoFM 과 다르게 급격히 감소하는 것을 볼 수 있다.

$$\mathrm{AMI}(A,B) = \frac{[MI(A,B) - E(MI(A,B))]}{[\max(H(A),H(B)) - E(MI(A,B))]} \cdots\cdots (5)[12]$$

이는 AMI 수식 (5)가 각 클러스터가 가지는 정보량의 기대값을 제거하고, 최대값으로 나누어 줌으로써 MoJoFM 과 마찬가지로 일반화 문제를 해결하였기 때문이다. 해당 결과는 Hadoop_core 0.19.0 에서도 마찬가지로 나왔으며 AMI 와 MoJoFM 만을 비교한 결과는 그림 4 와 같다.

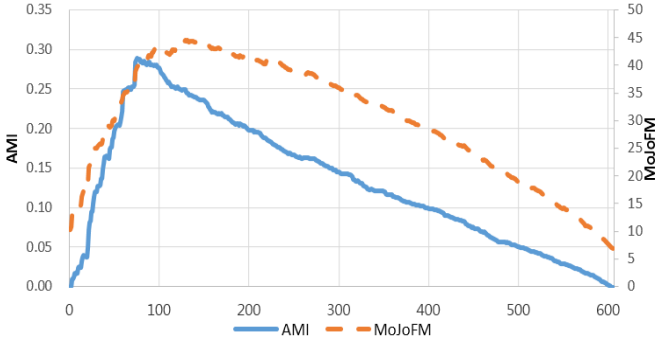


그림 4. Hadoop_core 0.19.0 의 n 개 클러스터링 결과와 정답지의 MoJoFM, AMI 비교

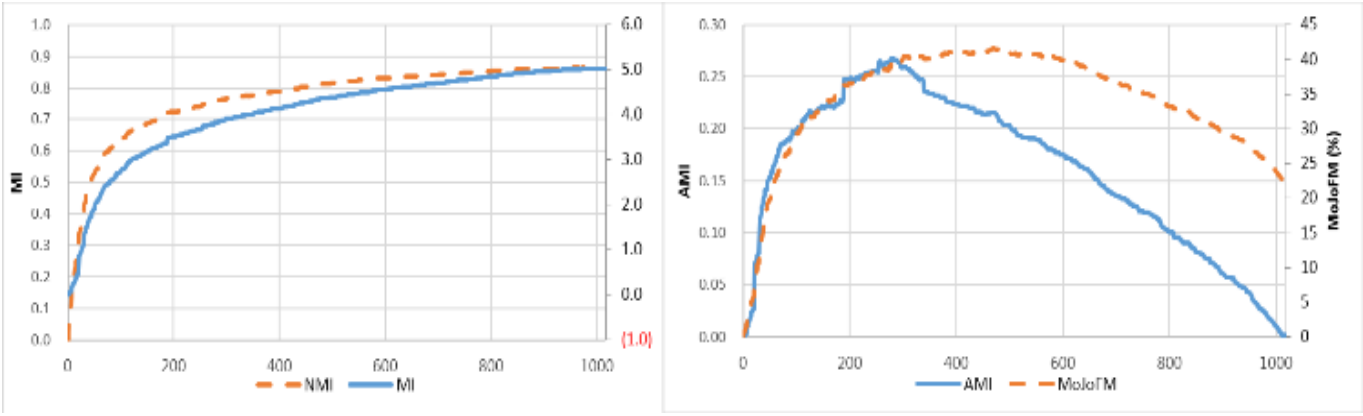


그림 5. Apache_oodt 0.2 의 n 개 클러스터링 결과와 정답지의 MoJoFM, MI 시리즈 비교

그림 4 와 그림 5 의 그래프 추이에서 AMI 는 감소폭이 MoJoFM 에 비해 가파른 것을 볼 수 있다. 또한 각 결과의 피어슨 상관계수는 Apache_oodt 0.2 가 0.67, Hadoop_core 0.19.0 은 0.93 이로 상관성이 높은 것을 알 수 있다.

표 2 MoJoFM 과 AMI 의 클러스터 개수에 따른 평가 비율

	정답지 클러스터 개수	Max(MoJoFM) 클러스터 개수	관측 비율	Max(AMI) 클러스터 개수	관측 비율
Apache oodt 0.2	216	468	30%	282	13%
Hadoop core 0.19.0	66	129	116%	75	95%

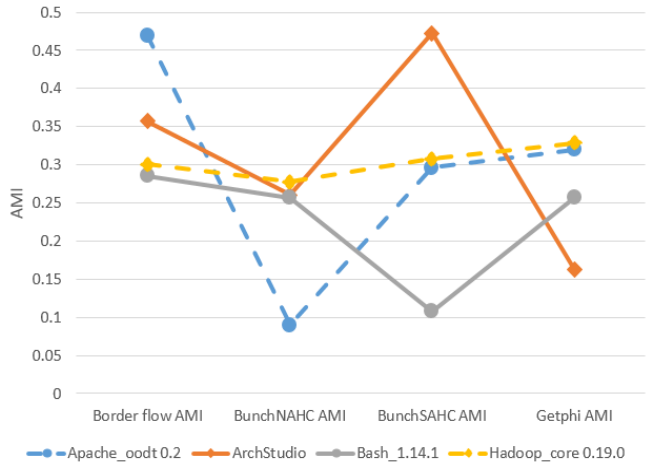


그림 6. 네 개 프로젝트의 AMI 복원 평가 결과

해당 추가 실험에서 한 가지 흥미로운 점이 표 2 와 같이 나타났다. Apache_oodt 의 정답지 클러스터가 216 개 일 때, MoJoFM 은 정답지 클러스터 개수로부터 116% 벗어난 지점인 468 개의 클러스터를 가질 때 가장 좋은 복원 결과임을 반영한 반면 AMI 는 30% 지점인 282 개를 반영하였다. hadoop_core 역시 마찬가지로 나타났으며 이 같은 현상은 AMI 는 급격하게 평가치를 떨어트리는 반면 MoJoFM 은 비교적 완만하게 감소하는 그림 5 로 설명할 수 있다. 즉, AMI 는 정답지의 클러스터 개수와 멀어질수록 MoJoFM 보다 엄격한 평가를 하는 것을 알 수 있다.

이러한 특징을 반영하여 3.1 의 데이터에 AMI 를 적용한 결과는 그림 6 과 같다. 그림 2 와 그림 3 과는 다르게 MoJoFM 과 비슷한 양상을 보이는 것을 알 수 있다. 또한 추가 실험의 결과로 볼 때, BunchSAHC 이용한 bash 프로젝트의 결과는 정답지와 클러스터 개수가 많은 차이가 있음을 짐작할 수 있다.

4. 결론 및 향후 연구

3 절의 실험을 통해 MoJoFM 과 함께 사용할 수 있는 소프트웨어 복원 평가 메트릭으로 AMI 를 사용할 수 있음을 보였다. 또한 AMI 는 MoJoFM 과는 다르게 정답지의 클러스터 개수와 근접한 개수를 가장 좋은 복원 결과 값으로 반영하는 것이 강점임을 알 수 있다. 이는 소프트웨어 복원 평가의 궁극적인 목표가 정답지와의 비교가 아닌 실세계의 소프트웨어 복원임을 생각해 볼 때, AMI 를 통한 평가치는 소프트웨어의 클러스터의 개수를 결정 하는 요소로 작용할 수 있음을 나타낸다.

하지만 본 실험에서 사용한 프로젝트의 셋이 적으며, 추가 실험 역시 두 개의 프로젝트로 특성을 알아보았으므로 일반화를 할 수 없다. 따라서 향후 연구로 다양한 실험셋을 통해 본 연구에서 보인 특성이 지속적으로 나타나는지 관찰 및 분석하고자 한다. 또한 AMI 가 본 실험과 마찬가지의 특성을 보인다면 MoJoFM 과 AMI 의 복원 평가가 어떤 특성에 강점을 보이는지 주성분 분석(PCA)을 통해 분석할 예정이다. 이를 통해 AMI 의 강점이 부각된다면 소프트웨어 복원 평가 메트릭의 한 부분으로 AMI 를 사용할 수 있을 것이다.

참고문헌

[1] C. Hofmeister, R.L. Nord and D. Soni, “Describing Software Architecture with UML”, in Proceedings of The International Federation for Information Processing on Software Architecture, Vol. 12, pp. 145-159, 1999.

[2] Mark Shtern and Vassilios Tzerpos, “Clustering Methodologies for Software Engineering”, Advances in Software Engineering, 2012.

[3] V. Tzerpos and R.C. Holt, “MoJo : A Distance Metric for Software Clusterings”, in Proceedings of 6th Working Conference on Reverse Engineering, pp. 187-193, 1999.

[4] Z. Wen and Vassilios Tzerpos, “An effectiveness measure for software clustering algorithms”, in Proceedings of 12th IEEE International Workshop on Program Comprehension, 2004.

[5] Brian S. Mitchell, “On the Automatic Modularization of Software Systems Using the Bunch Tool”, in Proceeding of IEEE Transaction on Software Engineering, Vol 32, Issue 3, pp. 193-208, 2006.

[6] V. Tzerpos and R. C. Holt, “ACDC : An Algorithm for Comprehension-Driven Clustering”, in Proceeding of 20th Working Conference on Reverse Engineering, pp.258-268, 2000.

[7] L.Šubelj and M.Bajec,"Community structure of complex software systems: Analysis", Physica A: vol.390, pp.2968-2975, May 2011.

[8] William M. Wells III, Paul Viola, Hideki Atsumi, Shin Nakajima and Ron Kikinis, “Multi-model volume registration by maximization of mutual information”, Journal of Medical Image Analysis, Vol. 1, issue 1, pp. 35-51, 1996.

[9] P. Andritsos and V. Tzerpos, “Information-Theoretic Software Clustering”, in Proceedings of IEEE Transactions on Software Engineering, Vol. 31, Issue 2, pp. 150-165, 2005.

[10] A. N. Ngomo and F. Schumacher, “BorderFlow: A Local Graph Clustering Algorithm for Natural Language Processing”, in Proceedings of 10th International Conference on Computational Linguistics and Intelligent Text Processing, Vol. 5449, pp. 547-558, 2009.

[11] M. Bastian and S. Heymann, “Gephi : An Open Source Software for Exploring and Manipulating Networks”, in Proceedings of Third International AAAI Conference on Weblogs and Social Media, 2009.

[12] N. X. Vinh, Julien Epps and James Bailey, “Information theoretic measures for clusterings comparison: is a correction for chance necessary?”, in Proceedings of the 26th Annual International Conference on Machine Learning, pp. 1073-1080, 2009.

클러스터 앙상블을 이용한 소프트웨어 아키텍처 모듈-뷰 복원

조충기, 이찬근

중앙대학교 컴퓨터공학부
서울 동작구 흑석동 84
cndrldhQk@gmail.com, cglee@cau.ac.kr

요약: 소프트웨어 모듈-뷰 복원은 아키텍처 문서가 누락된 거대 시스템의 유지보수 작업을 효율적으로 지원할 수 있는 기반을 제공한다. 기존에 제안된 복원 연구들은 데이터 분석에 사용되는 클러스터링 알고리즘을 소프트웨어 복원에 적합한 형태로 변형하여 사용해왔다. 그리고 복원 품질을 높이기 위한 알고리즘 고도화 방안 등을 제안해왔다. 그러나 대부분의 연구들은 하나의 단일 클러스터링 기법만을 사용하는 한계를 가지고 있다. 본 논문에서는 데이터 마이닝이나 패턴인식 분야에서 활발히 연구되고 있는 클러스터 앙상블 기법을 소프트웨어 모듈-뷰 복원에 적용하는 연구를 진행한다. 여러 개의 클러스터링 기법의 결과들을 하나의 통합적인 클러스터링 결과로 도출하는 이 기법은 단일 클러스터링 기법을 사용한 경우에 비해 더 높은 복원 품질과 견고한 성능을 보인다. 대표적인 클러스터 앙상블 기법인 EA(Evidence Acculmulation)을 소프트웨어 모듈-뷰 복원에 적용해 본 실험 결과 단일 기법보다 더 뛰어난 복원 품질과 견고성을 보이는 것을 볼 수 있었다.

핵심어: 클러스터링, 클러스터 앙상블, 모듈-뷰 복원

1. 서 론

소프트웨어는 시간이 지남에 따라 그 크기가 거대화되고 있고 그에 따라 복잡도 또한 크게 증가하고 있다. 현재 대부분의 상용 소프트웨어 시스템의 코드 라인 수는 백만 단위를 넘어 선지 오래 되었다. 이러한 거대 시스템의 유지보수 작업은 굉장히 어렵고 비용이 많이 들어가는 작업이다. 시스템 아키텍처 문서는 이런 어려운 작업을 도와 줄 수 있는 버팀목이 될 수 있다. 거대하고 복잡한 시스템을 의미있고 독립적인 서브시스템으로 바라볼 수 있는 추상화된 정보들을 제공하기 때문이다.

그러나 많은 레거시 시스템들의 경우 제대로 된 아키텍처 문서가 존재 하지 않거나 존재 하더라도 오랜 기간 유지보수를 거치면서 현재 시스템의 상태를 제대로 반영하지 못하고 있는 문서들만 존재하는 경우가 많다. 이런 경우 시스템의 아키텍처 복원 작업이 요구된다. 그러나 개발자들이 직접 손으로 이 작업을 수행하는 것은 너무나 큰 비용이 소모된다. 이런 문제를 해결하기 위해 자동적인 아키텍처 복원을 위한 기법들이 제안되어 왔다[1]. 이러한 자동 복원 연구들은 클러스터링 기법들을 사용해서 대상이 되는 소프트웨어 시스템을 의미 있고 독립적인 서브시스템으로 분할하는 접근법을 사용하고 있다.

다양한 클러스터링 기법들은 각자의 고유한 관점으로 데이터를 바라보고 클러스터링을 수행한다. 하지만 각 기법들의 관점은 모든 상황에 합치될 수는 없기 때문에 많은 경우에 있어 일관된 성능을 보이는 성질이 바람직하며, 이를 클러스터링 기법의 견고성으로 표현한다[2]. 이에 데이터 마이닝, 패턴 인식 분야에서는 클러스터링의 성능을 높이고 여러 상황에서도 견고한 성능을 달성하기 위해 클러스터 앙상블 기법에 대한 연구를 진행하고 있다. 클러스터 앙상블은 다양한 클러스터링 기법들의 결과들을 모아서 하나의 합치된 결론을 도출하는 방법이다. 해당 기법을 사용하면 단일 기법을 사용한 경우에 비해 높은 복원률과 견고한 성능을 얻을 수 있다고 알려져 있다[3].

그러나 소프트웨어 아키텍처 복원 분야에서는 클러스터 앙상블을 적용한 연구는 거의 진행되고 있지 않다. 대부분의 연구는 단일 클러스터링 기법만을 적용하는 한계를 가지고 있다. 따라서 본 논문에서는 소프트웨어 아키텍처 복원에 클러스터 앙상블 기법을 적용하는 연구를 수행하여 해당 기법이 소프트웨어 모듈-뷰 복원에도 효과적으로 적용 될 수 있는지 알아본다. 또한 클러스터 앙상블 기법을 적용하기 위해서는 반드시 여러 개의 다양한 클러스터링 결과를 양산하는 과정이 요구된다. 하지만 이 작업은 쉬운 작업이 아니다. 본 논문에서는 소프트웨어 모듈-뷰 복원 연구에 널리 사용되는 Bunch[11] 도구를 사용하여 상대적으로 손 쉽게 다수의 서로 다른 클러스터링 결과들을 생산하였다.

다음 2 장에서는 연구의 배경이 되는 소프트웨어 아키텍처 모듈-뷰에 관련된 사항들에 대해 알아본다. 3 장에서는 클러스터 앙상블 기법에 대해 살펴본 다음 4 장에서 해당 기법을 적용한 모듈-뷰 복원 실험

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성 지원사업(IITP-2015-H8501-15-1012)과 한국연구재단 기초연구사업(과제번호 NRF-2014-005519)의 지원을 받았습니다

과 그 결과에 대해서 논한다. 그리고 5 장 결론으로 이 글을 마무리 한다.

2. 배 경

2.1 소프트웨어 아키텍처 모듈-뷰 복원

소프트웨어 아키텍처 복원 연구에서는 주로 모듈-뷰의 형태로 아키텍처를 복원한다. 다양한 아키텍처 뷰 가운데서 유지보수 작업을 수행하는 개발자에게 가장 중요한 뷰는 모듈-뷰 이기 때문이다. 모듈-뷰는 소스코드 관점에서 바라본 소프트웨어의 아키텍처를 보여준다. 본 논문에서도 모듈-뷰 복원을 통해 소프트웨어 아키텍처 복원을 수행한다.

소프트웨어 모듈-뷰 복원은 소스코드에 존재하는 파일이나 클래스와 같은 개체들을 미리 정의된 특정한 기준에 따라 군집을 구성하는 방식으로 진행된다. 이때 생성된 군집들은 각각 독립적인 서브시스템을 의미하게 되며 그 내부에 존재하는 원소들은 해당 서브시스템이 담당하는 기능들을 수행하기 위해 유기적으로 동작하는 소스코드 개체들을 나타낸다. 전형적인 모듈-뷰 복원 절차는 아래와 같다[4].

(1) 소스코드 개체 및 특성 선택
(2) 추출된 특성을 통한 개체들 사이의 유사도 계산
(3) 클러스터링 알고리즘 적용
(4) 생성된 클러스터링 결과에 대한 평가

가장 먼저 클러스터링 대상이 되는 개체가 어떤 것인지 결정해야 한다. 객체지향시스템의 경우에는 주로 클래스를 개체로 선택하며 파일 또한 빈번히 사용되는 개체 단위이다. 개체를 결정한 다음에는 개체들 사이의 유사도를 계산하기 위한 특성을 추출한다. 다양한 특성들이 존재하지만 클래스나 파일 사이의 의존관계가 가장 널리 사용되는 특성이다. 한 클래스가 특정 클래스의 객체나 함수를 자주 사용한다면 두 클래스 사이의 의존관계가 깊다고 볼 수 있고 이 의존관계의 깊이를 통해 두 클래스 사이의 유사도를 측정 할 수 있다. 깊은 의존관계를 가지고 있는 두 클래스는 특정 임무를 수행하기 위해 유기적으로 협력하는 것을 의미하며 이는 두 개체간의 유사도가 높다고 판단 할 수 있다. 이렇게 유사도를 계산하여 특정 클러스터링 알고리즘을 수행하면 복원된 모듈-뷰를 얻을 수 있다.

해당 복원 결과가 얼마나 유용한지 판단하기 위한 평가 방법은 크게 2 가지로 나누어 진다. 만약 평가 기준으로 사용 할 수 있는 전문가들이 직접 만든 대상 소프트웨어의 모듈-뷰가 존재한다면 그것과 자동으로 복원된 결과의 비교를 통해 해당 복원 기법이 얼마나 좋은 결과를 생산해 내는지 판단 할 수 있다. 만약 위와 같은 평가 기준이 존재하지 않는다면 자체적인 평가 기준을 통해 기법을 검증한다. 자체적인 기준으로 자주 사용되는 것은 생성된 클러스터의 개수이다. 극단적으로 많거나 적은 클러스터를 형성하는 기법은 상대적으로 그렇지 않은 기법보다 복원능력이 떨어진다고 판단 할 수 있다.

3. 클러스터 앙상블

다양한 데이터 소스나 학습모델을 하나로 통합하는 접근방법은 이미 여러가지 분야에서 사용되고 있는 방법이다[3]. 데이터 마이닝 분야에서 자주 사용되는 기법인 랜덤 포레스트는 여러 개의 결정 트리를 사용해서 하나의 결과를 생성하는 기법으로 이러한 접근방법을 적용한 대표적인 사례이다[5]. 클러스터링의 경우에도 여러 개의 클러스터링 기법들의 결과들을 하나로 통합하여 복원 품질과 견고성을 높이는 시도가 계속되고 있다. 이러한 기법들을 클러스터 앙상블이라고 한다. 지금까지 다양한 클러스터 앙상블 기법들이 제안되어 왔다. Strehl 등의 연구에서는 하이퍼 그래프 모델링을 기반으로 3 가지의 클러스터 앙상블 기법을 제안했다[6]. Fern 등의 연구에서는 이분 그래프를 통한 모델링을 기반으로 클러스터 앙상블을 시도하였고 기존 그래프 모델링 방법들보다 높은 성능을 보여주었다[7]. Fred 와 Jain 은 대상이 되는 데이터 집합의 개체들 중 자주 같은 클러스터에 군집화되는 개체 쌍들에 대한 데이터를 기반으로 하는 EA(Evidence Accumulation) 클러스터 앙상블 기법을 제안하였다[8]. Topchy 등의 연구에서는 통계적 방법인 혼합모델을 사용하는 클러스터 앙상블 기법을 제안하였다[9]. 본 논문에서는 가장 직관적이고 간단하면서도 그 성능이 뛰어나다고 알려진 EA 기법을 사용해서 클러스터 앙상블을 수행한다[10].

3.1 Evidence Accumulation

Fred 등이 제안한 EA 기법은 다음과 같은 절차로 수행된다.

(1) 여러 개의 클러스터링 결과 생성
(2) 여러 개의 클러스터링 결과에서 같은 클러스터에 포함되는 개체 쌍에 대한 패턴을 기반으로 연관 행렬(Co-association Matrix) 생성
(3) 생성된 연관 행렬을 통해 계층적 클러스터링 (Agglomerative Hierarchical Clustering) 수행
(4) 적절한 클러스터의 개수를 결정하여 최종 결과 도출

가장 먼저 여러 개의 클러스터링 결과를 생성하는 과정이 필요하다. 다양한 기법을 사용하여 생성 할 수 도 있고 단일 기법의 사용자 입력 값을 다양하게 조정하여 여러 개의 결과를 도출 할 수도 있다. 해당 논문에서는 Bunch[11] 도구를 사용하여 다양한 클러스터링 결과를 추출하는 방법을 채택하였다.

클러스터링 결과들의 집합이 완성되면 각 개체 쌍들이 얼마나 자주 같은 클러스터에 포함되는지에

대한 패턴을 파악하여 연관 행렬을 생성한다. 개체수가 n 개라고 가정하면 n x n 연관 행렬이 생성되는데 이 행렬의 원소 co_assoc(i,j)는 개체 n_i 와 n_j 가 전체 N 개의 클러스터링 결과에서 같은 클러스터에 포함된 횟수를 의미하게 된다. 즉, 다양한 클러스터링 기법들의 결과에서 자주 그리고 반복적으로 같은 클러스터에 포함되는 개체들을 중심으로 클러스터를 구성하는 것이 이 기법을 관통하는 기저 개념이 된다. 생성된 클러스터링 결과의 개수를 N 개라고 하면 연관 행렬은 아래와 같이 나타낼 수 있다. $votes_{ij}$ 는 개체 쌍 (i,j)가 전체 N 개의 클러스터링 결과에서 같은 클러스터에 할당된 횟수를 의미한다.

$$\text{co\_assoc}(i,j) = \frac{votes_{ij}}{N}$$

위에서 생성된 연관 행렬은 데이터 집합을 구성하는 개체들 사이의 유사도로 간주 할 수 있으며 따라서 이 행렬을 기반으로 클러스터링 기법을 적용 할 수 있다. 해당 연구의 저자들은 널리 사용되는 기법 중 하나인 계층적 클러스터링 기법(Agglomerative Hierarchical Clustering Algorithm)[12]을 사용하였다. 이 기법은 클러스터링 결과의 클러스터 개수를 사용자에게 입력으로 받아서 클러스터링을 수행한다. 이 논문에서 수행한 실험에서는 가능한 모든 클러스터 개수에 대해서 EA 기법을 적용한 후에 가장 높은 성능을 보이는 결과를 채택하는 방법을 사용했다.

4. 실 험

4.1 대상 프로젝트

해당 실험은 총 4 개의 시스템을 사용해서 수행되었다. ArchStudio[13]는 이클립스의 플러그-인 형태로 구현된 개발 지원 도구이며 소프트웨어 아키텍처와 시스템에 대한 모델링, 시각화, 분석 및 구현 기능 등을 제공한다. java 로 구현되었고 약 28 만 라인의 소스코드와 800 여개의 클래스가 존재한다. Bash[14]는 리눅스와 맥 OS X 등의 운영체제에서 기본 셸로 사용되는 커맨드 라인 기반 인터페이스이다. C 언어로 제작 되었으며 약 7 만라인의 소소코드와 200 여개의 파일로 구성된다. apache OODT[15]는 데이터 집약적인 분산 시스템을 지원하는 미들웨어 프레임워크이다. 주로 자바언어로 구현되었으며 약 18 만 라인의 코드로 구성된다. Apache Hadoop[16]은 분산처리를 위한 오픈 소스 프레임워크이다. Java 로 구현되었으며 약 20 만 라인의 코드 그리고 1700 여개의 클래스로 구성되어 있다. 프로젝트 별 세부 정보는 표 1 에서 볼 수 있다.

표 1 - 실험 대상 프로젝트

프로젝트 이름	버전	도메인	구현언어	LOC
ArchStudio	4	아키텍처 개발 환경	java	280K
Bash	1.14.1	운영체제 셸	C	70K
OODT	0.2	데이터 미들웨어	Java	180K
Hadoop	0.19.0	분산 처리 프레임워크	Java	200K

4.2 다수의 클러스터링 결과 생성

클러스터 앙상블을 수행하기 위해서는 여러 개의 클러스터링 결과를 생성하는 과정이 필요하다. 이 실험에서는 여러 개의 클러스터링 결과를 양산하기 위해 Bunch[11]를 사용하였다. Bunch 는 소프트웨어 클러스터링에 널리 사용되는 도구 중 하나로 목적함수 기반의 클러스터링 기능을 제공한다. MQ 라는 목적함수를 기반으로 탐색 알고리즘을 적용해서 해당 목적함수를 최대화 시키는 클러스터링 결과를 찾는다. 목적함수 MQ 는 소스코드 개체들이 높은 응집도(cohesion)와 낮은 결합도(coupling)를 가질수록 높은 값을 가지도록 설계된다. 목적함수를 최대화 하는 결과를 찾기 위한 방법으로는 전역탐색(Exhaustive Search)과 언덕 오르기(Hill-Climbing) 방법 등이 제공된다.

해당 실험에서는 효과적인 클러스터링 결과 양산을 위해 언덕 오르기 방법을 사용해서 200 개의 서로 다른 클러스터링 결과를 생성하였다. 언덕 오르기 방법의 특성상 클러스터링을 수행 할 때 마다 매번 다른 결과가 생성된다. 또한 bunch 는 클러스터링 수행을 위한 여러 가지 초기 설정 값을 입력 할 수 있다. 클러스터링 결과의 다양성을 더욱 높이기 위해서 초기 설정값을 매번 다르게 입력하여 클러스터링을 수행하였다.

4.3 클러스터링 평가 방법

실험에 사용된 프로젝트들은 전문가들이 작성한 모듈-뷰가 존재한다[17]. 따라서 이 실험에서는 EA 기법으로 복원된 결과와 제공된 정답 사이의 유사도를 측정하여 클러스터 앙상블 기법이 얼마나 효과적인지 평가하였다. 해당 실험에서 사용한 클러스터링 평가 방법은 아래 4 가지 방법들을 사용하였다. 아래의 방법들은 소프트웨어를 포함한 여러 분야에서 서로 다른 두 개의 클러스터링 결과를 비교하는데 널리 사용되고 있는 방법들이다.

표 2 - 실험 결과

	MoJoFM	ARI	F-Measure	AMI
(ArchStudio4)				
EA	0.642	0.434	0.588	0.585
Bunch	0.413(0.562)	0.152(0.378)	0.368(0.509)	0.284(0.474)
(Bash)				
EA	0.509	0.391	0.536	0.420
Bunch	0.343(0.509)	0.182(0.470)	0.379(0.530)	0.222(0.474)
(apache OODT)				
EA	0.492	0.304	0.486	0.469
Bunch	0.162(0.290)	0.085(0.178)	0.176(0.313)	0.246(0.355)
(apache Hadoop)				
EA	0.549	0.309	0.456	0.452
Bunch	0.317(0.453)	0.136(0.249)	0.303(0.413)	0.261(0.389)

(1) MoJoFM[18]

소프트웨어 클러스터링에서 널리 사용되는 방법으로 서로 다른 두 개의 클러스터링 결과가 얼마나 유사한지 측정 할 수 있는 기능을 제공한다. 한 클러스터링의 결과가 또 다른 클러스터링 결과와 똑같아지기 위해 수행해야 하는 이동(Move)과 합병(join) 연산의 횟수를 통해 두 결과 사이의 유사성을 구한다.

(2) ARI(Adjusted Rand Index)[19]

RI(Rand Index)[19]는 두 클러스터링 사이의 유사도를 측정하는 측정 방법으로 데이터 마이닝을 비롯한 다양한 분야에서 사용된다. 데이터 집합을 구성하는 개체들의 전체 쌍에 대해 두 클러스터링 결과에서 같은 클러스터에 포함되는 쌍 또는 두 클러스터링 결과에서 서로 다른 클러스터에 포함되는 쌍의 비율을 계산하여 유사도를 도출한다. ARI 는 정규화 그리고 랜덤 클러스터링 결과가 높은 값을 가지지 않도록 하는 RI 의 발전된 형태이다.

(3) F-Measure

패턴인식과 정보검색 분야에서 주로 사용되는 방법으로 정밀도(Precision)와 재현율(Recall)의 조합으로 구성된다. 본 실험에서는 F1 측정방법을 사용해서 클러스터링 결과를 비교하였다.

(4) AMI(Adjusted Mutual Information)[20]

새넌의 정보이론을 바탕으로 두 클러스터링 사이의 유사도를 측정한다. 각 클러스터링 결과들을 하나의 확률변수로 모델링 한 다음 두 확률변수의 상호 정보량(Mutual Information)을 통해 유사도를 측정한다.

4.4 실험 결과 및 분석

실험 결과를 통해 EA 기법을 사용하여 클러스터 앙상블을 적용했을 때 단일 기법을 통한 결과보다 얼마나 정답에 가까운 결과를 도출하는지 그리고 얼마나 더 견고한 성능을 보이는지 살펴본다.

표 2 는 각 프로젝트 대해서 EA 기법의 결과와 단일 기법인 Bunch 의 결과를 비교한 것이다. Bunch 의 결과들은 200 개의 클러스터링 결과에 대한 평균값을 나타내고 괄호 안에 있는 값은 최대값을 의미한다. ArchStudio 의 경우 MoJoFM 값이 bunch 의 최대값보다 약 8% 정도 상승하였고 다른 평가 기준들의 값 역시 향상된 것을 볼 수 있다. Hadoop 의 경우에도 ArchStudio 와 비슷한 양상의 성능의 향상을 볼 수 있었다.

OODT 는 다른 프로젝트에 비해서 Bunch 기법의 결과가 좋지 않은 편이다. 모든 데이터에 대해서 항상 좋은 결과를 기대 할 수 있는 클러스터링 기법이 존재 하지 않는 것처럼 해당 프로젝트에서 bunch 기법은 큰 효과를 보지 못하였다. 그러나 EA 기법을 적용한 결과는 상당히 좋은 성능 향상을 나타내고 있다. OODT 처럼 특정 클러스터링 기법이 약점을 보이는 데이터 셋이 존재하고 EA 기법은 이러한 약점들을 상쇄시키는 역할을 할 수 있다. 즉, EA 기법을 사용하면 단일 기법이 약점을 보이는 데이터 집합에 대해서도 높은 수준의 성능 향상을 보여줄 수 있기 때문에 다양한 데이터에 대해서 견고한 성능을 보장한다고 볼 수 있다.

Bash 프로젝트에서는 EA 기법이 큰 효력을 발휘하지 못한 것으로 보여진다. 4 가지의 평가기준에서 EA 기법의 결과와 Bunch 의 최대값과 비슷하거나 오히려 조금 낮아진 경우도 발생하였다. 이는 아마

Bash 프로젝트의 특성 때문이라고 보여진다. 다른 프로젝트에 비해 Bash 의 규모는 상당히 작은 편에 속한다. 파일 단위로 개체를 선정 했을 때 채 200 개가 넘지 않는다. 따라서 200 개의 Bunch 결과 중에 가장 좋은 클러스터링과 EA 기법의 결과와 큰 차이가 나지 않는 것으로 파악된다. 그러나 200 개 결과의 평균값과 비교하면 EA 기법은 여전히 좋은 결과를 보여주고 있다.

5. 결론 및 향후 연구

본 논문에서는 소프트웨어 모듈-뷰 복원을 위해 클러스터 앙상블 기법을 적용하는 연구를 진행하였다. 기존의 복원 연구들은 단일 클러스터링 기법에만 의존하는 한계를 보이고 있기 때문이다. 대표적인 클러스터 앙상블 기법인 EA 을 적용한 결과 단일 기법을 적용한 경우에 비해 높은 복원율 그리고 견고한 성능을 보이는 것을 확인 할 수 있었다. 향후 연구로는 EA 기법과 Bunch 이외의 아키텍처 복원 분야에서 많이 사용되는 기법들과의 성능을 비교하는 실험을 진행할 예정이다. 이를 통해 EA 기법의 실질적인 경쟁력을 살펴 볼 수 있을 것이다. 또한 다양한 클러스터 앙상블 기법들은 적용해 보는 것 또한 향후에 수행해 볼만한 연구이다.

참고문헌

[1] M Shtern, V Tzerpos, “Clustering methodologies for software engineering”, Journal Advances in Software Engineering, Volume 2012, Atricle No.1, January 2012.

[2] G Forestier, P Gançarski, C Wemmert, “Collaborative clustering with background knowledge”, Data & Knowledge Engineering, Volume 69, Issue 2, Pages 211–228, February 2010.

[3] J Ghosh, A Acharya, “Cluster ensembles”, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, Volume 1, Issue 4, Pages 305–315, July/August 2011.

[4] S Muhammad, O Maqbool, A Abbasi, “Evaluating relationship categories for clustering object-oriented software systems”, IET Software, Volume 6, Issue 3, Pages 260–274, June 2012.

[5] P Robillard, W Maalej, J Walker, T Zimmermann, “Chapter 3 Data Mining in Recommendation Systems in Software Engineering”, Springer-Verlag Berlin Heidelberg 2014.

[6] A Strehl, J Ghosh, “Cluster ensembles --- a knowledge reuse framework for combining multiple partitions”, Journal The Journal of Machine Learning Research, Volume 3, Pages 583-617, May 2003.

[7] Z Fern, E. Brodley, “Solving Cluster Ensemble Problems by Bipartite Graph Partitioning”, Proceedings of the international conference on Machine learning, 2004.

[8] L Fred, K Jain, “Combining Multiple Clusterings Using Evidence Accumulation”, IEEE transactions on pattern analysis and machine intelligence, Vol. 27, No. 6, June 2005.

[9] A Topchy, K Jain, W Punch, “A Mixture Model for Clustering Ensembles”, Proceedings of the SIAM International Conference on Data Mining, 2004.

[10] N Nguyen, R Caruana, “Consensus Clusterings”, the IEEE International Conference on Data Mining, 2007.

[11] S Mitchell, S Mancoridis, “On the automatic modularization of software systems using the Bunch tool”, IEEE Transactions on Software Engineering, Vol.32, No.3, Pages 193-208, 2006.

[12] J Cui, C Heung Seok, “Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems”, Information and Software Technology, Volume 53, Issue 6, Pages 601–614, June 2011.

[13] ArchStudio [online]. available: http://isr.uci.edu/projects/archstudio/index.html

[14] Bash Wiki [online]. available: https://en.wikipedia.org/wiki/Bash_(Unix_shell)

[15] Atlassian Confluence Open Source Project, Apache OODT [online]. available: https://cwiki.apache.org/confluence/display/OODT/Home

[16] Hadoop Wiki [online]. Available: https://en.wikipedia.org/wiki/Apache_Hadoop

[17] J Garcia, I Krka, C Mattmann, N Medvidovic, “Obtaining Ground-Truth Software Architectures”, Proceedings of the International Conference on Software Engineering, 2013.

[18] Zhihua Wen and Vassilios Tzerpos, “An effectiveness measure for software clustering algorithms”, the IEEE International Workshop on Program Comprehension, 2004.

[19] L Hubert, P Arabie, “Comparing partitions”, Journal of Classification, Pages 193-218, 1985.

[20] N Vinh, J Bailey, “Information Theoretic Measures for Clusterings Comparison: Is a Correction for Chance Necessary?”, Proceedings of the International Conference on Machine Learning, 2009.

온라인 게임 위치 기반 그룹 분류를 통한 봇 탐지 기법에 대한 연구

이세희, 김수아, 이지형

성균관대학교 전자전기컴퓨터공학과
경기도 수원시 장안구 서부로 2066
e-mail : bluffrin@gmail.com, { sa4956, john}@skku.edu

요약: 온라인 게임 시장이 커짐에 따라 불법적으로 이득을 얻는 봇들도 발전하고 있다. 온라인 게임 내의 봇들은 '작업장'이라는 이름으로 조직적으로 활동하고 있으며 온라인 게임 유저들과 게임 회사에 막대한 피해를 끼치고 있다. 기존에도 캐릭터의 행위 시퀀스를 보거나 플레이 스타일을 보고 봇을 탐지하는 연구가 있었지만, 봇들의 행동이 갈수록 지능화되어 사람의 패턴과 비슷해져서 탐지가 어려워지고 행위 기반 로그를 실시간으로 필요로 한다는 점에서 필요 이상의 많은 로그를 수집해야 하기 때문에 서버에 무리를 주는 등 한계점을 보이고 있다. 본 연구에서는 게임 봇 탐지를 위해 개개인의 특성이나 패턴이 아닌 위치정보를 사용하여 그룹을 생성한 뒤, 그 그룹 내의 상태 차이로 만든 특징을 사용하는 방식을 제안한다. 실험을 통해 제안 기법이 봇을 합리적인 수준으로 탐지할 수 있음을 보였다.

핵심어: 게임 봇 탐지, 그룹 모델, 인공지능, 온라인 게임

1. 연구 배경

온라인게임에서 봇이 성행함에 따라 유저와 회사 측에서 큰 피해를 입고 있다. 게임 봇이란 사람이 아닌 인공지능이 플레이어 대신 자동 명령을 수행하며 자동 사냥 혹은 자동 채집으로 게임 내 재화를 쉽게 얻을 수 있고 캐릭터의 성장도 쉽고 빠르게 할 수 있는 프로그램이다. 게임 봇은 다른 유저의 플레이를 방해하고 불법적으로 게임 아이템을 취득하면서 게임 내 시장경제 균형을 무너트린다. 상대적으로 적은 노력과 시간을 투자하여 게임 내 재화를 취득하기 때문에 유저들에게 상대적인 박탈감을 느끼게 하며 게임 운영을 어렵게 한다. 이러한 게임 봇이 생기는 이유는 게임 내 재화를 현금으로 바꿀 수 있기 때문이다. 이를 위해 불법 사용자들은 조직적, 전문적으로 시설을 차려 '작업장'을 운영하여 대규모로 부당한 수익을 얻고 있다.

이러한 문제로 인하여 효과적인 게임 봇 검출 방법에 대한 연구가 지속적으로 이루어지고 있다. 하지만 이런 감시망을 피하기 위해 작업장들이 사용하는 봇 프로그램은 갈수록 지능화되고 있다. 기존의 개인 사냥패턴이나 개인 활동만으로는 기존 사용자와의 차이가 거의 없어져 분류에 무리가 있다고 판단되어 플레이어들을 위치기반으로 그룹을 지어 그 그룹간의 특성으로 감지를 하는 새로운 기법에 대해 연구하였다.

온라인게임의 큰 특징 중에 하나는 협동으로, 게임을 진행하기 위해선 여럿이서 움직이는 것이 가장 효과적이다. 대부분의 플레이어들은 자신의 레벨에 맞는 구간에서 다른 플레이어들과 비슷한 행동 양상을 보인다. 또, 다른 플레이어와의 협동 및 친목활동으로 함께 행동하는 특징을 보인다. 이에 따른 사람 그룹과 봇 그룹은 매우 다른 양상을 보인다. 플레이어들은 플레이어들과 어울리고 봇들과 어울리지 않는다. 봇들은 봇들과 파티를 지어 지능적으로 사냥 및 작업을 한다.

온라인 게임 사업자는 기존 기법을 사용하여 게임 봇들을 탐지하고 불량 사용자들의 계정을 제제하지만 작업장들은 여러 캐릭터들을 바꿔가면서 수많은 계정을 키워내기 때문에 캐릭터를 하나하나 일일이 제거하는 것은 큰 효과를 보지 못한다. 대신 그룹단위 추출을 하게 되면 더욱 효과적인 제제를 가할 수 있게 될 것이다.

봇 길드, 봇 파티 등 그룹으로 하는 탐지에 대한 연구가 다소 진행되어 왔다[1,2]. 하지만 파티단위 탐지와 같은 경우에는 활강 특성을 사용하는 등 특정 온라인 게임에서만 적용되는 특성을 다루었다. 본 논문에서는 이처럼 특정 게임에 국한된 것이 아닌 보다 범용적으로 적용할 수 있는 방법을 제안한다.

본 논문의 구성은 다음과 같다. 2 장에서는 봇 탐지 기법 연구들을 소개하고 3 장에서는 제안한 방법을 단계별로 소개할 것이다. 4 장에서는 국내 유명 온라인 게임 열혈강호의 실제 데이터 샘플에 적용하여 제안한 방법의 효용성을 분석하였다. 5 장에서는 결론을 도출하고 향후 연구에 대해 논의한다.

2. 관련 연구

데이터마이닝 기법, 통계적 기법, 유사도 패턴 매칭 기법 등 봇을 검출하는 여러 방식이 있다[3]. Jina Lee 등은 서버 단에서 캐릭터의 행위 시퀀스를 특징으로 추출하고, 나이브 베이즈 분류 알고리즘에 적용하여 MMORPG 에서 게임 봇을 분류하였다[4]. Ahmad 등은 아이템의 거래 네트워크를 통해 봇을 탐지하였다[5]. Yeounoh Chung 등은 유저의 플레이 스타일을 통해 유저를 분류하고 그 뒤에 특징을 추출하여 봇을 탐지하였다[6]. Jin Lee 등은 캐릭터의 성장 유형으로 분류를 하여 하드코어 유저와 봇 유저를 구분하였다[7]. M. van Kesteren. 등은 캐릭터의 이동정보와 행위 기반 로그 사이의 시간간격을 통해 봇을 탐지하였다 [8].

봇들의 행동이 갈수록 지능화되어 유저의 사냥 패턴과 봇들의 사냥 패턴이 거의 유사하다. 심지어 자동사냥 시스템까지 생겨 봇을 사냥 패턴만으로 구분하는 것에는 한계가 있다.

대부분의 사냥 및 활동이 몇 초 안 되는 사이에 한 캐릭터로부터 수십 번의 입력을 받고 이루어지기 때문에 사냥 및 활동 특성이나 입력 로그를 사용하는 기법은 서버에 무리를 준다. 또, 약간의 시간 간격이 있는 로그가 주어졌을 때 봇 유저를 탐지하기 어려운 단점이 있다.

본 연구는 사냥 패턴을 사용하지 않고 위치, 상태 로그를 사용하여 봇을 탐지하여 위의 문제를 해결하고자 한다. 또한, 본 논문의 제안기법은 상태 로그를 사용하기 때문에 상대적으로 로그를 적게 사용하여 서버에 무리를 주지 않는 장점이 있다.

3. 제안 방법

3.1 봇 그룹 탐지를 위한 분석 모델 제안

봇 그룹 탐지를 위한 분석 모델은 다음 그림 1 과 같다.

먼저 데이터수집을 하였다. 게임 로그로부터 플레이어들의 x,y 좌표를 이용한 위치, 현재 레벨, 파티 여부, 접속 시간에 대한 로그를 한 사람당 5 분단위로 수집하였다. 그 후 가우시안 혼합 모델을 기반으로 5 분에 한번씩 플레이어들의 좌표정보를 이용하여 군집화를 하였고, 이를 기반으로 그룹을 생성하였다.

마지막으로 봇 그룹과 유저 그룹을 구분할 수 있는 특징을 추출하였다. 특징들은 개인 활동보다는 그룹 내의 상태 차이를 중심으로 하였다. 추출된 특징을 기반으로 로지스틱 회귀분석(Logistic Regression), 서포트 벡터 머신(Support Vector Machine, SVM) 분류 알고리즘을 사용하여 봇 그룹을 탐지하였다.

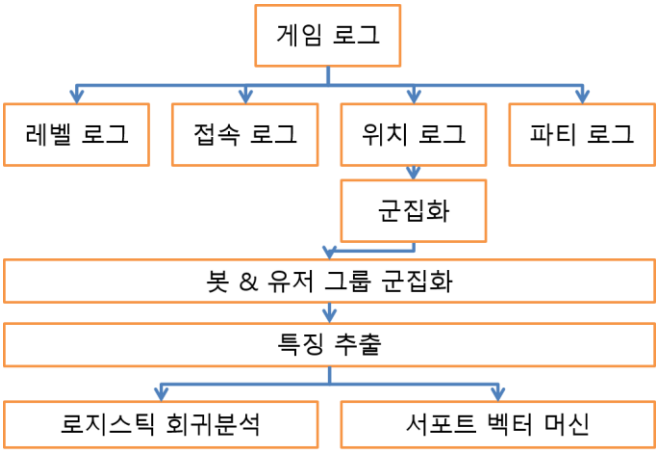


그림 1 봇 그룹 탐지를 위한 분석 모델

3.2 봇 유저 그룹 분류

같은 작업장 봇들은 사냥 및 활동을 위해 서로 그룹을 지어 다니고, 사람들은 보다 효율적이고 사회적인 플레이를 위해 사람들과 그룹을 지어서 다닌다. 봇 그룹과 유저 그룹을 분류하고 그 분류가 어느 정도 정확한지 알기 위해 위치정보를 이용하였다.

일반 유저와 봇 유저의 그룹을 만드는 과정은 다음 그림 2 와 같다.

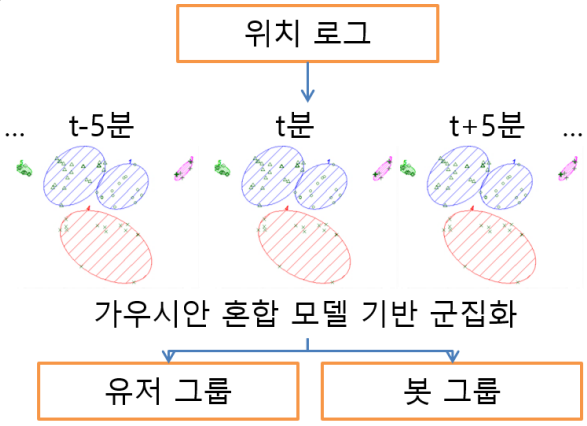


그림 2 유저/봇 그룹 생성

로그 시간 5 분마다 한번씩 좌표로그를 이용하여 가우시안 혼합 모델 기반 군집화를 하였다. 이 군집화 정보를 가지고 한 번의 군집화에서 같은 곳에 분류된 횟수를 플레이어들간의 얼마나 연관이 있는지에 대한 score 로 정의한다. 이 때 한 플레이어를 기준으로 score 가 가장 높았던 플레이어를 그 플레이어의 그룹으로 간주한다. 유저 사이의 관련 점수 $P_{i,j}$ score는 다음 식 1 과 같다.

$$P_{i,j}\ \text{score} = \sum_{t=0}^{T} (G_{t,P_i} = G_{t,P_j}) \qquad [1]$$

$$P_i\ Pair = Max(P_{i,j}score)$$

G_{t,P_i} : 시간 t 에서 i 번째 player 의 그룹 정보

3.3 특징 선택

유저 그룹과 봇 그룹을 모델링 하기 위한 사람과 게임 봇의 패턴 차이 특성을 제안한다. 여기서는 그룹내의 관계를 중요한 특징으로 간주하였다. 유저 그룹과 봇 그룹을 구분할 수 있는 특징은 다음 표 1 과 같다.

표 1. 그룹 특성

검출 속성	행동 패턴	
	유저 그룹	봇 그룹
동시 접속률	접속시간이 비슷한 사람끼리 짝을 이룸	같이 다니는 봇과 접속시간이 많이 겹치지 않음
서로의 거리	서로의 거리차가 다양함	서로의 거리차가 일정함
레벨 차이	비슷한 레벨과 짝을 이룸	레벨차가 큼
파티 정보	개인시간이 존재함	개인시간이 존재하지 않음

작업장들은 감시를 피하기 위해, 계정 수를 늘리기 위해 여러 캐릭터를 돌아가면서 키운다. 레벨이 높은 한 캐릭터를 중심으로 다른 캐릭터들과 파티를 맺어 사냥을 한다. 또한, 주로 레벨이 높은 캐릭터가 사냥을 하며 레벨이 낮은 캐릭터는 파티 경험치로 쉽게 레벨을 올린다는 특징이 있다.

이때 사람 그룹의 동시접속률과 봇 그룹의 동시접속률은 차이를 보인다. 사람 그룹의 동시접속률은 매우 높은 반면 봇 그룹은 한 캐릭터가 다른 여러 봇들과 번갈아가면서 함께 다니기 때문에 동시접속률이 낮을 것이다. 동시접속률을 측정하기 위한 식은 다음 식 2 와 같다.

$$\text{동시접속률} = \frac{s}{(f1.t + f2.t - s)} \quad [2]$$

s : 동시접속시간
f1.t : 기준 유저 접속 시간
f2.t : 짝 유저 접속 시간

봇 그룹은 서로의 거리가 일정하고 개인시간이 존재하지 않아, 서로 모든 활동을 같이하며 일정한 간격을 유지하면서 활동 할 것이다. 그에 비해 사람은 서로간의 거리가 일정하지 않을 것이며 개인시간을 보냄으로써 더욱 다양한 거리를 보일 것이다. 그룹 유저간의 거리를 측정하기 위한 식은 다음 식 3 과 같다.

$$\text{서로의거리} = E(\sqrt{(f1.x - f2.x)^2 + (f1.y - f2.y)^2}) \quad [3]$$

E( ): 시간에 대한 평균
f1.x : 기준 유저 x 좌표 f1.y 기준 유저 y 좌표
f2.x : 짝 유저 x 좌표 f2.y 짝 유저 y 좌표

봇 그룹은 레벨이 높은 캐릭터가 다른 캐릭터들을 키워서 수를 늘리는 방식을 채택하고 있기 때문에 봇 그룹 내의 레벨차가 크다. 반면에, 사람으로 이루어진 그룹은 레벨이 비슷한 사람끼리 같이 활동하기 때문에 레벨차가 크지 않고 비슷하다. 그룹 유저간의 레벨 차이를 측정하기 위한 식은 다음 식 4 와 같다.

$$\text{레벨 차이} = \mathrm{E}(|\mathrm{f1.level} - \mathrm{f2.level}|) \quad [4]$$

f1.level : 기준 유저의 레벨
f2.level : 짝 유저의 레벨

사람 그룹은 항상 서로 파티상태를 유지하고 활동하지 않지만 봇 그룹은 항상 서로 파티상태를 유지하고 활동한다. 봇들과 달리 유저들은 상점을 들르거나 개인 퀘스트를 깨는 등 개인활동이 있기 때문이다. 사람들은 사냥 활동이 아닌 개인활동을 할 때 파티를 해제하고 있기 때문에 동시 접속 중일 때 둘 다 파티를 맺고 있을 확률로 봇 그룹인지 사람 그룹인지 판별할 수 있다. 파티 정보를 측정하기 위한 식은 다음 식 5 와 같다.

$$\text{파티 정보} = \frac{party\ time}{s} \quad [5]$$

party time : 둘 다 파티중인 시간
s : 동시접속시간

4. 실험

4.1 데이터 셋

본 논문에서 제안하는 봇 그룹 탐지 모델을 국내 온라인게임인 열혈강호의 실제 데이터에 적용하여 그 효용성을 평가하였다. 한 달 동안 게임 서버에서 얻은 로그 데이터를 사용했다. 로그 데이터는 각 플레이어에 대해 5 분에 한번씩 얻어졌다.

표 2. 플레이어와 로그의 수

그룹	로그의 수	플레이어의 수
유저 로그	754201	584
봇 로그	608122	824
합계	1362323	1408

4.2 실험 설계

게임 데이터에서 수집한 좌표정보에 따라 그룹을 지었다. 이동정보를 기반으로 하기 때문에 전처리 과정을 통해 이동을 하지 않는 데이터를 제거하였다. 위치정보의 분산을 기준으로 데이터를 제거한 뒤 같은 그룹에 있던 횟수로 짝을 지었다. 이 때 유저는 유저끼리, 봇은 봇끼리 짝을 지을 확률을 측정하였고 그 결과는 다음 그림 3 과 같다.

가로축은 같은 그룹에 있던 횟수를 의미한다. 약 한 달간의 로그에서 얻었을 때, 같은 그룹으로 분류

된 score 가 300 이 넘으면 봇은 봇끼리, 사람은 사람끼리 분류될 확률이 약 97%로 사람은 사람끼리, 봇은 봇끼리 함께 행동한다는 것을 확인할 수 있다.

이렇게 분류된 그룹들을 통해 위 3.3 절에서 설명한 특성을 기반으로 봇 탐지 모델을 생성하고 이를 평가하였다.

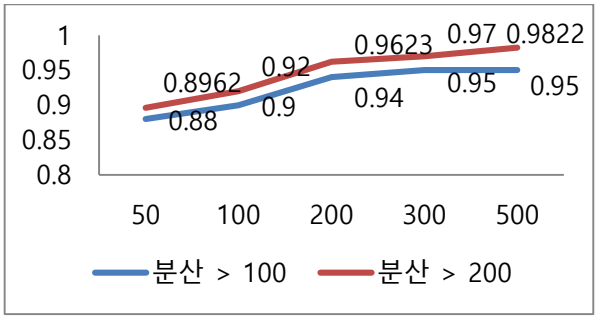


그림 3 봇-봇 유저-유저 그룹 분류 확률

게임 데이터에서 수집한 레벨, 좌표, 접속정보, 위치정보로부터 특성을 추출하여 로지스틱 회귀분석, 서포트 벡터 머신의 분류 알고리즘을 통해 분류모델을 생성하였고, 데이터 셋을 랜덤하게 10 등분하여 9 개의 서브셋으로 학습하고 나머지 1 개 서브셋으로 평가를 수행하는 10-fold cross validation 기법을 이용하여 분류 모델을 검증하였다.

더 큰 움직임을 보이는 데이터를 뽑을수록, 더 높은 score 의 그룹을 채택할수록 정확도가 더욱 높아지는 것을 볼 수 있었다. 좌표의 분산이 100 이하인 데이터를 제거한 것과 분산이 200 이하인 데이터를 제거한 것에 대하여 실험을 수행하였다. 또, score 에 따른 영향을 측정하기 위하여 score 가 100 이상인 것, 200 이상인 것에 대한 두 가지 실험을 하였다.

4.3 실험 결과

먼저 표 3 에서 각 특성에 대한 Accuracy 를 볼 수 있다. Accuracy 는 전체 그룹에서 봇 그룹은 봇 그룹으로, 사람 그룹은 사람 그룹으로 예측 할 비율을 말한다. 각각 특성들의 Accuracy 도 높지만 여러 특성을 동시에 사용했을 때 가장 높은 정확도를 보였다. 특히 레벨 차이 특성과 파티 정보 특성이 가장 큰 영향을 주는 것을 확인할 수 있었다.

표 4 는 각 분류기에 대한 Precision, Recall 을 나타낸다. Precision 은 봇 그룹으로 예측한 것들 중 실제 봇 그룹일 확률을 의미한다. Recall 은 실제 봇 그룹 중 봇으로 예측한 비율을 의미한다. 실험결과 세가지 기법 모두 높은 성능을 보이고 있다. 특히 로지스틱 회귀분석을 사용했을 때 가장 좋은 성능을 보이고 있다.

5. 결 론

현재까지의 게임 봇 탐지 기술들은 대부분 게임 플레이어의 행동변화를 집중적으로 보았다. 개인이 아닌 파티나 길드와 같은 그룹으로 탐지를 했을 때는 활강과 같은 게임에 종속되는 특성을 사용하거나 길드 내부 정보와 같은 기존 로그데이터를 이용하여 범용적으로 적용할 수 없는 한계가 있었다. 본 논문에서는 다른 MMORPG 에도 사용할 수 있는 범용적인 상태특성을 사용하였다. 위치 별로 그룹을 지어 각 상태의 차이특성을 이용한 제안 모델은 score 가 높을 때 99%이상의 높은 Precision 을 보였다.

본 연구에서 제안하는 탐지 방법은 그룹단위로 탐지를 하기 때문에 한 봇 그룹이 탐지가 되면 이와 관련된 다른 봇들까지 한꺼번에 탐지가 되기 때문에 더 큰 효과를 볼 수 있을 것이라 생각된다. 향후에 둘간의 관계가 아닌 셋 이상의 그룹 네트워크를 만들어 탐지한다면 더 좋은 결과를 기대할 수 있을 것이다.

표 3. 특성별 Accuracy

	분산> 100 이고 score > 100		분산> 200 이고 score > 200	
분류 기법	로지스틱 회귀	SVM	로지스틱 회귀	SVM
서로의 거리	0.7893	0.8361	0.8370	0.8783
레벨 차이	0.8796	0.8813	0.9348	0.9065
동시접속률	0.8010	0.8194	0.7978	0.8022
파티 정보	0.8779	0.8662	0.8957	0.887
거리+레벨+동시접속률	0.8729	0.8930	0.9348	0.9134
거리+동시접속률+파티	0.9515	0.9498	0.9674	0.9652
레벨+파티	0.9565	0.9632	0.9891	0.9826
모든 속성 사용	0.9749	0.9716	0.9935	0.9891

표 4. 실험 결과

	분산 > 100 이고 score > 100		분산 > 200 이고 score > 200	
분류 기법	Precision	Recall	Precision	Recall
로지스틱 회귀분석	0.9802	0.9611	0.9948	0.9897
SVM	0.9688	0.9650	0.9846	0.9897

Acknowledgement

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 산업융합원천기술개발사업(정보통신)의 일환으로 수행하였음(10041244, 스마트 TV 2.0 소프트웨어 플랫폼). 또한, 본 연구는 미래창조과학부 및 정보통신기술진흥센터(IITP)의 SW 컴퓨팅산업원천기술개발사업의 일환으로 수행하였음(B0101-15-0559, 디지털 소상공인 지원을 위한 지역 비즈니스 전략 분석 및 맞춤형 영상홍보 창작 SW 플랫폼 개발).

참고문헌

[1] Ah Reum Kang a, Jiyoung Wooa, Juyong Park b, Huy Kang Kim, "Online game bot detection based on party-play log analysis," Computers and Mathematics with Applications , Vol.65, No.9, pp.1384–1395, 2013

[2] Harang Kim, Huy Kang Kim, "Research on online game bot guild detection method," Journal of The Korea Institute of Information Security & Cryptology, Vol.25, No.5, pp.1115-1122,2015

[3] Byung Il Kwak, Huy Kang Kim, "A survey and categorization of anomaly detection in online games", Journal of the Korea Institute of Information Security and Cryptology , Vol.25, No.5, pp.1097-1114, 2015

[4] Jina Lee, Jiyoun Lim, Wonjun Cho and Huy Kang Kim, "In-Game Action Sequence Analysis for Game BOT Detection on the Big Data Analysis Platform", Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Vol.2, pp.403-414, 2015

[5] Ahmad, M. A., Keegan, B., Sullivan, S., Williams, D., Srivastava, J., and Contractor, N., "Illicit bits: Detecting and analyzing contraband networks in Massively Multiplayer Online Games", Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing, pp.127-134

[6] Yeounoh Chung, Chang-young Park, Noo-ri Kim, Hana Cho, Taebok Yoon, Hunjoo Lee and Jee-Hyong Lee, "Game Bot Detection Approach Based on Behavior Analysis and Consideration of Various Play Styles", ETRI Journal, Vol. 35,No. 6, pp.1058-1067, 2013

[7] Jin Lee, Sung Wook Kang, Huy Kang Kim, "A study on hard-core users and bots detection using classification of game character's growth type in online games", Journal of the Korea Institute of Information Security and Cryptology, Vol. 25, No.5, pp.1077-1084, 2015,

[8] M. van Kesteren, J. Langevoort, and F. Grootjen, "A step in the right direction: Bot detection in mmorpgs using movement analysis", in Proc. of the 21st Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2009), 2009.

Matlab 기반 차량제어시스템의 측정 데이터 분석도구의 구현

이창호

현대오트론
경기도 성남시 분당구 판교로 344 엠텍 IT 타워
changho@hyundai-autron.com

요약: 자동차에 탑재되는 제어기 SW 의 복잡도 증가로 인하여 다양한 검증활동들이 수행되고 있다. 본 논문에서는 시스템 검증 또는 실차검증 수행중에 측정한 변수값들의 상관관계를 분석하여 사용자가 입력한 TestCase 의 Pass/Fail 결과를 그래프와 함께 자세히 표시해주는 도구를 구현하였다. 본 논문에서 구현한 도구는 Matlab GUI 기반으로 excel 파일을 통한 TestCase 입력이 가능하도록 구현되었으며 TestData 와 TestCase 입력 모듈, 분석모듈, 결과출력 모듈의 세부분으로 구성되었다.

핵심어: 데이터 상관관계 분석 도구, ASCII 분석 , Matlab, Measure Data Analyzer.

1. 연구배경

최근 자동차는 높은 수준의 전자장치와 그 전자장치를 제어하는 소프트웨어를 함께 개발해야 하는 혁신적인 과정에 있음에 따라, 자동차 전장품의 비율이 70~80%에 이르고, 한 대의 차량에 100 개에 육박하는 ECU 가 장착되는 등 그 복잡도가 증가하고 있다[1]. 이러한 복잡도는 극히 일부 선택된 경우의 수만을 미리 정의하여 검증하는 테스팅 기법만으로 극복될 수 없으며, 좀 더 엄밀하고 다양한 검증방식에 대한 이해와 적용이 필요하다[1]. 또한 복잡도 증가로 인하여 신속하고 효율적인 검증방법도 요구되고 있다고 볼 수 있다[2].
MDA(Measure Data Analyzer)와 같은 기존의 측정데이터 분석 도구들은 그래픽 화면, 모듈 방식의 화면 및 통계화면을 제공하지만 사용자가 시간에 따른 변수값의 변화를 수동으로 확인해야 하는 불편함이 있다[3]. 본 논문에서는 확인이 필요한 변수값들의 관계를 TestCase 로 작성하여 실행하면 도구가 Pass/Fail 결과를 자동으로 분석하여 도구에 대한 활용성을 높여주고 검증소요시간을 줄여줄 수 있는 도구를 구현하였다.

2. 기본 개념

2.1 측정 데이터 형식

본 도구에서 사용되는 측정 데이터는 그림 1 과 같이 구성되어 있다. 첫번째 행은 헤더부분으로 시간항목과 변수들의 이름을 나타낸다. 첫번째 열은 측정된 시간값을 나타내며 두번째 행부터 변수 이름과 측정값들이 나열된다.

	A	B	C	D	E	F
1	time	tmReq	tmSensor	ShiftP	engRPM	engSensor2
2	0.352	0	10.375	5	2830	1
3	0.362	0	18.5	4	2400	1
4	0.372	0	27.125	4	2300	1
5	0.383	1	27.125	4	2290	1
6	0.395	3	29.1	4	2280	0
7	0.405	1	29.1	4	2270	0
8	0.413	1	31.5	4	2260	0
9	0.423	1	31.5	4	2250	0
10	0.435	1	31.5	3	1980	0
11	0.445	3	24.1	3	1940	0
12	0.455	1	24.1	2	1590	0
13	0.465	1	24.1	1	900	0
14	0.475	1	18.9	1	900	0
15	0.485	1	18.9	2	1700	0
16	0.495	3	18.9	3	1900	0
17	0.505	1	33.2	3	2100	0
18	0.515	1	33.2	2	1900	1
19	0.525	1	33.2	1	1100	1
20	0.535	1	20.1	1	1000	1
21	0.545	1	20.1	1	900	1

그림1. 측정 데이터 형식

2.2 TestCase 입력파일

TestCase 입력파일은 그림 2 와 같이 엑셀형식의 파일로 작성 가능 하다. 첫번째 행은 TestCase 번호를 의미하며 두번째 행은 mode 를 지정할 수 있다. 세번째 행은 테스트케이스에대한 시간제한을 설정하는 항목이다. 테스트케이스 작성시에는 최소 1 개 이상의 변수가 필요한데 이 변수를 Main Variable 이라고 하며 테스트케이스 입력파일의 네번째부터 여섯번째 항목에 설정할 수 있다. 그 이후의 항목은 사용자가 비교를 원하는 변수를 설정할 수 있으며 최대 2 개까지 설정가능하다.

No	mode	timeLimit	MainVar	MainOp	MainValue	Cond1Var	Cond1Op	Cond1Value
1	rising	0.03	tmReq	=	3	tmSensor	>	30
2	rising	0.03	tmReq	=	2	tmSensor	>	30
3	falling	0.03	tmReq	=	1	tmSensor	>	30
4	range	0	tmReq	>	2			

그림2. TestCase Sample

3. 분석 실행 및 결과 확인

3.1 도구의 설계

본 논문에서 구현한 도구는 file 선택 모듈, 분석모듈, 결과출력의 세부분으로 구성되었다. 도구에서 처리 가능한 mode 는 3 가지 종류가 있는데 지정한 변수가 rising egde 일 때, 다른 변수값들을 확인할 수 있는 rising mode, 지정한 변수가 falling edge 일 때, 다른 변수값들을 확인할 수 있는 falling 모드가 있다. 그리고 range 모드는 전체 시간동안의 변수값과 비교대상 값의 크기를 비교해주는 모드 이다. Rising/falling mode 는 선택한 변수가 rising 또는 Falling edge 일 때 특정 시간내에 추가로 2 개 변수값의 범위를 확인할 수 있도록 구현하였다. ‘Result’ 윈도우의 Graph Index 를 클릭하면 오른쪽의 graph window 에 나타난 Graph 를 참고하여 Pass/Fail 여부를 쉽게 확인 할 수 있도록 구현하였다.

3.2 환경 설정

환경설정은 그림 3 에 나타낸 부분에서 할 수 있다. 분석대상파일은 ‘DataFile’ 항목의 ‘File 선택’ 버튼을 클릭하여 선택하고 ‘Load’ 버튼을 클릭하여 Load 한다. TestCase 파일은 ‘TestCase’ 항목의 ‘File 선택’ 버튼을 클릭하여 선택하고 ‘Load’ 버튼을 클릭하여 Load 한다. 그림 3 의 ‘TestCase’ 윈도우에 TestCase 가 정상 로드 된 것을 확인할 수 있다. Sample TestCase 는 rising, falling, rage 각 1 항목씩을 선정하였다.

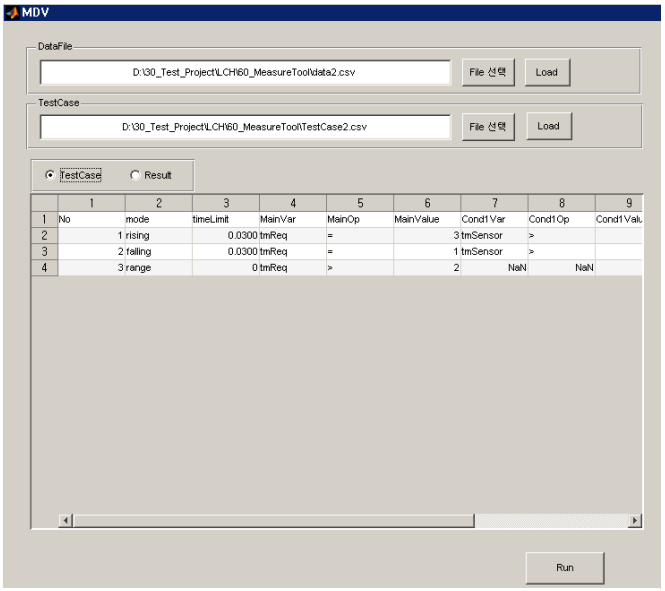


그림 3. 파일선택 UI 및 TestCase UI

3.3 결과 확인

분석대상파일과 TestCase 파일이 정상 로드되었다면 ‘Run’ 버튼을 클릭하여 Test 를 시작한다. 간단한 실행로그들을 그림 4 의 LogBox 에서 확인할 수 있다.

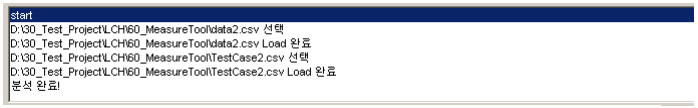


그림 4. LogBox

정상 수행이 되었다면, ‘Result’ 윈도우에서 실행결과를 확인 할 수 있다. Sample TestCase 의 실행결과를 그림 5 에 나타내었다. 첫번째 열은 TestCase 번호를 의미한다. 두번째 열은 전체 결과를 나타내거나 TestCase 에 대한 상세 결과번호를 나타낸다. 측정된 데이터에서 어떤 변수가 상승엣지일 때의 경우는 여러 번 나타날 수 있으므로 그런 경우는 ‘Result_번호’로 나타내고 각 결과들도 표시하였다. 그림 5 에서 TC_1 번의 결과를 확인해보면, TestCase1 번은 tmReq 의 변수가 3 으로 상승할 때 0.03 초 내에 tmSensor 의 변수가 30 보다 크고 ShiftP 의 변수가 0 보다 큰 경우가 있는지를 검사하는 테스트케이스 이다. Sample Data 에서는 이런 경우가 3 번 있었고 두번째 경우에는 Fail 이 발생하였는데 이 경우로 인하여 TestCase1 번은 Fail 로 처리되었다. 네번째 열의 ‘Graph Index’ 항목을 클릭하면 해당 결과에 대한 그래프를 볼 수 있다. 그림 6 에 TestCase1 번의 두번째 경우에 대한 그래프를 나타내었다. 0.445 초에 tmReq 변수가 3 으로 상승하나 그 후 0.03 초 내에 tmSensor 값이 30 보다 커지는 경우가 없기 때문에

Fail 로 처리되었다.

TestCase Result

	1	2	3	4
1	TestCase	Sub Result/Result No	Pass/Fail	Graph Index
2	TC_1	Fail		
3		Result_1	Pass	1_1_3
4		Result_2	Fail	1_2_3
5		Result_3	Pass	1_3_3
6	TC_2	Pass		
7		Result_1	Pass	2_1_3
8		Result_2	Pass	2_2_3
9		Result_3	Pass	2_3_3
10	TC_3	Fail		
11		Result_1	Fail	3_1_3

그림 5. Sample TestCase의 실행 결과

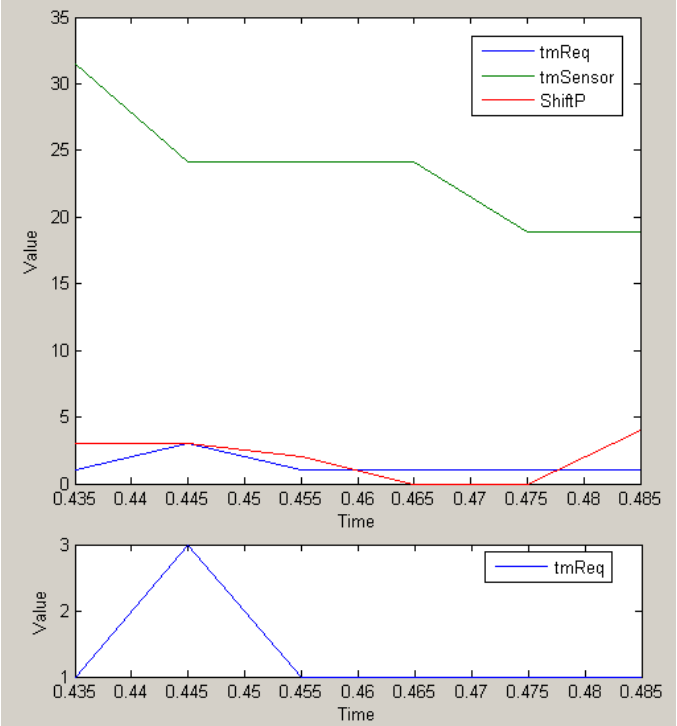


그림 6. Sample TestCase 1의 2번째 결과에 대한 그래프

4. 결론 및 향후 연구

본 논문에서는 Matlab GUI 기반으로 측정데이터 분석도구를 구현하였다[4]. 본 논문에서 구현한 도구는 기존 측정데이터 분석도구들이 가지고있는 사용자가 수동으로 그래프 등을 확인해야하는 불편함을 개선하여 사용자가 측정된 데이터 또는 변수값들의 관계를 TestCase 로 입력할 수 있고 실행결과를 Pass/Fail 형태로 확인이 가능하다는 점에서 그 개발의 의미를 가진다고 할 수 있다. 특히 실행된 TestCase 를 재사용할 수 있다는 점이 기존 측정데이터 분석도구대비 큰 장점이라고 볼 수 있다.
향후에는 검증모드를 추가하고 확인 가능한 변수의 수를 늘리는 것과 더욱 다양한 기능을 제공하는 그래프 컴포넌트 및 결과 리포트 출력 기능의 구현을 진행할 예정이다.

참고문헌

[1] 최윤자, 이우진 (2013), “ 자동차 전장용 소프트웨어 검증 동향”, 정보과학회지, 31(5), 18-24
[2] Andrew Burnard, “ Verifying and Validating Automatically Generated Code”, *Int. Automotive Conference (IAC)*, pp. 71-78, 2004.
[3] Measure Data Analyzer, [Online].Available : http://www.etas.com/ko/products/mda.php
[4] Matlab, [Online].Available : http://www.mathworks.co.kr/products/matlab/

카페 좌석 관리 시스템

박평우[1], 임종찬[2], 노우리[2], 문은미[3], 이석원[1]

아주대학교 소프트웨어융합학과[1], 아주대학교 미디어학과[2], 아주대학교 사이버보안학과[3]
경기도 수원시 영통구 월드컵로 206

bryan921105@gmail.com, yjc0301@ajou.ac.kr, yuwooloe@naver.com, pqpq0404@naver.com, leesw@ajou.ac.kr

요약: 본 프로젝트는 카페의 잔여 좌석을 자동으로 인식하여 해당 카페의 점주 및 점원, 그리고 고객들에게 실시간으로 알려주는 시스템이다. 본 시스템은 소프트웨어 공학적 관점으로 설계 및 구현되었고, 타이젠, 아두이노, 라즈베리파이, 서버로 구성되었다.

1. 서론

우리나라 사람들은 카페인 중독에 시달리고 있다. 한국 농림축산 식품부서와 한국 농수산 식품유동 공사에서 조사한 가공식품 소비현황 보고서에 따르면, 주당 소비 빈도가 가장 높은 음식은 커피로 일주일에 12.2 회, 1 인 당 하루 2 잔, 다른 음식을 먹는 횟수보다 배로 섭취한다고 한다. 또한, 번화가나 대학가 근처에서 카페 앞에 길게 줄 서있는 사람들, 커피를 들고 다니는 사람들도 심심치 않게 볼 수 있으며, 대학교 시험기간에는 대학가의 카페가 문전성시를 이루고 있다. 이는 요즘 학생들의 학습 형태의 변화로 인해, 혼자 앉아 책을 보며 공부하던 과거와 달리, 노트북을 이용하거나 팀을 이뤄 토론하며 학습하기 때문이다. 더욱이, 강의 자료가 대부분 PPT 나 PDF 등의 컴퓨터 파일로 작성되어 있고, 온라인 강의를 듣거나 조별 발표를 준비하여야 하는 경우가 많기 때문에 노트북을 쓸 일이 잦아지게 된 것이다. 따라서, 조작에 따른 소음 등으로 인해 눈치 보지 않고 공부하기 위하여 카페를 찾는 사람들이 늘고 있는 것이다. 서울 신촌 일대에는 맞춤형 공부 카페도 등장하는 추세이고, 카페는 점차 마시는 곳이 아닌 공부하는 곳, 길게 대화 및 토론을 나눌 수 있는 장소로 변해가고 있다. 이러다 보니, 카페가 아무리 많더라도 카페를 가려는 사람들의 수요를 카페의 좌석이 모두 충당하지 못하는 지경에까지 이르게 되었고, 카페를 가려는 사람들은 좌석에 앉지 못하고 자리가 빈 곳을 찾기 위하여 몇 번이나 여러 카페에 들락날락 거리고 헤맬 수 밖에 없게 되었다.

현재, 대부분의 카페에서는 원초적인 방법으로 좌석을 관리하고 있다. 이로 인해 카페를 찾는 고객뿐만 아니라, 매장 점원들도 어려움을 겪고 있고, 카페 점주도 금전적으로 손해를 볼 수 밖에 없는 실정이다. 고객들은 언제 어느 카페의 좌석에 착석할 수 있는 지 알기 어렵기 때문에, 직접 매장에 방문하여 두 눈으로 확인한 후, 자리를 이용할 수 있기 때문에 큰 불편함이 따른다. 현재 실생활에서 상용화 된 유사 시스템은 도서관이나 교통 시설 등에서 볼 수 있는데, 이 시스템은 특정 기관이나 매장에서만 사용할 수 있는 것으로 다소 협소한 범위 내에서만 유용하고, 본 프로젝트의 시스템과 용도 및 목적에 명백한 차이가 있다. 또한, 매장 점원들은 좌석을 관리하려면 매번 카페에 어떤 고객이 어느 좌석을 이용했고 언제 매장을 나가는지 직접 눈으로 확인 할 수 밖에 없다. 카페의 좌석을 일일이 확인하여 수동으로 일정 주기마다 반복하여 정보를 취득할 수 밖에 없는 현재의 상황은 노동력의 큰 손실과 자원의 낭비로 이어진다. 이는 당사자인 매장의 직원이 좌석을 항상 청결한 상태로 유지하기에 매우 힘이 들고, 미처 더러워진 자리를 발견하지 못할 수도 있다. 카페의 청결성은 결과적으로 매출과 직결되기에 카페의 점주도 손해를 볼 수 밖에 없다.

오늘날, 상용화가 가능하고 이미 개발된 유사 시스템은 본 도메인에 적용하기에 아직 무리가 따른다. 그 이유는 크게 두 가지로 분류할 수 있는데, 첫 번째로 자동적으로 좌석을 인식할 수 있는 시스템이 아니기 때문이다. 예를 들어, 학교 도서관 및 영화관 같은 시설에서 사용되는 대부분의 시스템은 매번 고객이 자리에 착석하거나 자리에서 나갈 때 어떠한 명령이나 정보를 시스템에 직접 전달해야 하는 큰 불편함이 따른다. 이는 카페나 다른 상업적 매장에서 이용하기에 고객 측면에서 불편함만 가중될 뿐이다. 또한, 카페의 점원 및 점주도 좌석을 제대로 관리하는데 여전히 어려움이 따르게 된다. 다음 두 번째 문제는 시스템 통합적 문제이다. 효율적인 카페 좌석 관리 시스템은 기본적으로 한 카페에서만 이용 가능한 것이 아니라 여러 카페에서 동시적으로 이용할 수 있는 시스템이어야 한다. 이는 한 카페의 좌석만을 고객이 확인하고 이용할 수 있는 것이 아니라, 여러 카페 중 남는 모든 좌석을 고객이 실시간으로 확인할 수 있어야 하는 것이다. 이러한 이유로

현재 유사 시스템은 카페에 도입될 수 없는 실정이다. 즉, 현 카페의 좌석 관리는 빈 좌석 현황을 자동적으로, 실시간으로 알 수 없기 때문에 고객과 카페의 점원 및 점주 모두가 불편함을 겪고 있는 것이다. 이는 카페 이해관계자들의 요구사항을 충분히 반영하지 못한 결과라고 판단할 수 있다.

본 시스템의 목적은 크게 두 가지로 설명할 수 있다. 첫 번째는, 카페를 이용하려는 고객들에게 빈 좌석을 빠르고 쉽게 찾을 수 있도록 도와주는 것이다. 프로젝트의 결과물로써 빈 좌석이 있는 제일 가까운 카페를 알려주고, 해당 카페에 대한 대략적인 정보도 알려준다. 또한, 전기 콘센트, 테이블의 크기, 한 테이블 당 좌석의 수 등과 같이 카페에 대한 구체적인 정보도 포함하여 알려주기 때문에 더욱 편하게 카페를 이용할 수 있도록 도와준다. 이로 인해 고객들은 자리가 없는 카페에 들어가 좌석을 확인할 필요가 없어지고 해당 카페의 주변 자리와 분위기 또한 예상해볼 수 있다. 두 번째는 카페 매장에서 체계적으로 좌석을 관리할 수 있도록 도와주는 것이다. 본 시스템은 카페의 점원이나 점주가 계산대에서 좌석을 한번에 둘러볼 수 있게 해준다. 따라서 수동적으로 좌석을 확인하고 청소하며 관리해야만 하는 작업을 피할 수 있고, 전체적인 매장 좌석 관리를 훨씬 더 용이하게 만들 수 있다.

이후, 본 프로젝트의 전반적인 과정은 해당 목적을 달성하기 위하여 실질적으로 카페라는 도메인 내 이해관계자들의 요구사항을 나열하고, 철저히 분석한 후, 이를 기반으로 소프트웨어 공학적 관점에서 설계, 개발 및 구현하였다.

2. 시스템 요구사항 분석 및 설계

본 시스템에 대한 요구사항은 크게 세 부분으로 나뉜다. 첫 번째, 고객 요구사항은 이용자에게 초점을 맞춘 요구사항으로써, 타이젠 어플리케이션을 통해 해당 시스템을 이용하는 고객과 라즈베리파이를 통해 관리하는 점원 및 점주를 기반으로 작성된 명세서이다. 카페 좌석 관리 시스템은 기본적으로 각 카페에 대한 서비스 정보를 충실하게 제공하여야 한다. 이 정보는 카페 매장의 위치, 좌석의 위치, 총 좌석의 개수, 잔여 좌석의 개수, 음료의 가격 및 정보, 그리고 이벤트 등을 포함한다. 시스템은 해당 정보를 실시간으로 고객에게 자동적으로 알려주고, 매장 점원이나 점주가 이를 직접 공지하고 글을 게시할 수도 있어야 한다. 두 번째는 시스템 요구사항으로, 시스템의 동작 수행에 대한 내용을 작성한 것이다. 이는 프로젝트의 주목적격인 좌석 관리의 인식 및 감지 부분으로써, 센서를 통해 시스템이 좌석의 사용 여부를 판단하는 것, 본 프로젝트에 부합한 서비스 제공을 위하여 시스템이 갖추어야 하는 기능 등이 담겨있다. 마지막 세 번째는 도메인 요구사항이다. 이는 시스템 외적인 부분과 관련된 것으로 안전 사항, 세부적인 센서의 요구사항 등의 내용으로 구성된다. 아래 표는 요구사항을 간략하게 나타낸 것이다.

서비스	내용
회원가입	타이젠 어플리케이션을 사용하기 위한 절차로써 서버에 정보를 저장한다.
로그인	회원가입 이후, 아이디와 비밀번호를 통해 인증 받는다.
카페 리스트 확인	타이젠 어플리케이션을 통해 카페 좌석 관리 시스템을 이용하는 카페를 보여준다.
실시간 좌석 확인	실시간으로 해당 카페의 잔여 좌석 현황을 보여준다.

표 1 타이젠 요구사항

서비스	내용
좌석 사용 여부	해당 좌석의 사용 여부를 인식 및 감지하여 서버로 정보를 전달한다.

표 2 아두이노 요구사항

서비스	내용
실시간 좌석 확인	실시간으로 해당 카페의 잔여 좌석 현황을 보여준다.

표 3 라즈베리파이 요구사항

서비스	내용
타이젠 통신	타이젠 어플리케이션에서 요청하는 정보를 송신한다.
아두이노 통신	아두이노 기기에서 인식 및 감지한 값을 받아 저장한다.
라즈베리파이 통신	라즈베리파이에서 요청하는 정보를 송신한다.

표 4 서버 요구사항

요구사항 분석 과정은 전체적인 시스템에 대하여 기능 및 서비스로 구분한 후, 유즈케이스 다이어그램 작성 작업을 통해 수행하였다.

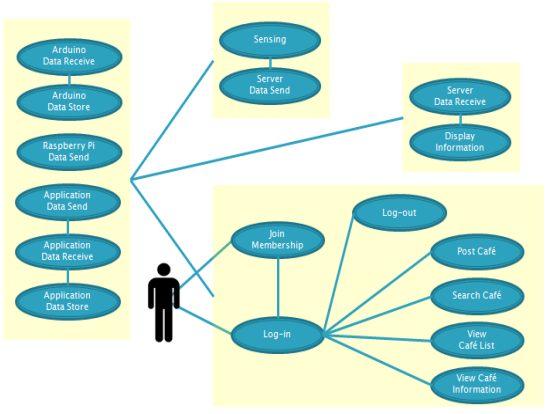


그림 1 유즈케이스 다이어그램

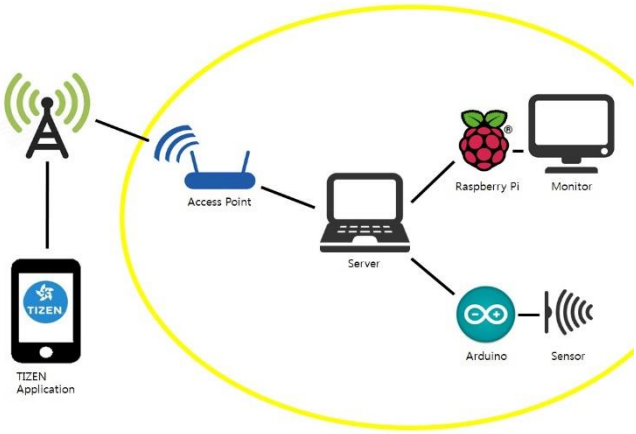


그림 2 전체 시스템 설계

카페 좌석 관리 시스템의 전체적인 설계는 위 그림 2와 같다.

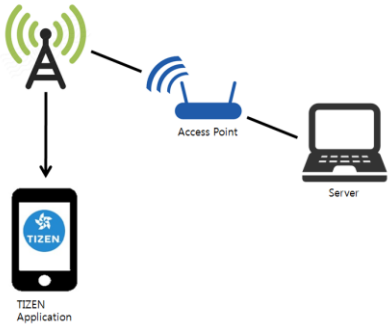


그림 3 타이젠 설계 부분

그림 3은 전체 시스템 설계 중 타이젠 부분을 나타낸 그림이다. 이 시스템 부분은 타이젠 어플리케이션을 통해 통신하여 고객에게 카페에 대한 서비스를 제공하는 부분으로, 타이젠 어플리케이션과 서버로 구성된다. 이 때 시스템은 앞서 말한 것과 같은 서비스 정보를 제공하는 역할을 수행하며, 모든 정보를 가지고 있는 서버는 타이젠 어플리케이션에서 고객이 원하는 정보를 요청할 때마다 통신하여 정보를 송수신한다. 또한, 서버와 타이젠 어플리케이션은 같은 연결 통신망에 있지 않은 경우에도 외부 통신망을 이용하여 언제든지 요청에 응답할 수 있도록 설계하였다.

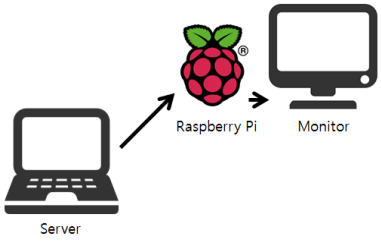


그림 4 라즈베리파이 설계 부분

그림 4는 라즈베리파이 부분에 대한 설계를 간단하게 나타낸 그림이다. 이 시스템 부분은 카페 점원 및 점주가 수월하게 카페 좌석 관리를 할 수 있도록 하는 부분으로, 현재 카페 좌석을 해당 매장의 계산대에서 보여주는 역할을 수행한다. 이는 라즈베리파이, 모니터, 서버로 구성되어 타이젠 어플리케이션 부분과 같이 실시간으로 현재 카페 내 총 좌석의 수, 잔여 좌석의 수, 사용 중인 좌석의 수 및 각각 해당 좌석의 위치를 확인할 수 있도록 한다.

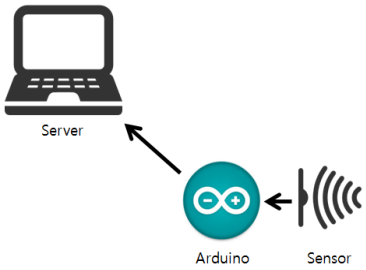


그림 5 아두이노 설계 부분

그림 5는 아두이노 설계 부분으로 아두이노 센서, 아두이노, 서버로 구성하였다. 이 시스템 부분은 좌석을 감지하고 판별하여 정보를 저장하는 부분이다. 즉, 직접적으로 좌석에 대하여 비어있는지 또는 사용 중인지 판단하고 실시간으로 그 값을 입력 받는 역할을 수행하며 지속적으로 그 값을 갱신하여 정보를 저장한다.

이와 같이 총 세 가지 시스템 부분의 설계는 전체적으로 하나의 시스템으로 통합 및 형성되어 카페 좌석 관리 시스템의 목적에 부합하도록 하였으며, 효율적으로 구현할 수 있도록 설계하였다.

3. 시스템 동작

본 프로젝트의 시스템은 철저히 위 시스템 설계 내용을 따르고 있고, 각각 고객과 점원 및 점주 측면에서의 요구사항을 기반으로 작성되었으며, 이에 대한 분석 과정을 통하여 동작한다.

시스템 동작 과정은 크게 상황인지 모형, 상호작용 모형, 구조 모형, 동적 모형 및 설계 패턴 등으로 분류하여 작성하였다. 그 중 상호작용 모형인 시퀀스 다이어그램은 다음 그림과 같다.

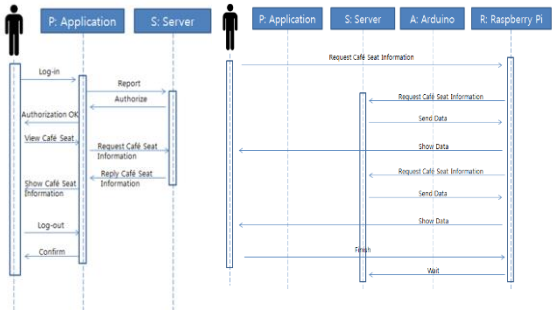


그림 6 시퀀스 다이어그램

위 그림 6의 좌측 그림은 고객이 타이젠 어플리케이션을 통해 수행한 로그인 과정을 나타낸 것이고, 우측 그림은 카페 점원 및 점주가 라즈베리파이를

통해 수행한 카페 좌석 현황 확인 과정을 나타낸 것이다. 이를 기반으로 카페 좌석 관리 시스템의 동작 과정을 알아볼 수 있다.

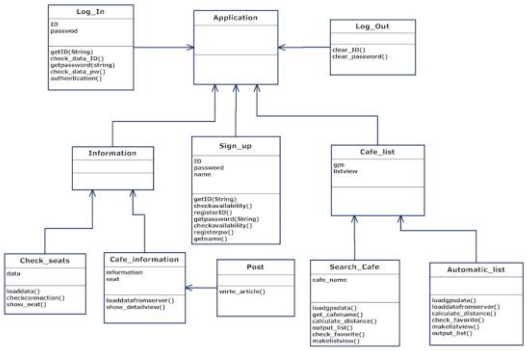


그림 7 클래스 다이어그램

그림 7은 구조 모형의 클래스 다이어그램을 나타낸 것으로써 본 프로젝트인 카페 좌석 관리 시스템의 클래스를 구체적으로 표현한다.

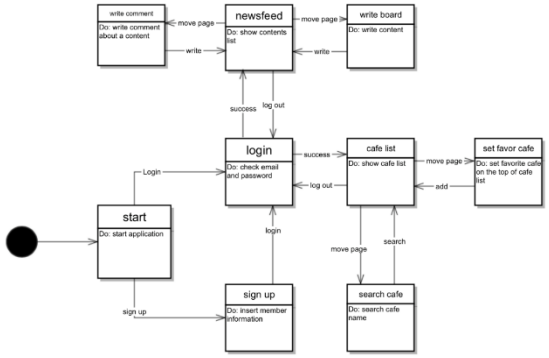


그림 8 타이젠 스테이트 머신 다이어그램

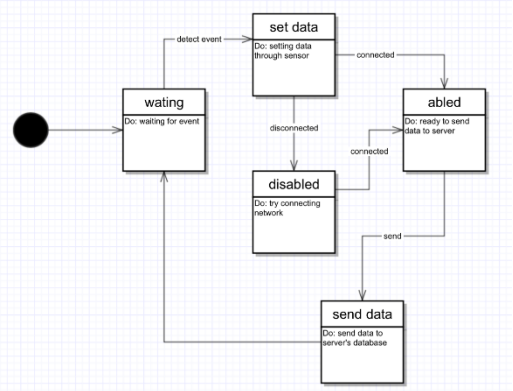


그림 9 아두이노 스테이트 머신 다이어그램

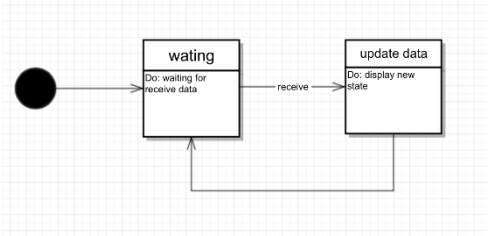


그림 10 라즈베리파이 스테이트 머신 다이어그램

위 그림 8, 그림 9, 그림 10은 스테이트 머신 다이어그램으로, 객체의 상태와 상태를 변화시키는 이벤트를 표현한 것으로써, 특정 객체의 생명주기 동안에 가질 수 있는 조건, 생명주기 내 수행하는 활동에 대하여 작성한 것이며, 각각의 그림은 순서대로 타이젠 어플리케이션, 아두이노, 라즈베리파이에 대한 것이다.

이 외에도 많은 모형과 다이어그램을 통해 시스템 동작 과정을 설계하였고 이를 기반으로 동작한다.

시스템 동작 시나리오는 고객, 점원 및 점주 두 과정으로 나누어 볼 수 있다. 먼저, 카페 좌석 관리 시스템이 설치된 카페를 방문한, 타이젠 어플리케이션을 설치한 고객의 시나리오는 다음과 같은 순서로 이루어진다. 1. 고객은 타이젠 어플리케이션 사용을 위하여 회원으로 가입한다. 2. 고객은 회원가입 정보를 입력하여 로그인 한다. 3. 타이젠 어플리케이션에서 제시하는 주변 카페 목록을 통해, 고객은 가려고 하는 카페를 선택한다. 4. 고객이 선택한 카페에 대하여 카페의 위치, 세부정보, 현재 잔여 좌석의 수와 위치 및 이벤트 등을 확인한다. 이와 같이 크게 총 네 가지 과정으로 고객의 관점에서 시스템은 동작한다. 이 외, 점원 및 점주의 시스템 동작 시나리오는 다음과 같다. 1. 점원 및 점주는 카페 내 라즈베리파이를 통해 해당 카페의 정보를 확인한다. 2-1. 카페의 정보를 게시 및 수정한다. 2-2. 카페의 좌석 관리를 위하여 실시간으로 좌석 현황을 확인할 수 있다. 이 두 과정으로 카페의 점원 및 점주는 시스템을 충분히 활용할 수 있다.

본 프로젝트의 시스템은 위 기본 시나리오대로 동작하고 이 외에도 추가적으로 다른 시나리오를 진행할 수 있도록 지원한다.

4. 구현 및 결과

프로젝트의 구현 및 결과로써 실제로 구현한 시스템은 다음 그림과 같다.

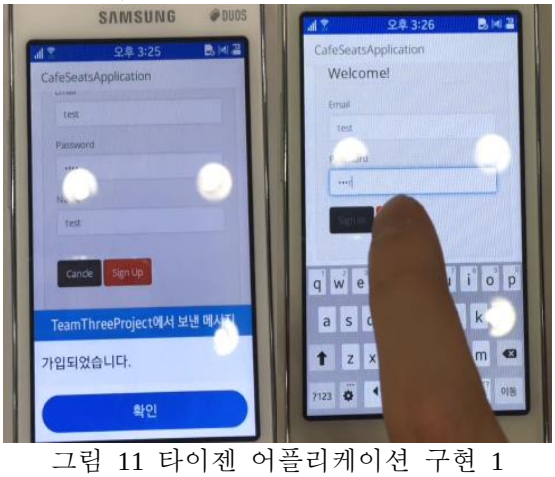


그림 11 타이젠 어플리케이션 구현 1

그림 11 은 타이젠 어플리케이션에서 회원가입과 로그인 과정을 나타낸 그림이다. 고객은 타이젠 어플리케이션을 통해 서버에 회원 정보를 송신한 후, 인증을 받아 로그인 작업을 완료한다.

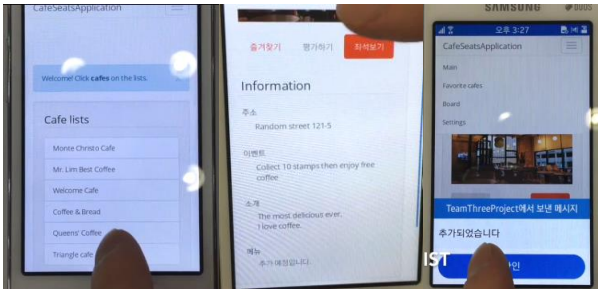


그림 12 타이젠 어플리케이션 구현 2

그림 12 는 개발한 타이젠 어플리케이션에서 카페 목록과 해당 매장의 정보, 즐겨찾기 기능을 나타낸다.

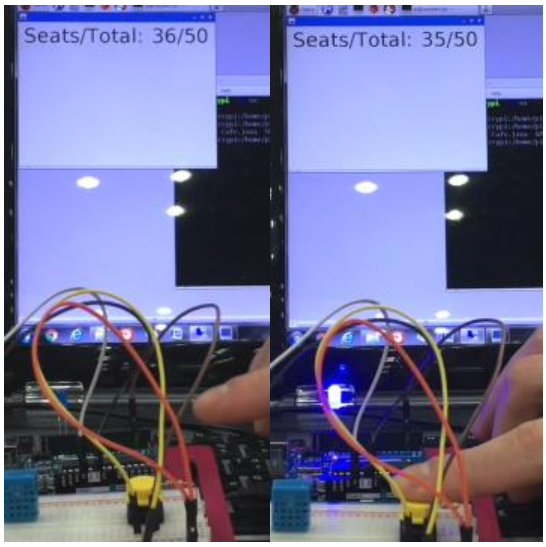


그림 13 라즈베리파이 통신 구현

그림 13 의 좌측 그림은 아두이노 센서를 누르기 전인, 좌석에 고객이 착석하기 전 라즈베리파이의 화면이고, 우측 그림은 아두이노 센서를 통해, 좌석에 고객이 착석한 후 라즈베리파이의 화면이다. 자세히 살펴보면, 좌측의 그림은 전체 50 개 좌석 중 36 개의 잔여 좌석을 화면에 나타내고 있는 반면에, 우측 그림은 전체 50 개 좌석 중 한 자리가 채워진 35 개의 잔여 좌석 수를 화면에 보이고 있다.

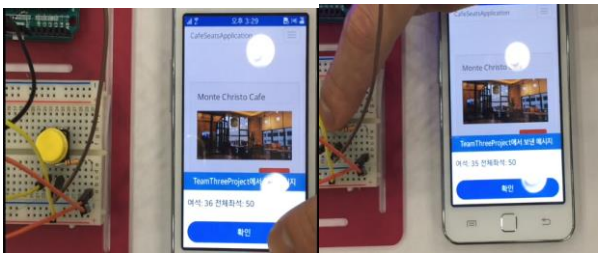


그림 14 타이젠 어플리케이션 통신 구현

그림 14 는 위 그림 13 과 마찬가지로 아두이노 센서를 눌러, 좌석 사용 유무에 따른 결과를 타이젠 어플리케이션을 통해 나타낸 그림이다. 그림 14 의 위 그림은 해당 좌석에 사람이 앉지 않은 경우이고 아래 그림은 좌석에 사람이 앉아 잔여 좌석의 수가 하나 줄어든 모습이다.

5. 테스트

구현 및 결과 이후 테스트를 통해 시스템의 구현 정도를 검증하였다. 테스트는 개발 테스트, 릴리즈 테스트를 통하여 진행하였고, 기능 별로 나누어 유닛 테스트, 시스템 테스트를 수행하였다. 테스트는 기기 간 통신과 센서의 동작에 초점을 맞추었다.

테스트 이름	아두이노 서버 연결
테스트 라벨	UNT_01
테스트 설명	아두이노의 센서 값이 실시간으로 서버와 동기화되는지 확인한다.
테스트 방법	아두이노의 센서 값을 바꾸어 서버에 저장된, 변화하는 값과 동일한가를 확인한다.
테스트 결과	49/50 (98%)

표 5 아두이노 서버 연결 테스트

테스트 이름	타이젠 서버 연결
테스트 라벨	UNT_02
테스트 설명	타이젠 어플리케이션에서 요청한 정보를 서버에서 받아올 수 있는지 확인한다.
테스트 방법	타이젠 어플리케이션에서 카페 정보나 좌석의 현황을 요청하는 경우, 서버에서 제대로 받아오는지 확인한다.
테스트 결과	50/50 (100%)

표 6 타이젠 서버 연결 테스트

테스트 이름	라즈베리파이 서버 연결
테스트 라벨	UNT_03
테스트 설명	라즈베리파이에서 요청한 정보를 서버에서 받아올 수 있는지 확인한다.
테스트 방법	라즈베리파이에서 카페 정보나 좌석의 현황을 요청하는 경우 서버에서 제대로 받아오는지 확인한다.
테스트 결과	50/50 (100%)

표 7 라즈베리파이 서버 연결 테스트

테스트 이름	전체 시스템 연결
테스트 라벨	SYST_01
테스트 설명	아두이노에서 인식 및 감지한 값을 실시간으로 타이젠 어플리케이션, 라즈베리파이, 서버에서 즉시 동기화되는지 확인한다.
테스트 방법	아두이노의 센서 값을 유지하거나 변화시켜서 타이젠 어플리케이션,

	라즈베리파이, 서버의 값을 확인한다.
테스트 결과	48/50 (96%)

표 8 전체 시스템 연결 테스트

표 5는 아두이노와 서버 간 연결로, 아두이노 기기가 좌석의 사용 여부를 판단하여 즉시 재시도 없이 서버로 값을 전송하는지 검증하였다. 표 6은 이와 마찬가지로 서버와 타이젠 어플리케이션 통신을, 표 7은 서버와 라즈베리파이 통신을 검증하였다. 이를 통하여 여러 차례 유닛 테스트 시행하여 작업을 완료하였고, 표 8을 통해 전체적인 시스템 테스트를 진행하였다. 이 외에도 총 10가지 테스트를 통해 본 프로젝트의 시스템은 100%에 근사한 테스트 결과를 보였고, 안정적이고 높은 신뢰성을 보장하는 시스템을 구현하였다고 판단할 수 있었다.

6. 결론

본 프로젝트는 처음부터 체계적인 설계, 개발 및 구현, 그리고 테스트 등의 소프트웨어 공학적 설계를 통해 완료한 프로젝트이다. 기존의 대학교 학부 과정의 프로젝트 수행과 달리, 소프트웨어 공학적 관점에서 철저하게 분석하고, 업무 및 일정을 분할하였으며, 개발과 구현까지 완료하였다. 소프트웨어 공학은 허용된 시간, 타당한 비용 범위 내에서 올바르고 효과적인 융통성 있는 보수가 쉬운 프로그램을 작성하기 위한 지침으로, 본 프로젝트에서는 이를 기반으로 진행하였다.

이후, 실제적인 비즈니스적 측면에서 실제 카페의 허가를 받아 직접 도입해 볼 수 있을 것이다. 먼저, 손님이 적은 지역의 작은 카페들을 우선적으로 접촉하여 카페의 실내 구조, 테이블 및 좌석의 위치와 개수 등을 파악하고 그 외 음료에 대한 정보 및 이벤트 등을 입력하여 시스템을 설치한다. 해당 카페에 찾아오는 고객들에게는 타이젠 어플리케이션을 설치하여 사용하도록 권장한 후, 시범적으로 테스트를 시행해볼 수 있다. 이를 통하여 사업 계획 테스트 모델인 '유동인구와 손님이 적은 지역의 작은 카페' 내에서 해당 시스템을 도입한 일부 카페가 상대적으로 매출이 오르거나 긍정적인 효과를 보았다면, 점차 확장해나갈 수 있는 가능성이 높아지게 된다. 또한, 시스템에서 발생할 수 있는 오류를 점차 수정하고 보완해나가면서 시스템은 더욱 완전해지고 점진적으로 유동인구가 많은 지역에 도입할 수 있을 정도의 시스템으로 발전할 것으로 보인다. 최종적으로는 카페 시장에서 유행을 선도하여 앞으로 크게 성장할 수 있을 것으로 보인다.

기대효과로 카페 좌석 관리 시스템은 좌석이 있는, 좌석의 관리가 필요한 모든 곳에 이용될 수 있다. 이 시스템은 좌석에 부착한 센서를 통해 현재 매장 내 손님의 착석 유무를 판단하고 실시간으로 동기화하여 다른 예상 고객들에게 시각적으로 바로 알려주는 데에 의미를 가진다. 따라서 카페뿐만 아니라, 영화관, 도서관, 음식점, 강의실, 교실 등 수많은 곳에서 긍정적인 효과를 예상할 수 있다. 즉, 본 프로젝트는 고객들이 기다릴 필요 없이 원하는 좌석에 앉게 도와주는 것, 매장의 좌석이 낭비되는 것을 막는 것, 매장과 고객 모두에게 시간과 공간이라는 자원을 효율적으로 사용하도록 돕는 것 등을 기대효과로 가질 수 있다. 또한, 이 시스템이 활성화된다면 사람이 존재하는 모든 장소에 설치하여 개개인의 시간과 공간, 에너지를 크게 절약할 수 있을 것이며, 경제적, 사회적으로도 막대한 효용을 가져올 수 있을 것이다.

7. Acknowledgement

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 실전적 SW 교육(SW 중심대학)지원사업의 연구결과로 수행되었음(R7115-15-1005)

참고문헌

[1] Ian Sommerville. "Software Engineering". Pearson, 2009

[2] 강민성 외 4인. "사용자 중심의 효율적인 좌석 자원 관리 시스템 설계 및 구현". 한국정보통신학회, 2007.

[3] 박용배 외 4 인. "사용자 맥락에 따른 좌석 추천 시스템 제안". 한국 HCI 학회, 2015.

[4] 신동성 외 6 인. "NFC 를 이용한 카페 웹 어플리케이션 설계". 한국정보과학회, 2014.

[5] 김용준 외 3 인. "POS 시스템 기반의 매장 좌석정보 시스템". 한국정보과학회, 2015.

[6] 김지표. "대학 도서관 열람실의 효율적 운영방안". 대한산업공학회, 2006.

블록체인 기반 서비스의 트랜잭션 검증 효율성 개선 방안

고동휘, 이우승, 조수환, 고덕윤, 이일로, 박수용 [2], 최수진 [1]

서강대학교 서강미래기술연구원[1]
서울 마포구 신수동

서강대학교 컴퓨터공학과[2]
서울 마포구 신수동

요약: 2009년 최초의 블록체인에 기반을 둔 애플리케이션인 비트코인이 개발된 이후 블록체인은 비트코인과 같은 전자 화폐 시스템뿐 아니라 클라우드 저장소 서비스, 블록체인 컴퓨팅 서비스 등 다양한 분야에 적용되었다. 그러나 블록체인 기반의 서비스들은 트랜잭션의 정확성을 확보하기 위해 모든 참여한 모든 노드가 검사를 수행해야 한다. 이는 동일한 검사를 모든 노트에서 실행하므로, 컴퓨팅 자원 사용 측면에서 매우 비효율적이다. 본 논문에서는 이 문제를 해결하기 위하여 블록 해쉬기반의 무작위 알고리즘을 통하여 무작위 노드에 블록검증권한을 줌으로써 효율성을 높이는 방안을 소개한다.

핵심어: 블록체인, 해시, 분산합의, 분산 블록체인

1. 연구배경

비트코인은 정부나 은행과 같은 중앙화된 조직이 아닌 분산 네트워크 상에서 발행, 저장, 유통되는 전자 화폐이다. 비트코인은 2009 년 처음 발행되어 현재 시점까지 중앙화된 기관 없이 화폐의 가치를 유지하고 있으며, 사용자 수와 거래 규모 측면에서 급격하게 증가하고 있다. 이러한 비트코인의 화폐 가치를 유지하는 배경에는 블록체인 기술이 있다.

블록체인은 중앙화된 서버 없이 탈중앙화된 네트워크 환경에서 보안성과 무결성을 유지하는 기술이다. 이러한 강점에 기반하여 다양한 컴퓨팅 데이터를 공유할 수 있게 되었고, 전자 화폐 이외의 다른 시스템에도 적용되기 시작하였다. StorJ[2]는 블록체인을 이용한 클라우드 스토리지 시스템으로 스토리지 제공자에게 자체 코인을 발급하여 다른 컴퓨터의 디스크 공간을 이용하고 분할 검증을 통하여 신뢰도를 유지하는 서비스를 제공한다. 이더리움[3]은 전자화폐를 스마트 계약(smart contract)을 수행하는 어플리케이션을 등록하여 실행할 수 있는 플랫폼이다. 비트코인은 단순히 전자 화폐의 송금을 제공하는데 반해 이더리움은 계약에 의한 거래를 제공해 준다.

그러나 블록체인은 컴퓨팅 자원 낭비가 심한 비효율적인 운영 구조를 갖고 있다. 비트코인의 경우 거래 트랜잭션의 무결성을 검증하기 위해 최대 한 시간이 소요되며, 각 참여 노드는 30Gbyte 크기의 장부를 저장해야 한다. 또한 거래를 승인 하기 위하여서 모든 노드가 장부를 통해 트랜잭션을 검증하고, 검증 결과를 기반으로 거래를 승인해야 한다. 즉 블록체인에 참여한 모든 노드는 동일한 장부를 모두 저장하고 있어야 하고, 동일한 검증 작업을 모두 수행해야 한다. 이러한 사유로 인해 블록체인은 전체 네트워크 관점에서 자원 효율성이 매우 떨어진다.

본 논문에서 거래 트랜잭션 검증을 위해 모든 노드 들이 같은 작업을 수행하는 비효율성을 해결하기 위해 여러 거래 내역을 분산하여 검증하는 기법을 제안한다. 그러나 거래 트랜잭션 검증이 분산되면 필연적으로 거래의 무결성 보장이 어려워 진다. 왜냐하면, 검증하는 노드 수가 줄어들면, 데이터 조작을 위한 51%의 공유가 상대적으로 쉬워지기 때문이다. 본 논문에서는 이러한 문제를 해결하기 위해 거래 트랜잭션 검증을 예측할 수 없는 노드에 할당하여 자원 효율성을 높임과 동시에 무결성을 유지하기 위한 기법을 소개한다.

본 논문은 아래와 같이 구성된다. 2장에서는 배경 지식으로 블록체인에 대해 기술하고, 기존의 블록체인에 적용된 거래 트랜잭션 검증 합의 알고리즘들을 살펴본다. 3장에서는 본 논문에서 제안하는 해시 값을 이용한 분산 블록체인 검증 방안을 소개하고 4장은 검증. 5장에서 결론을 기술한다.

2. 배경 및 기존 연구

2.1 블록체인

이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 정보통신 기술진흥센터(No.10044457, 자율지능형 지식/기기 협업 프레임워크 기술개발)와 2015 년도 정부(미래창조과학부)의 재원으로 한국 연구재단-차세대정보·컴퓨팅기술개발사업 (No. 2012M3C4A70333 48)의 지원을 받아 수행된 연구임

앞에서도 언급한 바와 같이, 블록체인이란 비트코인에서 탈 중앙화된 화폐관리를 위해 사용된 핵심기술이다. 블록체인은 중앙 관리 기관이 없이 임의의 사용자를 통해 데이터 위변조가 불가능 하게 하여 거래의 무결성을 유지한다. 블록체인은 무결성 유지를 위해 블록체인 네트워크에 참여한 모든 참여 노드는 거래 내역(장부)를 저장하고, 새롭게 생성되는 거래 트랜잭션을 저장된 장부를 통해 검사한다. 예를 들어, 한 참여 노드가 거래를 조작하여 이중 결재를 감행하여도, 다른 모든 노드가 이중 결재 여부를 검증하여 합의를 부결함으로써, 거래의 승인이 거부된다. 따라서 전체 노드 중 51% 이상이 거짓으로 합의하여 주지 않는 이상, 데이터의 조작은 발생하지 않게 된다.

그러나 블록체인은 컴퓨팅 자원 사용 측면에서 매우 비효율적이다[8]. 앞에서도 언급한 바와 같이 블록체인 네트워크에 참여한 모든 노드는 한 거래 트랜잭션을 검증하기 위해 동일한 검증작업을 수행한다. 이러한 문제점은 특히 트랜잭션 검증을 위해 복잡한 알고리즘을 수행해야 하는 이더리움의 경우 더욱 부각될 수 있다. 예를 들면, 이더리움 네트워크 상에서 대출 계약을 수행하는 경우, 계약자의 거래 내역과 신용상태를 검증하고 이를 승인하는 검증 알고리즘을 수행하여야 한다. 이 때 모든 노드에서 검증 알고리즘을 수행하는 기존의 블록체인 합의 방식은 매우 비효율적이기 때문에 개선이 필요하다.

2.2 암호화 해시 함수

암호화 해시 함수(Cryptographic hash function)는 해시 함수의 일종으로 임의의 입력 메시지를 고정된 길이의 문자열을 출력 하는 압축 함수이다. 이 때 출력 값은 입력 값에 의존하는데 즉 입력 값의 데이터가 변하면, 출력 값 또한 변경되어, 주로 데이터의 무결성 검증, 메시지 인증에 사용한다. 해시 함수는 일방향성과 충돌 회피성이라는 2가지 성질을 만족해야 한다. 먼저, 일방향성은 해시함수 H에서 주어진 출력 해시 값 h를 통해 H(x)=h를 만족하는 x 값을 찾는 것이 계산적으로 불가능한 것을 말한다. 즉 해시 함수 출력 값을 통해 원본 데이터를 추출할 수 없어야 한다. 강한 충돌 회피성이란 H(x)=H(y)를 만족하는 경우가 발생하지 않아야 함을 의미한다. 즉, 모든 입력 값에 대해 해시 함수의 입력 값이 다르다면 출력 값은 달라야 함을 의미한다.

2.3 관련연구

블록체인의 컴퓨팅 자원 효율성을 개선하기 위한 몇몇의 연구가 발표된 바 있다. 대다수의 효율성 개선 연구는 블록체인에 일반적으로 적용된 작업 검증(Proof of Work, 이하 PoW)을 개선하는데 초점을 두고 있다. 블록체인의 작업검증은 거래 트랜잭션의 집합인 블록을 생성하기 위한 자격을 획득하기 위해 특정 문제의 답을 찾는 과정을 의미한다. 이 때 많은 전력과 컴퓨팅 자원이 소모되기 때문에, 이러한 방식을 개선하는 것은 자원 효율성을 높이는데 매우 중요하다.

2.3.1 대리 지분 검증 (DPoS : Delegated Proof of Stake)

대리 지분 검증은 기존 블록체인의 PoW 에 약 한 시간의 지연 시간이 발생하는 문제를 해결하기 위해 나온 알고리즘으로 블록체인 어플리케이션 중 하나인 BitShares[5]라는 서비스에서 사용하였다. 대리 지분 검증(이하 DPoS)은 지분 보유자들이 투표를 하여 블록 검증 대리자를 선출하는 방식이다. 대리자들은 전체 트랜잭션을 검증할 수 있는 권한을 받게 된다.

이 연구는 전체 자원 사용의 효율성은 개선하였으나, 대리자 노드가 모든 트랜잭션을 검증하여야 하므로, 한 노드당 처리해야 하는 트랜잭션의 양은 기존의 기법과 동일하다. 그리고, 선출된 대리자의 지속기간이 길수록 담합의 위험도 높아진다.

2.3.2 텐더민트(Tendermint) 기법

텐더민트 기법[6]은 각 노드가 보유한 화폐 총량에 따라 지분을 받고, 이에 따라 의사결정권을 갖게 하는 기법이다. 이 때, 한 노드가 거짓으로 검증을 수행한 경우, 해당 노드의 모든 지분의 권리를 박탈함으로써 무결성을 보장한다. 이 기법은 비잔티움 장애 허용(Byzantine fault tolerance)을 탈 중앙화된 분산환경에서 처리할 수 있도록 개선하였다. 이 기법은 기존의 블록체인 어플리케이션에서 사용하는 PoW 과정을 없애고, 지분에 기반한 합의를 통해 전체 컴퓨팅 자원 효율성을 개선하였다.

이 기법은 DPoS 와 마찬가지로 특정 단일 노드에 할당되는 트랜잭션의 수는 동일하다는 문제점 뿐 아니라, 지분이 높은 일부 노드가 담합 하는 경우 데이터의 무결성이 보장되지 않을 가능성이 있다. 이 기법은 데이터의 위변조를 사전에 방지하는 정책이 아닌 위변조 행위에 대한 징벌적 처리를 통해 무결성을 유지하기 때문이다.

3. 효율적 검증을 위한 블록체인 합의 알고리즘 개선방안

앞에서 언급한 바와 같이 기존의 블록체인은 거래 트랜잭션의 검증을 위하여 모든 노드 들이 검증에 참여한다. 이러한 기법은 전체 블록체인 네트워크의 컴퓨팅 자원 효율성을 현저하게 떨어뜨리는 주요 원

인이 된다. 본 논문에서는 이를 해결하고자 새로운 검증 방안을 제안한다.

컴퓨팅 자원 효율성을 개선하기 위해 전체 트랜잭션을 분할하여 처리한다. 그림 1 은 분산 트랜잭션 할당을 개괄적으로 보여준다. 먼저 특정 블록 분할 알고리즘(예: 공개키 범위 기반 분할)에 의해 트랜잭션을 분할하여 블록을 생성한다. 그리고 각 블록의 검증을 담당할 노드를 할당하고, 해당 노드는 할당된 블록의 트랜잭션의 검증을 수행한다.

그러나 노드들이 트랜잭션을 분할 검증할 경우 소수의 악성 노드들에 의해 위변조가 가능해진다. 예를 들어 한 블록을 전체 노드의 1/10 개의 노드가 검증하는 경우 전체의 6%만 담합하여도 데이터의 위변조가 가능해진다. 그림 2 는 이러한 상황을 보여준다. 전체 트랜잭션 수가 40 개인 경우 네 개의 노드가 10 개씩의 트랜잭션을 검증하게 된다. 이 때 31~40 번 트랜잭션을 위변조를 위해서는 이를 담당하는 9 개의 노드 중 다섯 개 이상만 합의하면 가능해진다.

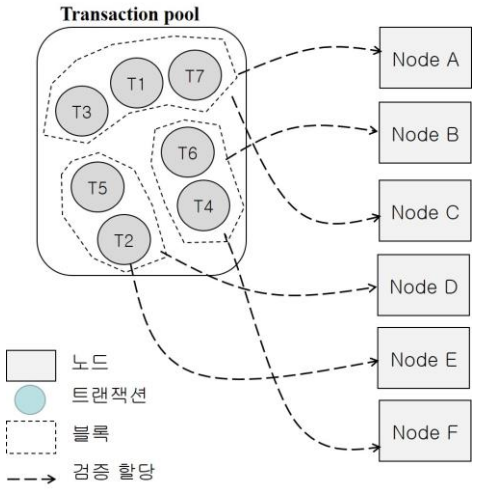


그림 1 분산 트랜잭션 할당

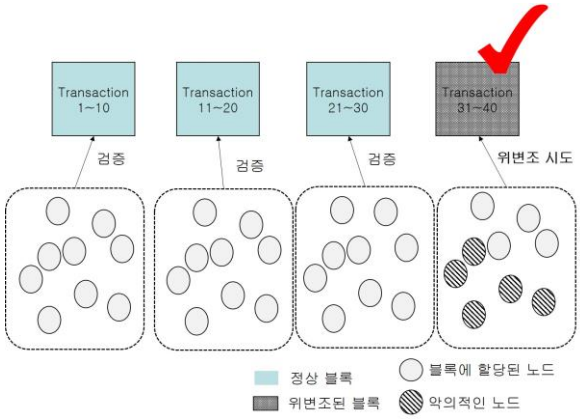


그림 2 분산 블록체인에서의 위변조 시도

일반적인 블록체인 어플리케이션들은 블록을 생성 할 때 이전 블록의 해시 값을 사용하여 생성해야 하므로, 이전 블록이 완성되기 전에 다음 블록이 생성될 수 없다. 이를 통해 블록의 실시간성을 제공한다. 본 연구에서는 예측 불가능한 이전 블록의 해시를 이용해서 무작위 값을 추출하여 이 값에 따라 노드 들에게 권한을 부여한다. 무작위 값을 통해 노드에 블록을 할당하는 규칙은 알고리즘 1 과 같다.

```
Input : prevHash (이전 블록 해시 값)
Output : 할당 블록 여부

1: procedure isAssignedBlock(prevHash)
2:   r1 = rand(prevHash + nodeId);
3:   r2 = SHA256(rand(r1));
4:   r3 = SHA256(nodeId);0
5:   if(r2.substr(38,1) == r3.substr(38,1))
6:     return true;
7:   return false;
```

알고리즘 1. 할당 블록 확인 슈도코드

먼저 이전 노드의 해시값과 자신의 노드 아이디를 기반으로 임의의 수를 구하고 이를 해시함수로 해시값을 구한다(2~3라인). 이 알고리즘은 해시함수로 SHA256 알고리즘을 사용한다. 그리고 노드 아이디를 기반으로 해시 값을 구한 후(4라인) 해당 해시 값의 임의의 위치의 문자 값이(38번째 문자) 같으면(5라인) 이 블록(prevHash)의 다음 블록은 해당 노드의 검증 대상이 된다(6라인).

이전 블록의 해시 값을 이용하여 블록을 할당하기 때문에, 할당이 실시간으로 발생하므로, 실시간성이 확보되고, 데이터의 위변조를 위한 담합에 위험성이 적어진다. 이러한 방법을 통해 적은 노드가 트랜잭션을 검증하여도 무결성을 보장할 수 있다.

4. 검증

4.1 실험 설계

본 연구의 유효성을 검증하기 1000 개의 비트코인 지갑과 50 개의 블록에서 약 2,500 개의 트랜잭션 데이터를 수집하였다. 1000 개의 비트코인 지갑이 각각의 노드라고 가정하고, 이 중 20%가 악의적인 노드로 선정한다. 알고리즘 1 을 수행하여 50 개의 블록을 50 개에서 1000 개의 노드에 할당하였다. 여기서 임의의 수를 추출하기 위한 알고리즘은 매르세네 트위스터(Mersenne Twister)[7] 난수 생성 알고리즘을 채택하였다. 알고리즘 1 에 의해 한 노드는 여러 개의 블록을 검증할 수 있고, 한 블록은 여러 노드에서 검증 받을 수 있다.

그림 3 은 제안한 기법의 전체 노드의 트랜잭션 검증 횟수를 보여준다. 기존의 블록체인은 모든 노드에

서 전체의 트랜잭션을 처리하는 데에 반해, 본 논문의 기법은 분산하여 처리하므로, 전체 처리량이 현저하게 줄어든다. 본 실험 에서는 모든 노드 수에서 평균 검증횟수가 약 (6)%로 줄어들었다.

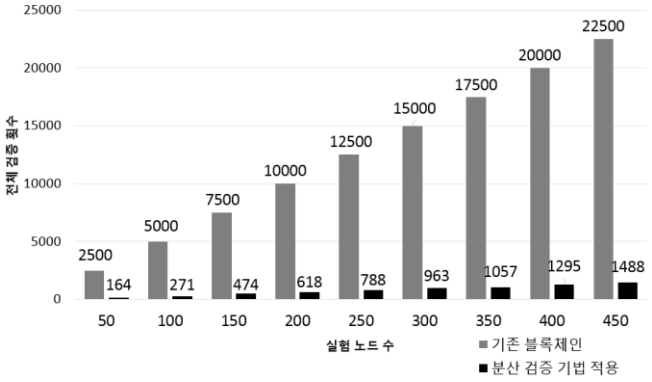


그림 3 기존 블록체인과의 반복 횟수 비교

그림 4는 전체 실험 노드 중 20%가 악의적으로 데이터 위변조를 시도하는 노드로 가정하고, 전체 50개의 노드 중 위변조가 가능한 블록의 수를 측정하였다. 위변조가 가능한 블록은 해당 블록을 검증하는 노드 중 악의적인 노드의 수가 50%가 넘으면, 위변조가 가능한 것으로 식별하였다. 실험 결과 최소 300개의 노드가 확보된다면, 위변조가 불가능한 것으로 판단되었다.

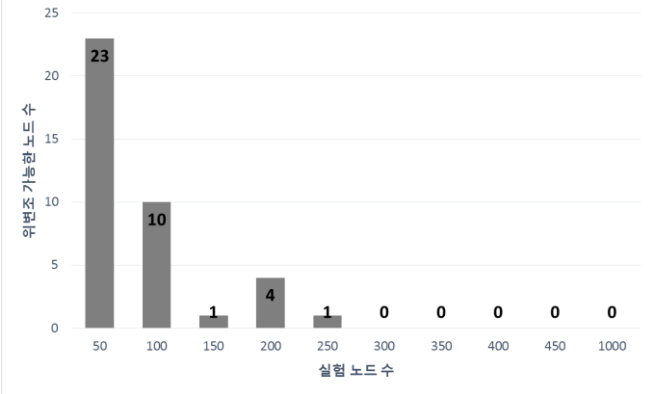


그림 4 노드수에 따른 조작 성공횟수

5. 결론

본 논문에서는 블록체인 네트워크에서 전체 검증의 횟수를 줄여서 컴퓨팅 자원의 효율성을 높이는 검증 기법을 제안하였다. 즉 검증 대상의 트랜잭션을 분할하고 이를 임의의 노드에 할당함으로써, 기존 블록체인의 검증 수 대비 94%가 개선되었다. 그리고 300 개 이상의 노드가 확보되는 경우, 위변조의 가능성이 매우 희박해짐을 알 수 있었다.

본 논문에서 제안하는 기법은 트랜잭션을 수집하여 블록을 생성할 때 트랜잭션간의 의존성이 있으면, 정확한 검증이 힘들다는 한계점이 있다. 따라서 향후 연구로는 본 연구를 뒷받침 할 수 있는 전체 트랜잭션에서 각 트랜잭션 간의 의존성을 측정하고 이를 통해 블록을 분할하는 기법을 연구하고자 한다.

참고문헌

[1] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." *Consulted* 1.2012 (2008): 28.

[2] Shawn Wilkinson , “Storj A Peer-to-Peer Cloud Storage Network” http://storj.io/storj.pdf

[3] Vitalik Buterin, “A Next-Generation Smart Contract and Decentralized Application Platform” https://github.com/ethereum/wiki/wiki/White-Paper

[4] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance.", Springer Berlin Heidelberg, 2004.

[5] “BitShares Delegated Proof of Stake”, https://bitshares.org/technology/delegated-proof-of-stake-consensus/

[6] Jae Kwon, “Tendermint: Consensus without Mining” http://tendermint.com/docs/tendermint.pdf

[7] M. Matsumoto & T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator", ACM Trans. Model. Comput. Simul. 8, 3 (1998).

[8] J. Barkatullah and T. Hanke. "Goldstrike 1: CoinTerra's First Generation Crypto-currency Mining Processor for Bitcoin." (2015).

사물 인터넷 환경에서의 그룹 사용자를 위한 그룹 구성 정보 기반 서비스 추천 방법

이진서, 고인영

한국과학기술원 전산학부
대전광역시 유성구 대학로 291
{jinseo.lee, iko}@kaist.ac.kr

요약: 다양한 사물 인터넷 기기가 등장으로 인해 이를 활용한 다양한 서비스를 제공할 수 있게 되었고 이러한 서비스의 대상은 개인 사용자뿐만 아니라 그룹 사용자를 포함하고 있다. 본 연구는 사물 인터넷 환경에서 그룹의 구성 정보를 기반으로 사용자 기반 협업 필터링을 사용하여 서비스 사용 기록이 없는 새로운 사용자에게도 서비스를 추천할 수 있는 방법을 개발하고 실제 사물 인터넷 테스트 베드 환경에서 수집된 데이터를 통해 추천 성능을 확인하였다.

핵심어: 추천 시스템, 그룹 추천, 서비스 추천, 사물 인터넷

1. 서 론

최근 다양해진 사물 인터넷(Internet of Things, IoT) 기기들의 등장으로 인해 해당 기기들을 조합, 활용하여 다양한 서비스를 제공할 수 있게 되었다. 나아가 휴게실과 같은 공공 장소에도 다양한 사물 인터넷 기기가 설치될 것이며 서비스의 대상은 개인뿐만 아니라 그룹 사용자를 포함하게 될 것이다. 다양한 서비스를 매번 선택하는 것은 사용자에게 번거로운 일이 될 수 있으며 서비스 추천 시스템(Service Recommender System) 은 이러한 문제를 효과적으로 해결할 수 있는 방법 중의 하나이다.

본 논문에서는 사물 인터넷 환경의 추천 시스템을 구축하는 데 있어 서비스 수행 기록이 없는 새로운 그룹 사용자가 사물 인터넷 환경의 장소에 입장할 때 서비스를 추천하는 상황을 가정한다.

기존의 그룹 추천 시스템에 대한 대다수의 연구에서는 그룹 구성원들의 항목에 대한 개별적인 추천 결과를 종합하거나 개별적인 선호도 정보를 통합하는 방식을 사용한다[1]. 이는 개인으로서의 항목에 대한 선호도가 그룹으로서의 항목에 대한 선호도와 동일한 양상을 보일 것이라는 암묵적인 가정에 기반한다. 즉, 위와 같은 상황에서 기존 그룹 추천 시스템 방식과 같이 각 구성원들이 개별적으로 수행한 서비스 수행 기록을 통합하여 서비스를 추천하는 방법을 생각해 볼 수 있다. 하지만 사물 인터넷 환경과 같이 개인으로서 선호하는 서비스와 그룹으로서 선호하는 서비스가 다른 경우가 빈번한 환경에서 이러한 방식은 적절치 않다.

본 논문에서는 이러한 문제를 해결하고자 그룹 구성 정보가 유사한 그룹 사용자들이 수행한 서비스 수행 기록을 통합하여 서비스를 추천하는 그룹 유사도 기반 협업 필터링 기법을 제안한다. 또한 실제 구축된 사물인터넷 테스트 베드로부터 수집된 데이터를 통해 기존의 그룹 추천 시스템과의 추천의 정확도를 비교 평가한다.

3. 그룹 유사도를 사용한 사용자 기반 협업 필터링 기법

사용자 기반 협업 필터링은 일반적으로 사용자들의 항목에 대한 평점을 바탕으로 유사한 양상을 보이는 사용자를 이웃 사용자로 선택하여 이웃 사용자들의 평점을 종합하여 사용하지 않은 항목의 평점을 예측하고 이를 바탕으로 항목들을 추천한다[2].

이를 위해 가장 먼저 그룹들의 서비스 사용 기록에 기반하여 그룹 사용자×서비스 행렬을 생성한다. 행렬 내 값을 평점이라 부르며 해당 서비스에 대한 해당 그룹 사용자의 서비스 수행 횟수의 상용로그 값을 사용한다. 이렇게 정규화를 하는 이유는 일반적인 평점과 같이 범위를 한정시키기 위함이다.

다음으로 이웃에 해당하는 유사 그룹을 선택한다. 하지만 새로운 그룹 사용자는 어떠한 서비스에 대해서도 수행 기록이 존재하지 않으므로 평점을 바탕으로 그룹간 유사도를 측정할 수 없다. 따라서 본 논문에서는 그룹간 구성 정보가 그룹 유사도를 측정하는 데 사용할 수 있을 것이라 가정 하에 전체 그룹 구성원 수와 공통된 그룹 구성원 수를 사용하였다. 위의 두 기준은 별도의 추론 과정 없이 획득 가능하여 객관성을 보장한다. 첫 번째로, 전체 그룹 구성원의 수를 기반으로 하는 유사도 측정은 아래와 같은 식을 사용하며, 두 그룹의 전체 구성원 수가 비슷하면 수행하는 서비스가 유사할 것이라는 가정에 기반한다.

$$sim_{memberNum}(G_i, G_j) = \frac{\min(|G_i|,|G_j|)}{\max(|G_i|,|G_j|)}$$

공통된 그룹 구성원의 수를 기반으로 하는 유사도는 아래 식과 같이 Jaccard 유사도를 사용하며, 두 그룹에 공통적으로 포함된 구성원들이 존재 할 경우 해당 그룹 구성원들은 각각의 그룹이 수행하는 서비스에 유사한 영향을 미칠 것이라는 가정에 기반한다.

$$sim_{commonMem}(G_i, G_j) = \frac{|G_i \cap G_j|}{|G_i \cup G_j|}$$

각각의 유사도를 그룹 유사도로 사용하여 특정 임계치 이상의 그룹 유사도를 가지는 그룹들을 이웃 그룹으로 선택한다. 선택된 이웃 그룹의 서비스 수행 기록을 가중치 합 방식으로 통합하여 새로운 그룹 사용자의 각 서비스에 대한 평점을 예측한다.

$$\hat{r}_{g,s} = \frac{\sum_{g' \in G}(sim(g,g') \times r_{g',s})}{\sum_{g' \in G} sim(g,g')}$$

마지막으로 예측된 평점을 바탕으로 평점이 높은 순서로 서비스 N 개를 추천한다.

4. 실험 및 결과

본 실험에서 사용된 실험 데이터셋은 KAIST 김병호-김삼열 IT 융합 빌딩(N1) 8 층 세미나실과 휴게실에 설치된 미래 인터넷 테스트 베드에서 수집하였다. 실험에서는 10-Fold 교차 검증법(10-Fold Cross Validation)을 사용하여 추천 정확도를 비교하였다. 베이스라인(Baseline)으로는 가장 일반적인 그룹 추천 시스템의 평균적 추천 전략을 사용하였다.

우리의 접근법은 전체 그룹 구성원 수, 공통된 그룹 구성원 수 각각을 유사도로 사용하는 방식과 두 가지 유사도를 동시에 사용하는 방식을 사용하여 실험하였다. 새로운 사용자의 서비스에 대한 평점을 예측한 후 평점을 기준으로 Top-1 의 서비스의 추천 정확도를 측정하였다.

그림 1. 세미나실 내 Top-1 서비스 추천 정확도 비교

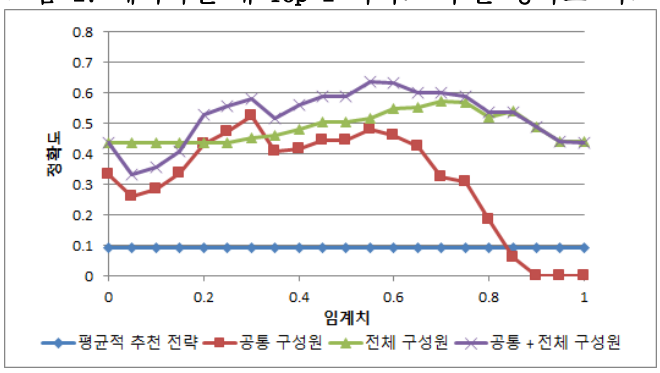


그림 2. 휴게실 내 Top-1 서비스 추천 정확도 비교

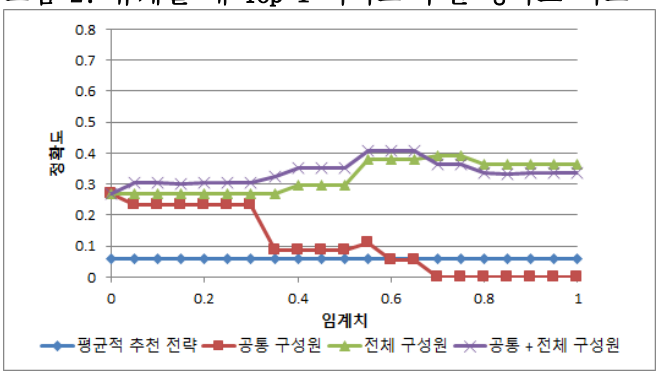


그림 1 과 그림 2 의 결과를 보면 전체적으로 그룹 유사도 기반의 협업 필터링 방식의 추천 정확도가 베이스 라인에 비해 훨씬 더 높음을 알 수 있다. 또한 공통된 그룹 구성원 수를 기반으로 하는 유사도는 임계치를 적절히 선택하는 것이 특히 중요하며 장소에 따라 적정 임계치가 다름을 확인할 수 있다. 마지막으로 두 가지의 그룹 구성원 정보를 모두 활용한 방식이 장소의 특성에 영향을 적게 받으면서 안정적인 성능을 보임을 확인할 수 있었다.

5. 결 론

본 논문에서는 사물 인터넷 환경에서 그룹 구성 정보를 사용하는 그룹 유사도 기반의 이웃 기반 협업 필터링을 통해 서비스 사용 기록이 없는 그룹 사용자에게 서비스를 추천하는 방법을 개발하였다. 또한 실제 데이터로 실험한 결과, 사물 인터넷 환경에서는 기존의 그룹 추천 시스템 방식이 적절치 않으며 본 논문에서 제안한 방식이 훨씬 높은 추천 정확도를 보임을 확인하였다.

Acknowledgement

본 연구는 미래창조과학부 및 정보통신기술연구진흥센터의 방송통신·산업기술 개발사업의 일환으로 수행하였음. [B0101-15-0334, IoT 기반 스마트 홈 커뮤니티에서 안전하고 행복한 삶을 위한 소셜 매칭 및 소통 서비스 기술 개발]

참고문헌

[1] Berkovsky, Shlomo, and Jill Freyne. "Group-based recipe recommendations: analysis of data aggregation strategies." Proceedings of the fourth ACM conference on Recommender systems. ACM, 2010.

[2] Schafer, J. Ben, et al. "Collaborative filtering recommender systems." The adaptive web. Springer Berlin Heidelberg, 2007. 291-324.

미션-크리티컬 인메모리 DBMS 의 결함탐지를 위한 결함주입 테스트 방법 및 도구

서광익, 마영철, 이재효, 이종정, 박준호
알티베이스
e-mail : {kwangik.seo, youngchul.ma, jaehoy.lee, jongjung.lee, joonhoo.park}@altibase.com

Fault Injection Test Method and Tool for Fault Detection of In-Memory DBMS as Mission-critical System

Seo Kwangik, Ma youngchul, Lee Jeahoy, Lee Jongjung, Joonho Park.
Altibase

요 약

최근 DBMS 는 단순히 데이터베이스 관리의 편의성과 효율성을 제공해주는 목적을 넘어, 사용 목적과 용도가 다양해 지면서 구현하는 기술 또한 고도로 복잡해지므로 보다 철저한 검증이 필요한 상황이다. 본 논문에서는 인메모리 DBMS 에 결함을 주입하고 그 결과를 확인할 수 있는 결함주입 테스트 도구 구현을 통하여 효율적으로 오류를 검출하고 예외사항 처리 및 에러 메시지 출력등을 통하여 실제 시스템 설계 의도대로 처리되는지를 검증하였다.

1. 서론

미션-크리티컬 시스템이란 오동작이 목표활동 및 비즈니스 수행에 직접적인 실패를 발생시킬 수 있는 시스템을 말한다. 예를 들면 우주선의 항로 결정에 영향을 주는 우주선 항해 시스템라든지, 통신사에서의 실시간 요금계산 실패로 인한 기업 이익 손실에 영향을 미칠 수 있는 실시간 과금 시스템 등이 이에 속한다. 최근 이러한 미션-크리티컬 시스템의 복잡도와 고도화로 인해 더욱 철저한 검증이 필요하게 되었고, 기능 요구사항뿐 아니라 신뢰성 또는 안정성, 고가용성과 같은 비기능 요구사항을 검증할 수 있는 방법이 필요하게 되었다. 하지만 이러한 비기능 테스트의 테스트 방법은 오류를 발생하기 위한 테스트 데이터 또는 테스트 케이스를 수동으로 발생하거나, 발생된 오류의 재생이 매우 어렵다. 특히 시스템의 서비스가 중지되거나 소스 파일 처리 결과에 대한 실패를 발생시키는 결함을 예측하거나, 계획된 시간에 발생시키기란 매우 어렵다. 따라서 본 논문은 미션-크리티컬 시스템의 한 종류인 인메모리 기반 DBMS 에 결함을 주입하고 결함을 시스템 설계 의도대로 처리하는지 검증하는 결함 주입 테스트 방법을 구현하고 검증한다.

2 장에서는 결함주입 테스트에 대한 개념적인 설명을 하고, 3 장에서는 결함주입 테스트 도구의 설계와 구현물, 그리고 검증 결과를 설명한다. 마지막으로 4 장에서는 본 고의 결론 및 향후 계획을 정리한다.

2. 결함주입 테스트

결함 주입 기법은 하드웨어 기반 결함 주입과 소프트웨어 기반 결함 주입으로 나뉜다. 그 중 본 논문에서는 소프트웨어 기반 결함 주입에 대해 설명한다. 소프트웨어 기반 결함 주입은 컴퓨터 프로그램 등에 발생 가능성이 있는 결함을 주입하는 것을 말하며, 결함 주입 테스트는 결함 주입을 통해 해당 결함에 대한 소프트웨어의 강건성(robustness) 또는 신뢰성을 검증하는 일련의 행위와 절차를 말한다.

이러한 결함 주입 테스트는 유입된 결함에 의해 시스템 장애가 발생하면, 장애를 분석하여 장애 발생이 실제 운영될 시스템에서 재발하지 않도록 시스템을 수정한다.

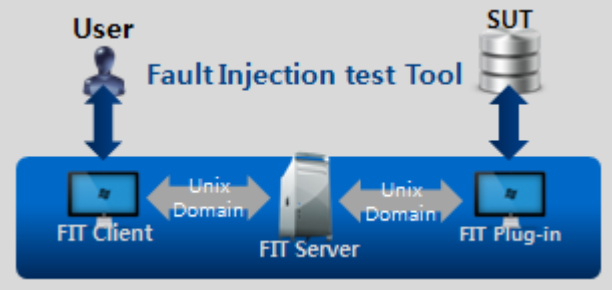
3. 결함주입 테스트 도구의 설계 및 구현

3.1 결함주입 시스템 식별

소프트웨어 결함주입 기법을 설계하기 위해서는 주입 대상 시스템에 따라 주입 방법이 달라 질 수 있으므로, 먼저 결함이 주입될 시스템을 정의해야 한다. 본 논문에서 다룰 테스트 대상은 비즈니스 및 주요 관리 시스템에 사용되는 미션-크리티컬 시스템의 한 종류인 인메모리 기반 DBMS 이다. 인메모리 DBMS 는 최근 초고속 빅데이터 처리를 위한 인메모리 컴퓨팅 분야에서 가장 주목 받고 있는 기술 중 하나이다. 이러한 인메모리 DBMS 는 메모리 기반 데이터 저장과 관리가 목적이므로 물리적인 시스템 메모리 기술과 이를 관리하는 OS 기술이 필수적이다. 하지만 시스템

레벨의 프로그램 및 관리 기술을 테스트하는 방법은 쉽지 않다. 따라서 본 논문에서는 알티베이스에서 오랜 기간 연구하고 개발해 온 빅데이터 처리용 인메모리 DBMS 를 대상으로 하여 결함주입 테스트 방법을 설계하고 도구를 구현한다. 또한 본 논문에서 결함주입 테스팅 도구는 FIT(Fault Injection testing Tool)로 지칭한다.

3.2 FIT 개념도

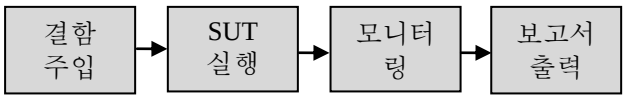


(그림 1) FIT 개념도

FIT 은 테스트 대상인 SUT(System Under Test)의 소스코드에 결함을 정의한다. SUT 가 수행되는 동안 결함이 삽입된 부분이 호출되면 관련 매크로가 수행되어 결함이 발생한다. FIT 서버는 SUT 를 모니터링하고 Client 는 모니터링 결과를 사용자에게 출력해준다.

3.3 결함 주입 절차 및 방법

결함을 주입하는 방법은 소스코드에 직접 주입하는 정적 주입 방식과 런타임에 주입하는 동적 주입 방식이 있다. 본 연구는 사전에 인메모리 DBMS 에서 자주 발생하는 결함 정의하고, 각 결함이 발생하는 위치와 유형을 소스코드에 직접 정의하는 정적 주입 방식을 사용한다.



(그림 2) 결함 주입 테스트 절차

3.3.1 결함주입

사전에 정의되어 있는 결함을 소스코드에 주입하는 단계로, 결함을 삽입할 위치를 찾아 결함이 발생할 문장 바로 위에 결함을 삽입한다. 삽입 후 빌드를 통해 SUT(System Under Test, 테스트 대상 시스템)가 가능하도록 한다.

3.3.2 SUT 실행

테스트 결과를 관리하고 모니터링 하기 위한 결함주입 도구 서버와 모니터링 된 결과를 사용자에게 출력해주는 클라이언트를 수행한다. 또한 결함이 주입된 SUT 를 구동하고 결함을 발생시킨다. 이때 구동될 결함의 정보는 클라이언트가 서버에 등록하여 서버가 결함을 관리하고 모니터링 할 수 있도록 한다.

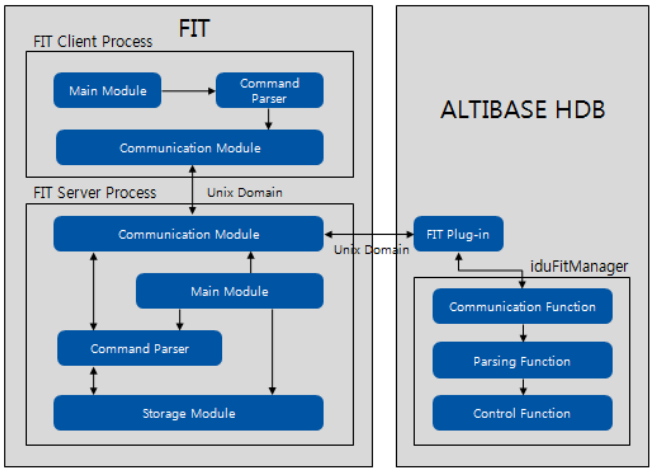
3.3.3 모니터링

모니터링은 결함관리 도구의 서버가 SUT 가 실행되는 동안 발생한 결함의 이름, 유형, 위치, 결과 등을 모니터링하고 파일에 저장한다.

3.3.4 보고서 출력

결함이 발생하는 동안 모니터링한 결과는 서버가 파일로 기록하고, 기록된 내용은 결함주입 도구의 클라이언트가 사용자에게 출력한다.

3.4 FIT 구조도



(그림 3) FIT 구조도

FIT 구조도는 결함을 포함하고 있는 SUT인 ALTIBASE HDB와 FIT Client/Server로 구성된다. 주요 컴포넌트에 대한 설명은 다음과 같다.

- FIT Client Process
  - FIT Client 프로세스는 주로 사용자가 입력한 결함 정보를 수집해서 UNIX Socket IPC 통신 방식으로 FIT Server에게 전송한다. 그리고 FIT Client 프로세스는 전송된 결함 정보의 처리 결과를 실시간으로 수신 받고 출력한다.
  - CP(Command Parser)는 입력한 정보를 FIT 서버에게 전송하기 전에 유효성에 대한 검증하는 모듈이다.
  - CM(Communication Module)은 IPC 방식으로 FIT 서버와 통신하는 모듈이다.

- FIT Server
  - FIT Server는 FIT 테스트의 핵심 컴포넌트이며, 주로 결함 정보를 관리하고, 알티베이스 DBMS의 결함 정보 조회 서비스를 제공하는 데몬 프로세스이다.
  - CM(Communication Module)은 프로세스 간의 통신을 담당하고 있으며, 내부 프로세스간 IPC가 가능한 내부 파일시스템을 사용하는 양방향 통신인 Domain Socket 통신 방식을 사용한다.
  - MM(Main Module)은 명령어의 기능 인터페이스를 제공하고 동시성을 제어한다. 또한 동작에 대한 로그를 기록한다.
  - CP(Command Parser)은 명령어를 파싱하고,

명령어에 따라 결함을 검사한다.
- SM(Storage Module)은 데이터를 저장하고 관리하는 기능을 제공한다.
- FIT Plug-in는 FIT 서버와 알티베이스 DBMS 사이에서 통신을 담당한다.

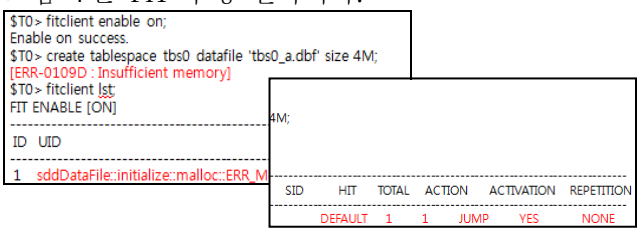
3.6 FIT 주요 기능

(1) Fault Management
다양한 종류의 결함에 대한 관리 기능을 제공한다.
FIT 는 다양한 종류의 결함에 대한 추가, 수정, 삭제, 조회 기능을 제공해주며, 고객이 이런 기능을 이용하여 결함을 관리할 수 있다.
(2) Failure Action
결함이 발생 시에 사용자가 지정된 행위(Action)에 따라 시스템 실패를 일으키는 기능을 제공한다.

Action Type	Default	Description	Example
hit	N/A	hit 액션을 통해서 사용자가 해당 Fault 가 발생했는지 알 수 있다.	-a hit
jump	N/A	jump 액션을 통해서 사용자가 대상 시스템에게 에러를 일으킬 수 있다.	-a jump
kill	N/A	kill 액션을 통해서 사용자가 대상 시스템을 비정상 종료 시킬 수 있다.	-a kill
sleep	TIMEOUT	sleep 액션을 통해서 대상 시스템이 주어진 밀리 초(millisecond)만큼 대기 한다. 값 0 은 무한대기 뜻한다.	-a sleep 5000
wakeup	N/A	wakeup 액션을 통해서 sleep 하는 쓰레드를 다시 실행 시킬 수 있다.	-a wakeup
wakeup	UID	wakeup 액션을 통해서 주어진 UID 를 따라 sleep 하는 특정한 쓰레드를 다시 실행 시킬 수 있다.	-a wakeup UID
sigsegv	N/A	sigsegv 액션을 통해서 사용자가 대상 시스템에 segmentation fault signal 을 직접 발생 시킬 수 있다.	-a sigsegv

(3) Trigger Count
트리거 카운트를 설정할 수 있는 기능을 제공한다.
고객이 트리커 카운트 수치를 설정하여 수치를 도달할 때까지 결함이 발생하지 않고 도달하면 결함이 발생한다는 것을 말한다.
(4) Call Path Support
결함 발생하는 순서를 제어 기능을 제공한다.
FIT 테스트 대상 시스템의 복잡도로 인해 결함이 발생한다는 결과가 같지만 실제 해당 결함을 유발 시키는 순서가 다를 수 있다.
이에 따라 결함을 유발 시키는 순서를 제어하여 고객이 원하는 순서대로 결함이 발생하는지 검증할 수 있다.
(5) Distribution
분산 처리 기능을 제공한다.
테스트 대상 시스템이 두 개 혹은 그 이상인 경우 FIT 수행 시 해당 결함은 오동작이 발생할 수 있다.
이에 따라 고객은 분산 처리 기능을 이용하여 결함이 발생할 특정 시스템을 정의하여 테스트 오동작을 피할 수 있다.
(6) Iteration
관리 중인 결함을 반복 사용 기능을 제공한다.
관린 중인 결함은 대다수 일회용이기 때문에 한번 발생한다면 더 이상 해당 결함이 유효하지 않는다.
고객이 결함 반복 사용 기능을 이용하여 특정 결함을 반복적으로 발생 시킬 수 있다.
(7) Fault Library
결함을 재활용하기 위한 Fault Library 를 제공한다.

그림 4 는 FIT 수행 결과이다.



(그림 4) FIT 수행 결과

FIT 수행 결과 '[ERR-0109D : Insufficient memory]' 메세지를 볼 수 있다. 이는 특정 결함이 삽입된 라인이 수행된 결과이고, 따라서 수행된 결함을 확이하기위해서는 FIT Client 의 명령어 lst 를 통해 상세 결함 정보를 확인할 수 있다.

4. 결론

본 연구에서는 빅데이터 시대에 핵심 기술요소인 인메모리 DBMS 에 결함을 주입하고 그 결과를 확인할 수 있는 결함주입 테스트 도구의 개념과 구현 결과를 설명했다. 수 십만 라인이 넘는 대규모의 소프트웨어이며, 분산 병렬 환경에서 네트워크와 컴퓨팅 자원에 영향을 많이 받는 소프트웨어의 경우 수동으로 극한의 테스트를 수행하기란 매우 어렵다. 본 연구는 이러한 도전 과제를 극복하고, 효율적으로 오류를 검출하고 예외사항 처리 및 에러 메시지 출력을 검증했다.
향후에는 FIT 의 다양한 결함 타입을 추가하여 고객의 고확정 요구사항이 증대하는 최근 현황에서 분산 환경에 대한 결함을 추가하고 개선할 계획이다.

참고문헌

[1] S. Shin and S. Cho, Smart-car Software Engineering, Acon, 2013.
[2] R. Rana, Improving Fault Injection in Automotive Model Based Development using Fault Bypass Modeling, Computer Science & Engineering, M. S. Thesis, Chalmers, University of Gothenburg, Gothenburg, Sweden, 2013.
[3] R. Thorhuus, Software Fault Injectikon Testing, M. S. Thesis, HTH, Royal Institute of Technology, Stockholm, 2000.

국내 기업의 SW 개발보안 적용사례 분석

박난경, 최진영, 임종인

고려대학교 정보보호대학원
서울 성북구 안암로 145
ranpark@korea.ac.kr, choi@formal.korea.ac.kr, jilim@korea.ac.kr

요약: 해킹 및 보안취약점 등으로부터 안전한 정보시스템을 개발하기 위해서는 정보시스템의 개발단계에서 보안을 고려해야 한다. 보안취약점의 원인이 되는 보안약점을 예방하기 위한 시큐어코딩, 등이 전자정부 시스템 구축 시 적용되고 있다. 그러나 소프트웨어 개발 전 단계가 아닌 일부 단계에서의 보안활동은 소스코드로 인한 보안취약점의 일부분을 예방할 수는 있으나 보안 규제 미 준수, 설계 오류 등에 대해서는 대처가 어려운 측면이 있다. 그러므로 보안 취약성 발생원인을 최소화하여 안전한 정보시스템을 구축하기 위해서는 SW 개발의 모든 단계에서 보안을 고려하여야 한다. 본 논문에서는 국내 기업에서 안전한 정보시스템 구축을 위해 적용하였던 개발단계 보안 강화 프로세스의 사례 (P-SDLC) 를 소개하고 두 개의 주요 개발보안 생명주기와 특성을 비교하여 이를 한국 기업에 적합한 SW 개발보안 방법론으로 제시한다.

핵심어: 소프트웨어 개발 생명 주기 (SDLC), 개발단계 보안을 강화한 프로세스 (secure SDLC), SW 개발보안, 보안약점, 보안취약점

1. 서론

장난감 업체 브이텍 (Vtech) 의 어린이 고객정보 유출, 미 인사관리국의 인사정보 유출, 애슐리 메디슨 (Ashley Madison)사의 고객 정보 유출, 크라이슬러사의 지프 체로키 차량 리콜, 병원 수액펌프인 심빅 (Symbig) 리콜 등은 2015 년 동안 발생한 대표적인 사이버 공격 사례이다 [1]. 이와 같이 사이버 공격은 분야와 대상을 막론하지 않고 도처에서 발생하고 있으며 SW 의 보안취약점을 이용한 지능적인 수법으로 지속적인 공격을 수행하는 APT 공격 또한 확산되고 있다.

이러한 사이버 공격은 응용 소프트웨어 자체에 내재된 보안취약점을 악용하는 것으로 약 75%가 어플리케이션에서 발생하는 것으로 알려져 있다 [6]. 그러므로 사이버 공격으로부터 안전한 정보시스템을 구축하기 위해서는 SW 개발 시 보안 취약성을 최소화하는 것이 매우 중요하게 되었다.

SW 개발과정은 SW 개발 생명주기 (SDLC: Software Development Life Cycle) 로 나타내며 이는 주어진 예산과 자원으로 개발방법, 개발환경 그리고 개발관리에 대한 포괄적인 접근방법을 설정하여 보다 높은 품질의 소프트웨어를 만들어 내기 위한 개발 프로세스라고 할 수 있다.

SW 개발보안은 SDLC 상에서 개발자 실수, 논리적 오류 등으로 인해 소프트웨어에 내포될 수 있는 보안취약점의 원인이 되는 보안약점을 최소화하고 사이버 보안 위협에 대응할 수 있는 안전한 SW 를 개발하기 위한 일련의 보안활동이다.

SW 개발보안의 대표적인 사례는 마이크로소프트사의 SDL (Security Development Lifecycle) [4] 과 미국토 안보부를 중심으로 한 Seven Touchpoint 가 있다 [2]. 국내에는 공공기관 정보시스템 구축단계에 적용되는 행정자치부의 소프트웨어개발보안제도가 있다 [7].

SW 개발보안제도는 MS SDL 이나 Seven Touchpoint 와 달리 구현단계를 중심으로 시큐어 코딩, 소스코드 보안약점 진단 등의 보안활동을 적용하고 있다.

그러나 구현 또는 테스트 단계에 국한된 보안활동은 소스코드 보안약점을 예방할 수는 있으나 설계 오류 등에 대한 대처에는 한계가 있다. 실제로 45 개의 e-비즈니스 어플리케이션을 분석한 결과 보안취약점의 70%가 설계과정의 오류로부터 발생하였다고 보고되었으며 [3], 이러한 설계과정의 오류는 대부분 제품 완성 후 베타테스트나 보안업데이트 및 패치로서 보완하고 있다. 이와 같이 설계단계에서 발생한 보안문제점을 운영 및 유지보수단계에서 해결하기 위해서는 설계단계에서 소요되는 비용보다 60~100 배가 더 소요된다고 한다 [5]. 그러므로 안전한 SW 를 개발하기 위해서는 설계단계를 포함한 모든 개발단계에서 보안을 강화하는 활동을 수행하여야 한다.

본 연구에서는 국내 A 기업 내 정보시스템 구축을 위해 SW 개발 주기의 전 단계에서 보안 강화 활동을 적용하였던 사례를 소개하고자 한다. 이를 위해

2 장에서는 관련 연구로서 MS SDL 및 Seven Touchpoint 를 살펴본다. 3 장에서는 적용한 보안 강화 프로세스 (P-SDLC) 를 소개한 후 MS SDL 과 Seven Touchpoint 와 특징을 비교하고 4 장에서 결론을 맺는다.

2. 관련 연구

Secure SDLC 의 대표사례인 MS SDL 과 Seven Touchpoint 의 주요 활동을 살펴본다.

2.1 MS SDL

MS SDL (MS Secure Development Lifecycle) 은 마이크로소프트사가 2004 년 이후 자사의 모든 SW 개발에 의무적으로 적용하도록 한 SW 개발 생명주기이다. MS SDL 의 목적은 설계, 구현 및 문서화 단계에서 발생할 수 있는 보안약점의 발생을 SW 개발 생명 주기의 초기에 최대한 예방하는 것이다. MS SDL 는 그림 1 과 같이 교육 - 요구사항 - 설계 - 구현 - 검증 - 배포 - 대응 단계에서 16 개의 필수 보안활동을 SW 개발생명주기 중에 수행한다.

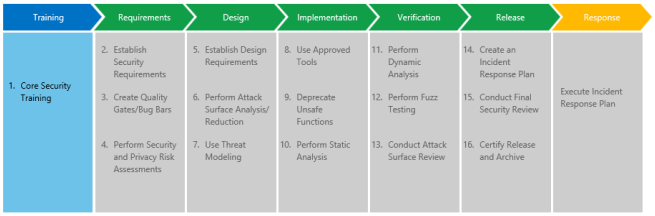


그림 1 MS SDL

- 교육: 모든 개발자가 의무적으로 보안의 개념, 최신 동향, 모범사례를 학습하고 훈련한다.
- 요구사항: 보안 요구사항 수립, 버그 유형 정의, 개인정보보호를 포함한 보안 위험 평가를 한다.
- 설계: 보안 설계검토 등의 설계 요구사항 수립, 공격표면 분석, 위협 모델링을 수행한다.
- 구현: 자동화 도구 사용, 안전하지 않은 기능 사용 중지, 정적분석을 수행하여 시큐어코딩을 준수한다.
- 검증: 동적 분석, 퍼징 테스트, 변경사항에 대한 공격표면 확인을 수행한다.
- 배포: 사고 대응계획을 수립하고, 별도의 보안부서로부터 최종 보안검토를 받는다.
- 대응: 사고 대응계획을 실행한다.

MS SDL 은 특징적으로 개발 전에 수행하는 보안교육을 강조하며 교육 단계는 개발 보안의 수행을 위해 학습하고 준비하는 단계 (Pre-SDL) 이다. 대응 단계는 SW 제품 출시 이후에 필요한 단계 (Post-SDL) 이다.

2.2 Seven Touchpoint

Seven Touchpoint 는 실무적으로 검증된 SW 개발 보안 프로세스로서 보안 강화 기법 (build security in)을 지원하는 중요한 기법 중 하나이다. 이 프로세스는 SDLC 의 각 단계에서 보안 기능과 관련한 모든 활동을 별도로 구분하여 관리한다면 소프트웨어의 보안성을 강화할 수 있다는 통찰에서 출발하였다. 중점적으로 관리해야 할 보안강화 활동으로 7 개의 터치포인트를 정의하였다 (그림 2). 즉, 터치포인트는 코드검토(code review), 아키텍처 위험분석 (architectural risk analysis), 침투테스트 (penetration testing), 위험기반 보안테스트 (risk-based security tests), 악용사례 (abuse cases), 보안 요구사항 (security requirement), 보안운영 (security operation) 으로 정의되며 6 단계의 개발과정에 따라 아래와 같이 구성한다.

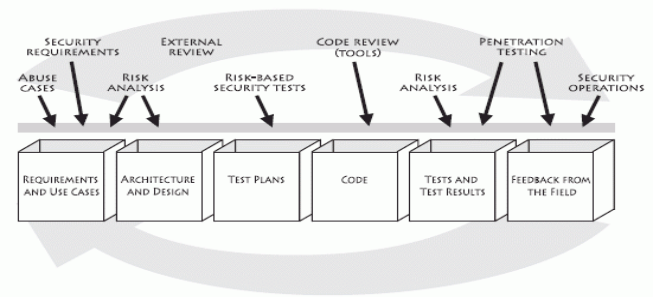


그림 2 Seven Touchpoint

- 요구사항과 유즈케이스: 악용사례, 보안 요구사항 정의 및 명세, 위험분석을 수행한다.
- 구조 및 설계: 공격저항 분석(Attack resistance analysis), 모호성 분석(ambiguity analysis), 약점 분석(weakness analysis) 등을 통해 위험요소를 분석한다.
- 테스트 계획: 공격패턴, 위험분석 결과, 악용사례를 통해 위험기반 보안테스트를 한다.
- 코드: 구현 오류(implementation bug), 특히 소스코드 취약성을 발견하기 위해 정적분석을 통한 코드 검토를 수행한다.
- 테스트와 테스트 결과: 위험분석, 침투테스트를 수행한다.
- 현장 피드백: 침투테스트와 보안운영을 통해 얻은 공격지식을 개발자에게 피드백한다.

3. A 사 SW 개발 생명 주기

A 기업의 기간 정보시스템을 구축하는 대규모 프

로젝트에서 적용하였던 개발단계 보안 강화 프로세스 (P-SDLC) 를 소개한다.

3.1 P-SDLC

구축대상에 해당하는 기간 정보시스템은 ERP, MES, SCM 등을 포함하였고 효과적인 관리를 위해 각 업무의 세부기능 단위로 프로젝트를 분할하여 다수의 프로젝트를 동시에 수행하도록 하였다. 이에 효과적인 보안 관리가 필요하게 되어 기존의 secure SDLC 를 보완하였다. 기존에는 사전 보안검토, 시큐어코딩, 모의해킹을 중점 수행하였으나 모의해킹 시에 중대한 설계상의 결함이 발견되는 경우에 비용, 일정 등의 제약으로 인해 결함에 대한 즉각적인 개선이 어려웠다. 이에 개선된 프로세스에서는 특징적으로 설계단계 보안검토에 중점을 두었고 각 단계 보안 강화 활동의 결과를 종합하여 최종 보안검토를 수행하도록 하였다 (표 1).

단계	계획수립/분석	설계	구현	테스트
보안 활동	-보안검토 -보안교육 -보안 요구사항, 위협요인 식별	- 보안통제 수립 -설계보안 자가점검 -보안부서 확인점검	- 시큐어코딩 -소스코드 보안약점 진단(도구) -진단결과 3 자 확인	- 실행코드 보안취약점 진단 (모의해킹) -최종 보안검토

표 1 개발보안 강화 프로세스 적용사례 (P-SDLC)

- 계획수립/분석: 프로젝트 추진계획과 구조적 점검사항에 대한 개발팀의 답변을 토대로 사전 보안검토를 수행한다. 보안 요구사항 및 위협요인에 대한 식별을 통해 보안규제 준수, 보안타당성을 검토하고, 비용을 분석하며 개발자에 대한 보안교육을 수행한다. 교육내용은 secure SDLC, 보안설계, 시큐어코딩, 개인정보보호, 모의해킹을 통한 취약점 발생사례 등에 중점을 두었다.
- 설계: 기업 내 보안 구조에 기초하여 보안설계를 수행하며 설계보안 점검항목을 기준으로 자가점검을 수행하고 보안부서는 자가 점검 결과를 토대로 확인점검을 함으로써 보안 통제를 수립한다. 이 단계에서는 구현에 앞서 보안 요구사항을 수용하거나 대안을 제시한다.
- 구현: 시큐어코딩을 수행하고 자동화된 정적분석 도구를 활용하여 소스코드 보안약점을 진단한다. 도구 적용결과에 따른 모든 보안약점은 원칙적으로 제거되어야 하며 그 결과에 대한 3 자 확인을 수행한다.
- 테스트: 개발의 마지막 단계로서 실행코드에 대한 모의해킹과 최종 보안검토를 수행한다. 모의해킹은 해당 정보시스템의 주변 연동 시스템까지 포함하여 수행한다. 최종 보안검토는 설계 보안점검을 통해 수립한 보안통제의 적용여부와 각 단계별 보안활동의 결과를 토대로 가동 심의를 지원한다.

3.2 MS SDL, Seven Touchpoint 와 특징 비교

적용 프로세스 (P-SDLC) 를 MS SDL, Seven Touchpoint 와 비교하였다 (표 2).

개발단계	P-SDLC	MS SDL	Seven Touchpoint
교육	- 보안기본교육	- 보안기본교육 - 보안심화교육	- 없음
분석	- 보안검토: 보안 타당성 검토 - 보안요구사항 식별	- 보안 요구사항 수립 - 보안 버그 유형 정의 - 보안위험 평가	- 악용사례 - 보안요구사항 정의 -위험분석
설계	- 보안설계검토	-보안설계검토 -공격표면분석 -위협모델링	- 위협모델링
구현	- 시큐어코딩 - 코드보안약점 진단(도구사용)	- 시큐어 코딩 - 코드보안약점 진단(도구사용)	- 시큐어코딩 - 코드보안약점 진단(도구사용)
테스트	- 실행코드 보안취약점 진단 (침투테스트) - 최종보안검토	- 실행코드 보안취약점 진단 (퍼지테스트, 침투테스트, 공격표면검토)	- 실행코드 보안취약점 진단 (침투테스트)
배포 및 지원	- 없음	- 최종보안검토 - 사고대응방안	- 없음

표 2 P-SDLC 와 MS SDL, Seven Touchpoint 비교

MS SDL 은 개발 보안을 전담하는 인사구조를 가지는 중량급의 대기업에서 적용 가능한 프로세스이다. Seven Touchpoint 는 보안활동을 개발활동과 구분하여 개발산출물을 중심으로 수행하는 경량급의 유연한 프로세스이다. P-SDLC 는 개발에 직접 참여하지는 않지만 개발 대상 업무를 충분히 이해할 수 있는 보안조직에 의해 수행되었으므로 MS SDL 보다는 가벼우나 Seven Touchpoint 보다는 다소 무거운 특징

을 가지고 있다.

- 교육 : MS SDL 은 연 1 회 이상 개발자에게 의무적으로 보안교육을 수행한다. 반면 Seven Touchpoint 는 교육을 강조하지 않는다. P-SDLC 역시 교육을 중시하며 교육내용은 MS SDL 의 교육내용과 유사하다.
- 분석: MS SDL 은 보안위험 평가와 보안버그 유형을 정의하고 보안 요구사항을 수립한다. Seven Touchpoint 는 악용사례와 위험분석을 통해 보안 요구사항을 정의한다. P-SDLC 는 사전 보안검토를 통해 보안 요구사항 및 보안 위협요인을 식별한다.
- 설계: MS SDL 과 Seven Touchpoint 는 위협모델링을 수행하고 MS SDL 과 P-SDLC 에서는 보안설계 검토를 수행한다. MS SDL 은 공격표면 분석을 수행한다.
- 구현: 공통적으로 시큐어코딩을 수행하고 정적 분석 도구를 활용하여 소스코드 보안약점을 진단한다.
- 테스트: 공통적으로 실행코드 보안취약점 진단을 위한 침투 테스트를 수행하며 MS SDL 은 퍼지 테스트와 공격표면 검토를 수행한다. P-SDLC 는 최종 보안검토를 수행한다.
- 배포 및 지원: MS SDL 에서만 사고 대응계획을 수립한다.

4. 결론

이상으로 국내 A 기업에 적용하였던 SW 개발보안 활동 (P-SDLC) 을 소개하였다. 국내 적용사례인 P-SDLC 는 MS SDL 보다는 가벼우나 Seven Touchpoint 보다는 강화된 특징을 가지고 있다. 또한 설계 단계에서 수행하는 보안 설계검토와 테스트 단계에서 수행하는 최종 보안검토를 강조하였다. 이는 보안 요구사항과 위협요인에 따라 설계 보안검토를 수행하고 설계사양의 변경을 추적한다는 의미에서 MS SDL 과 Seven Touchpoint 의 위협 모델링과 부분적으로 비교될 수 있다.

P-SDLC 를 다수의 프로젝트에 적용해 본 결과 특히 보안 설계검토와 최종 보안검토를 통해 보안통제의 적용을 효과적으로 확인할 수 있었다. 가령, 설계 단계에서 개인정보보호 요구사항에 대한 보안통제를 수립하는 과정에서 개발 범위 내 규제의 적용 범위를 최소화함으로써 개인정보 보호조치를 매우 효과적으로 수행하였다. 이와 같이 P-SDLC 와 같은 SW 보안 강화 프로세스를 공공 및 기업의 정보시스템에 확대하여 적용한다면 보안 규제 준수는 물론 핵심적인 보안 구조에 대한 일관성 있는 운영이 가능해질 것이다. 이는 결국 보안 운영 및 유지보수 비용을 절감시키고 국내 어플리케이션의 보안 수준을 한 계단 끌어올리는 계기가 될 것이다.

향후 의료, 자동차 등을 포함하는 사물 인터넷 분야에서 보안취약성은 인명에 영향을 미칠 정도로 심각한 결과를 초래할 수 있으므로 관련 어플리케이션에 대한 보안 강화활동이 매우 중요하다. 인명과 안전이 결부된 융합보안 분야에서도 P-SDLC 와 같은 secure SDLC 를 적용하는 것은 보안약점을 사전에제거하여 보안취약점의 발생을 예방함으로써 정보시스템의 안전을 위한 최소 기준선을 확보하는데 일조할 것이며, 또한 한국형 secure SDLC 개발의 초석이 될 것이라고 생각된다.

참고문헌

[1] Cadie Thompson, " The 14 scariest hacks of 2015, Techinsider, " http://www.techinsider.io/14-of-the-scariest-hacks-in-2015-8, Dec. 13, 2015

[2] G. McGraw, "Software Security: Building Security In, " Addison Wesley, 2006

[3] James W. Over, CMU Software Engineering Institute, " The Team Software Process for Secure Systems Development, " 2002 http://resources.sei.cmu.edu/asset_files/Presentation/2002_017_001_24393.pdf

[4] Microsoft, "Security Development Lifecycle, " https://www.microsoft.com/en-us/sdl/default.aspx, 2015

[5] P. Mohan, A. Udaya Shankar, K. JayaSriDevi, " Quality Flaws: Issues and Challenges in Software Development, " Computer Engineering and Intelligent Systems, ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online), Vol 3, No.12, 2012, http://www.iiste.org/Journals/index.php/CEIS/article/viewFile/3533/3581

[6] Theresa Lanowitz, " Now Is the Time for Security at the Application Level, " 1 December 2005, https://www.sela.co.il/_Uploads/dbsAttachedFiles/GartnerNowIsTheTimeForSecurity.pdf

[7] 행정자치부, " 행정기관 및 공공기관 정보시스템 구축 · 운영 지침, " 2013. 8

사물인터넷 환경에서 지리적 응집도를 고려한 동적 서비스 검색방법

백경덕, 김민협, 고인영

한국과학기술원 전산학부
대전광역시 유성구 대학로 291(구성동 373-1)
{blest215, minhyeop.kim, iko}@kaist.ac.kr

요약: 사물인터넷 환경에서 사용자가 자신의 태스크를 수행하기 위해서는, 필요한 사물인터넷 기반의 서비스를 검색하는 과정이 필요하다. 이 때 검색된 서비스들이 사용자 태스크를 효과적으로 수행하기 위해서는 지리적으로 서로 인접하여 위치해야 한다. 이러한 문제를 해결하기 위해 본 논문에서는 동적 모바일 애드혹 네트워크 환경에서 지리적 응집력을 갖는 두 가지의 서비스 검색 방법을 제안하고, 지리적 응집도를 바탕으로 기존의 서비스 검색 방법과 비교, 평가하였다.

핵심어: Distributed Service Discovery, Geographically Cohesive Services, Task-Oriented Computing, Internet of Things

1. 연구 개요

최근, 사용자가 자신 주위의 사용 가능한 사물인터넷(Internet of Things, IoT)[1] 기반의 서비스를 동적으로 탐색하고 활용하도록 하는 기술의 필요성이 부각되고 있다. 따라서 기기의 가용성이 수시로 변하는 동적 IoT 환경에서도 안정적으로 IoT 기반의 서비스를 검색하고 추적하는 방법에 대한 연구가 필요하다.

IoT 환경에서 사용자 태스크(User Tasks)를 기반으로 서비스를 제공하는 프레임워크는 태스크, 서비스, 자원의 3 단계로 구성된다[2]. 먼저, *태스크(Task)*는, 사용자 태스크 수행을 위해 필요한 서비스들의 조합으로 이루어진다. *서비스(Service)*는 IoT 기기(자원)을 통해 제공되는 기능의 추상화된 개념이다. *자원(Resource)*은 서비스 제공을 위해 필요로 하는 기능을 실제로 수행하는 물리적인 장치, 즉, IoT 기기이다. 일반적으로 자원은 제한된 연산 능력을 가지고 한정된 범위의 기능을 수행한다. 따라서 자원들은 연산 능력이 크고 및 네트워킹을 제공하는 *게이트웨이(Gateway)*에 연결되어 작동하며, 경우에 따라서는 게이트웨이가 자원을 위해 복잡한 연산을 수행하기도 한다.

자원들이 연결된 게이트웨이들은 서로 모바일 애드혹 네트워크(Mobile Ad-hoc Network, MANET)[3]를 구성하여, 사용자가 가장 가까운 게이트웨이에 연결함으로써 네트워크 내의 모든 서비스에 접근할 수 있도록 한다. 이 때, 게이트웨이들이 인접한 게이트웨이와의 통신을 순차적으로 수행하여 직접 연결되지 않은 게이트웨이와도 통신이 가능한데, 이 과정을 *라우팅(Routing)* 이라고 한다. 그리고 이때 사용된 알고리즘을 *라우팅 프로토콜(Routing Protocol)*이라 한다.

사용자 태스크 수행에 필요한 서비스 검색을 위해 기존 서비스 검색방법을 사용하면 서비스들이 독립적으로 검색된다. 이렇게 검색된 서비스들은 서로 다른 지리적 위치에 있는 IoT 자원을 사용하기 때문에 특정한 위치에서 사용자의 태스크를 수행하기에 적합하지 않을 수 있다. 따라서 본 논문에서는 사용자 태스크를 고려하여 지리적 응집력을 갖는 IoT 서비스를 검색하는 새로운 알고리즘을 제안한다.

2. 지리적 응집도를 고려한 서비스 검색방법

IoT 자원 연결 관계는 가중치 무방향 그래프(Weighted Undirected Graph)로 표현할 수 있다. 이 때 사용자 태스크에서 필요한 서비스를 지리적으로 응집된 공간에서 찾는 문제를 그래프로 변환하여, Traveling Salesman Problem(TSP)과의 대응을 통해 NP-hard 임을 증명할 수 있다. 따라서 이 연구에서는 TSP 문제를 풀기 위해 개발된 휴리스틱 알고리즘인 그리디 삽입(Greedy Insertion)과 그리디 여행(Greedy Traveling) 방법을 확장하여 근사적으로 해를 구한다.

그리디 삽입 방법은 게이트웨이 간의 연결을 파악하여 지리적으로 응집력을 갖는 게이트웨이들을 선택한다. 이 때 순차적으로 선택되는 게이트웨이들은 네트워크의 모든 연결 중에 가장 응집도 높은 집합을 형성할 수 있는 게이트웨이이다. 따라서 실행 시간은 오래 걸릴 수 있으나, 본 논문에서 목표한 바와 같이 더 높은 지리적 응집도의 자원 집합을 검색할 수 있다. 그러나 그리디 삽입 방법은 네트워크 내의 모든 연결을 파악하고 있는 경우에만 사용할 수 있으므로, 근접한 게이트웨이의 정보만 알고 있는 분산 IoT 환경에는 그 적용에 어려움이 있다.

그리디 여행 방법은 라우팅 프로토콜을 기반으로 하여 작동한다. 마지막으로 선택된 게이트웨이로부터 라우팅 정보를 바탕으로 가장 응집도 높은 게이트웨이를 순차적으로 선택해 나간다. 인접한 게이트웨이 중에서 다음 게이트웨이를 선택하기 때문에 전체 네트워크가 파악되지 못하는 분산 서비스 환경에서도

사용 가능하다. 실제 환경에서는 넓은 지역에 분포된 많은 수의 게이트웨이 위치 정보를 모두 파악하여 관리하는 것이 불가능하므로 그리디 삽입 방법보다 그리디 여행 방법이 더 효과적일 것이다.

3. 실험 및 평가

2 절에서 제시된 두 가지 서비스 검색 방법과 기존의 서비스 검색 방법을 비교하기 위해 무작위로 게이트웨이를 생성하여 시뮬레이션하였다. 실험에는 NS3 시뮬레이터[1]를 사용하였으며, 게이트웨이를 무작위로 배치하고, 와이파이를 사용해 무선 모바일 애드혹 네트워크를 형성한 다음, 애드혹 네트워크를 형성하는 과정에서 생성된 라우팅 테이블을 사용해 세 가지 방법을 실행하는 순으로 진행되었다. 각 방법은 검색된 게이트웨이 간의 응집도와 실행에 소요된 시간으로 평가하였다. 응집도는 0 과 1 사이의 값으로, 값이 높을수록 게이트웨이 집합이 서로 응집되어 있음을 나타낸다.

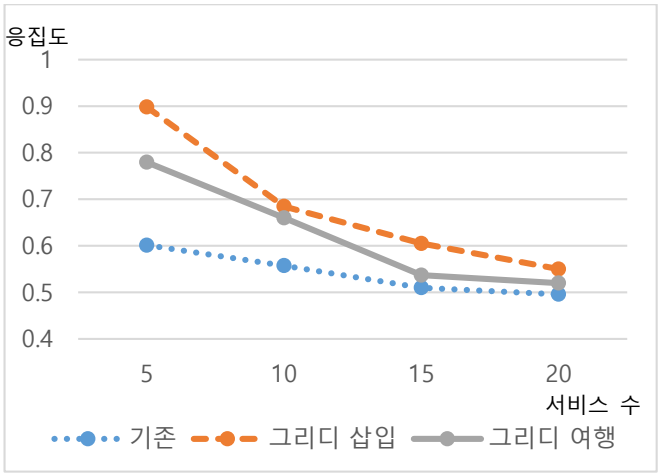


그림 1. 서비스 수에 따른 서비스 응집도 측정 결과

시뮬레이션 결과, 그림 1 과 같이 그리디 삽입 방법이 가장 높은 응집도를 보였으며, 그리디 여행 방법이 그 다음으로 높은 응집도를 보였다. 그리디 삽입 방법은 네트워크의 모든 연결 정보를 알고 있다는 가정 하에 실행되므로 가장 높은 응집도를 보인 것으로 분석된다.

실행 시간은 그리디 삽입 방법과 기존 방법이 비슷한 값을 보였고, 그리디 여행 방법이 가장 낮은 값을 보였다. 다른 두 방법과 그리디 여행 방법의 차이는 게이트웨이를 순차적으로 선택한다는 점이다. 다른 두 방법은 알고리즘을 통해 게이트웨이를 선택 후 일괄적으로 통신하기 때문에 와이파이 채널에 부담을 주어 통신 속도가 느려진 것으로 분석된다.

[1] https://www.nsnam.org/

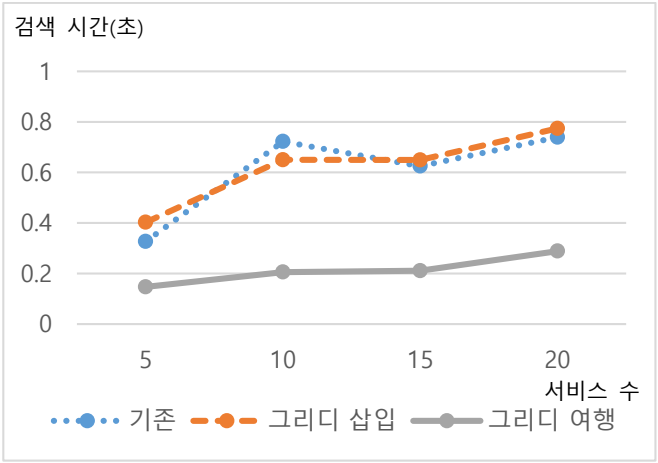


그림 2. 서비스 수에 따른 실행 시간 측정 결과

4. 결 론

본 논문에서는 여러 서비스들로 구성된 사용자 태스크를 유효하게 수행하기 위해 지리적으로 응집력을 갖는 자원 집합을 검색하는 두 가지 방법을 제시하였다. 그 결과 TSP 문제의 휴리스틱 해법인 그리디 여행을 응용한 방법이 더 우수한 성능을 보였다.

사용자 태스크를 구성하는 서비스들은 지리적으로 응집되어 있을 때 시너지 효과를 가진다. 이는 서비스의 수가 늘어남에 따라 영향력이 커지므로, 서비스의 종류가 많아졌을 때 검색된 서비스가 지리적으로 응집되어 있지 않다면 태스크를 유효하게 수행할 수 없다. 앞으로 더 많은 종류의 서비스가 개발되어 감에 따라 지리적으로 응집되어 있는 자원들을 검색하는 방법의 중요도는 더 늘어날 것으로 보인다.

Acknowledgement

본 연구는 민군겸용기술사업(Dual Use Technology Program) 지원을 받아 수행되었음(UM13018RD1).

참고문헌

[1] Gubbi, Jayavardhana, et al. "Internet of Things (IoT): A vision, architectural elements, and future directions." *Future Generation Computer Systems* 29.7 (2013): 1645-1660.

[2] Jimenez-Molina, Angel, and In-Young Ko. "Spontaneous task composition in urban computing environments based on social, spatial, and temporal aspects." Engineering Applications of Artificial Intelligence 24.8 (2011): 1446-1460.

[3] Macker, Joseph. "Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations." (1999).

분산 이동 실시간 시스템의 이동성 명세와 검증을 위한 시각화 도구

최영복, 박현주, 이문근

전북대학교 컴퓨터공학과
전북 전주시 덕진구 백제대로 567
moonkun@jbnu.ac.kr

요약: 본 연구에서는 ADOxx 메타-모델링 플랫폼을 사용하여 개발된 SAVE 도구를 사용하여 분산 이동 실시간 시스템의 명세와 검증을 위한 이중 접근 방법을 제안한다. 명세 단계에서는 프로세스의 이동성을 표현하고 시각화 한 언어인 δ-Calculus 를 사용하여 지정학적 공간에 프로세스의 이동을 표현한다. 검증 단계에서는 GTS Logic 을 사용하여 이동간의 의존성과 같은 요구사항을 시각화하여 표현한다. 이 두 가지 시각화 방법은 시스템의 명세와 검증을 더욱 이해하기 쉽게 할 수 있다. SAVE 는 시스템의 이동에 대한 요구사항의 명세와 검증을 시각화 할 수 있는 도구이다.

핵심어: δ-Calculus, SAVE, 프로세스 대수, 이동성, 시간, 공간

1. 서론

대부분의 실시간 시스템은 많은 수의 분산된 이동 프로세스들이 서로 상호작용하는 형태로 구성된다. 이러한 시스템의 거대함과 복잡성은 시스템의 명세와 검증을 매우 어렵게 한다. 또한 공간적으로 분산된 프로세스들의 상호작용과 이동은 이를 더욱 어렵게 한다. 따라서 이러한 시스템의 명세와 검증을 용이하게 하기 위한 새로운 방법론이 필요하다.

본 논문에서는 분산 이동 실시간 시스템[1]을 명세하고 검증하기 위하여 ADOxx 메타-모델링 플랫폼[2]을 사용하여 개발된 도구인 SAVE(Specification, Analysis and Verification Environment)를 사용한 이중 접근방법을 제안한다.

SAVE 를 사용한 이중 접근방법은 다음의 정형 기법을 사용한다:

1) δ-Calculus: δ-Calculus 는 여러 종류의 프로세스 동기 이동을 정의한 프로세스 대수이다. 다른 프로세스 대수들[3,4,5]과 비교하여, δ-Calculus 는 프로세스의 이동과 분산성을 더욱 명확히 표현할 수 있다. 특히, 이동과 분산성을 효과적으로 표현하기 위해 SAVE 를 사용하여 시각화 하였다.
2) GTS Logic: GTS(geo-Temporal Space) Logic 은 이동간의 의존성을 검증하기 위한 1 차 논리이다. GTS Logic 의 프로세스, 액션, 프로세스 간의 상호작용은 모두 시간과 공간 속성을 지닌다. GTS Logic 의 목표는 GTS 에 의존성을 시각화하여 표현하는 것이다. 다른 시간, 공간 논리[6,7,8]들과 비교하여, GTS Logic 은 이러한 의존성을 시각화 하여 표현할 수 있다는 특징이 있다.

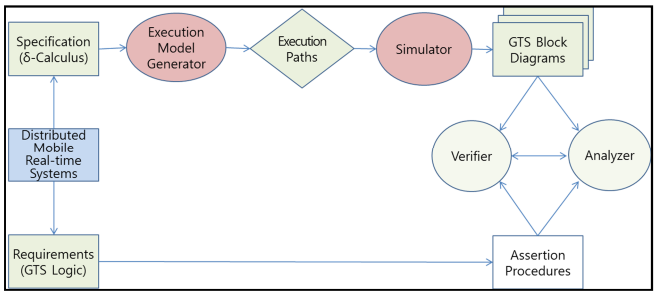


그림 1 명세와 검증을 위한 이중 접근방법

시스템의 명세와 검증을 위한 방법은 그림 1 에 표현되어 있다.

1) 시스템의 프로세스와 이동을 δ-Calculus 를 사용하여 명세한다.
2) 1)에서의 명세를 기반으로 실행 모델을 생성한다.
3) 2)에서 생성된 실행 모델에서 하나의 경로를 선택하고 이를 시뮬레이션 한 뒤, 시뮬레이션의 결과를 GTS 블록 다이어그램으로 생성한다.
4) 시스템의 요구사항을 GTS Logic 으로 표현하고, 이를 3)에서 생성된 GTS 블록 다이어그램에 표현한다.
5) 검증기를 사용하여 다이어그램에 표현된 요구사항이 충족되었는지를 검사한다. 요구사항의 충족 여부는 표현된 요구사항의 색상을 통해 확인할 수 있다.

SAVE 의 목표는 위의 모든 명세와 검증 과정을 다음의 그래프들을 사용하여 시각화 하는 것이다.

1) ITL: 이 그래프는 시스템의 전체적인 구성과

프로세스들의 통신, 이동과 같은 상호작용을 표현하는 시스템 관점의 그래프이다.

2) ITS: 이 그래프는 각각의 프로세스들이 실행하는 통신, 이동 등의 액션을 상세하게 표현한 프로세스 관점의 그래프이다. 이동 등의 액션에 의한 프로세스의 상태 변화는 ITL 에 적용되어 반영된다.
3) EM(Execution Model) 그래프: 이 그래프는 시스템의 모든 실행 가능한 경로에 대한 실행 결과를 보여주는 도달성 그래프[9]이다.
4) GTS 블록 다이어그램: EM 그래프에서 하나의 경로를 선택하여 이를 시뮬레이션 한 결과를 표현하는 그래프이다. 시뮬레이션의 결과로 나타난 모든 프로세스, 액션, 상호작용은 블록으로 표현된다.
5) GTS 요구사항 다이어그램: GTS 블록 다이어그램에 GTS Logic 을 사용하여 표현한 요구사항을 추가하여 시각적으로 표현한 그래프이다. 각각의 요구사항은 블록간의 연관관계로 표현된다.

SAVE 는 디자이너와 엔지니어에게 거대하고 복잡한 시스템을 시각적으로 명세하고 검증할 수 있는 포괄적이고 통합적인 기능을 제공할 수 있다.

SAVE 는 시스템의 기능적 요구사항의 명세와 보안적 요구사항의 검증을 모두 시각화할 수 있는 도구이다. 또한 ADOxx 는 다른 유사한 도구들[10,11,12]과 비교하여, 이러한 시각화를 위해 필요한 기능을 제공해주는 매우 뛰어난 메타-모델링 도구이다.

2. 목적

본 논문의 주 목적은 SAVE 를 사용하여 시스템의 명세와 검증의 모든 과정을 시각화 하는 것이다. 이는 다음의 시각적 속성을 지니는 그래프들을 사용하여 구성된다.

1) ITL: 시스템 관점의 그래프인 ITL 은 다음의 속성을 지닌다:
   - 공간: 공간은 노드들과 노드를 연결하는 선들로 구성된다. 각각의 노드는 특정 영역을 표현하며, 노드간의 포함관계가 허용된다.
   - 이동성: 이동 액션에 의한 프로세스의 이동을 표현한다.
2) ITS: 프로세스 관점의 그래프인 ITS 는 통신, 이동 등의 액션을 표현한다. 이러한 액션의 실행으로 인한 프로세스 구조의 변화는 ITL 에 반영된다.
   - 액션의 타입: 각각의 액션에 대한 타입, 모드 등의 속성들이 시각화되어 표현된다.
   - 시간: 각각의 액션은 준비 시간, 실행 시간 등의 시간 속성을 지니고 있다.
3) EM 그래프: EM 그래프는 다음의 속성을 지니는 도달성 그래프이다.
   - 실행 공간: 모든 실행 가능한 경로가 시각화되어 표현된다.
   - 추상화: 무한히 실행되는 경우에는 이를 그래프로 표현하기 어렵다. 따라서 루프와 같은 형태들은 추상화하여 표현해야 한다.
   - 시뮬레이션: 각각의 경로는 시스템의 분석과 검증을 위하여 모두 시뮬레이션된다.
4) GTS 블록 다이어그램: 이 다이어그램은 시뮬레이션의 결과를 표현한다. 프로세스, 액션, 상호작용은 모두 블록으로 표현된다.
   - 공간 속성: 모든 블록은 반드시 공간 속성을 지닌다.
   - 시간 속성: 모든 블록은 반드시 시간 속성을 지닌다.
5) GTS 요구사항 다이어그램: 모든 요구사항은 GTS Logic 을 사용하여 표현된다. 시각화를 위해 다음의 속성이 필요하다.
   - 관계: 모든 요구사항은 GTS 블록간의 관계로 구성된다. 관계는 적법하고 적절해야 한다.
   - 결합: 한 요구사항은 다른 요구사항들의 결합된 형태로 구성될 수 있다.

또한 이러한 시각화 그래프의 통합을 통해 일관성을 보장할 수 있다.

3. 방법론

3.1 δ-Calculus

δ-Calculus 는 분산된 환경에서의 실시간 시스템을 명세하기 위한 프로세스 대수이다. 이 프로세스 대수에서 시스템은 프로세스들의 집합으로 구성되며, 각각의 프로세스는 다른 프로세스를 포함할 수 있다. 각각의 프로세스는 통신, 이동, 제어와 같은 액션들의 순서로 구성된다. 프로세스의 실행은 액션들이 순차적으로 실행되며, 연산자에 따라 분기나 반복이 발생할 수 있다. δ-Calculus 에 대한 문법은 표 1 과 같다. δ-Calculus 의 특징은 여러 종류의 이동을 정의한 것이다. 각각의 액션에는 우선순위와 시간 속성을 정의할 수 있다. 한 액션이 실행될 때, 그와 연동되는 다른 액션은 실행되는 액션의 실행 시간과 겹치는 구간이 있어야만 연동될 수 있다.

$P ::= A$	Action
$\mid P_{(n)}$	Priority
$\mid P[Q]$	Nesting
$\mid P\langle r_t \rangle$	Channel
$\mid P + Q$	Choice

	$\mid P \parallel Q$	Parallel
	$\mid P\backslash_t F$	Exception
	$\mid A.P$	Sequence
$A ::= \emptyset$		Empty Action
	$\mid r_t(\overline{msg})$	Communication(Send)
	$\mid r_t(msg)$	Communication(Receive)
	$\mid M$	Movement
$M ::= m_t^p(k)\ P$		Movement Request
	$\mid P\ m(k)_t$	Movement Permission
$m ::= in \mid out \mid get \mid put$		Movement Action
$C ::=$ new P		Create Process
	$\mid$ kill P	Kill Process
	$\mid$ exit	Exit Process

표 1 δ-Calculus 의 문법

δ-Calculus 의 의미론은 전이 규칙으로 표현된다. 이러한 전이 규칙은 다음의 구조를 따른다.

$$\frac{\text{premise}}{\text{conclusion}}(\text{side condition})$$

표 2 는 δ-Calculus 의 통신에 대한 의미론을 나타낸다.

Action	$\frac{-}{r(a).P \xrightarrow{r(a)} P}$
ChoiceL ChoiceR	$\frac{P_{(n)} \xrightarrow{a} P'_{(n)}}{P_{(n)} + Q_{(n)} \xrightarrow{a} P'_{(n)}}$ $\frac{Q_{(n)} \xrightarrow{a} Q'_{(n)}}{P_{(n)} + Q_{(n)} \xrightarrow{a} Q'_{(n)}}$
ChoiceP	$\frac{P_{(n)} \xrightarrow{a_1} P_{(n)}', Q_{(m)} \xrightarrow{a_2} Q_{(m)}'}{P_{(n)} + Q_{(m)} \xrightarrow{a_1} P_{(n)}'}(n > m)$
ParIL ParIR	$\frac{P \xrightarrow{a} P'}{P \parallel Q \xrightarrow{a} P' \parallel Q}$ $\frac{Q \xrightarrow{a} Q'}{P \parallel Q \xrightarrow{a} P \parallel Q'}$
ParCom	$\frac{P \xrightarrow{a} P', Q \xrightarrow{\bar{a}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$
NestO NestI	$\frac{P \xrightarrow{a} P'}{P[Q] \xrightarrow{a} P'[Q]}$ $\frac{Q \xrightarrow{a} Q'}{P[Q] \xrightarrow{a} P[Q']}$
NestCom	$\frac{P \xrightarrow{a} P', Q \xrightarrow{\bar{a}} Q'}{P[Q] \xrightarrow{\tau} P'[Q']}$
ExT	$\frac{-}{(a_t.P)\backslash Q \rightarrow (a_{t-1}.P)\backslash Q}(t > 1)$
ExN	$\frac{-}{(a_t.P)\backslash Q \xrightarrow{a} P}(t > 0)$
ExE	$\frac{-}{(a_t.P)\backslash Q \rightarrow Q}(t = 0)$

표 2 δ-Calculus 의 통신 의미론

표 3 은 δ-Calculus 의 이동에 대한 의미론을 나타낸다.

In	$\frac{P \xrightarrow{in_t(k)\ Q} P', Q \xrightarrow{P\ in\ (k)} Q'}{P \parallel Q \xrightarrow{\delta} Q'[P']}$
Out	$\frac{P \xrightarrow{out_t(k)\ Q} P', Q \xrightarrow{P\ out\ (k)} Q'}{Q[P] \xrightarrow{\delta} P' \parallel Q'}$
Get	$\frac{P \xrightarrow{get_t(k)\ Q} P', Q \xrightarrow{P\ get\ (k)} Q'}{P \parallel Q \xrightarrow{\delta} P'[Q']}$
Put	$\frac{P \xrightarrow{put_t(k)\ Q} P', Q \xrightarrow{P\ put(k)} Q'}{P[Q] \xrightarrow{\delta} P' \parallel Q'}$
InP	$\frac{P_{(n)} \xrightarrow{in_t^p(k)\ Q_{(m)}} P'_{(n)}}{P_{(n)} \parallel Q_{(m)} \xrightarrow{\delta} Q_{(m)}[P_{(n)}']}(n \geq m)$
OutP	$\frac{P_{(n)} \xrightarrow{out_t^p(k)\ Q_{(m)}} P'_{(n)}}{Q_{(m)}[P_{(n)}] \xrightarrow{\delta} P'_{(n)} \parallel Q_{(m)}}(n \geq m)$
GetP	$\frac{P_{(n)} \xrightarrow{get_t^p(k)\ Q_{(m)}} P'_{(n)}}{P_{(n)} \parallel Q_{(m)} \xrightarrow{\delta} P'_{(n)}[Q_{(m)}]}(n \geq m)$
PutP	$\frac{P_{(n)} \xrightarrow{put_t^p(k)\ Q_{(m)}} P'_{(n)}}{P_{(n)}[Q_{(m)}] \xrightarrow{\delta} P'_{(n)} \parallel Q_{(m)}}(n \geq m)$
InN	$\frac{P \xrightarrow{in_t(k)\ Q} P', Q \xrightarrow{P\ in\ (k)} Q'}{P \parallel Q[R] \xrightarrow{\delta} Q'[P' \parallel R]}$
GetN	$\frac{P \xrightarrow{get_t(k)\ Q} P', Q \xrightarrow{P\ get\ (k)} Q'}{P[R] \parallel Q \xrightarrow{\delta} P'[R \parallel Q']}$

표 3 δ-Calculus 의 이동 의미론

δ-Calculus 에서 사용되는 대수 규칙은 표 4 와 같다.

$P + P = P$	Choice(1)
$P + Q = Q + P$	Choice(2)
$(P + Q) + R = P + (Q + R)$	Choice(3)
$P \parallel \emptyset = P$	Parallel(1)
$P \parallel Q = Q \parallel P$	Parallel(2)
$(P \parallel Q) \parallel R = P \parallel (Q \parallel R)$	Parallel(3)
$P[\emptyset] = P$	Nesting(1)
$R[P] + R[Q] = R[P + Q]$	Nesting(2)
$P \parallel (Q + R) = (P \parallel Q) + (P \parallel R)$	Distributive(1)
$(a_1 + a_2).P = a_1.P + a_2.P$	Distributive(2)

표 4 δ-Calculus 의 대수 규칙

δ-Calculus 는 시각화를 위하여 그래프 언어를 제공한다. 표 5 는 ITL 에서 사용되는 아이콘들이다. 각각의 프로세스는 노드로 표현되며 채널은 노드들을 연결한 선으로 표현된다. 노드는 다른 노드를 포함할 수 있다. 모든 이동은 선으로 이동 방향이 표현되며, 이동의 결과 노드의 위치가 변경되어 ITL 에 반영된

다.

○	Process	◌	Hide Process
→	Movement	———	Channel

표 5 ITL 의 아이콘

표 6 은 ITS 에서 사용되는 아이콘들이다. 각각의 액션들이 노드로 표현되며, 액션들의 순서는 선으로 표현된다. 프로세스에서 실행되는 모든 액션과 실행 순서가 표현된다. 그러나 이동의 결과는 ITS 에서는 표현되지 않는다.

□	Process Lane	◎	Start
○	Other Process	◉	End
◇+	Choice	◇‖	Parallel
◎	New	◉	End
⊗	Kill	▷	Send
◺	Receive	▭	Empty
✉	InRequest	✉	InRequest Priority
⩛	InPermission	⌂	OutRequest
⌂	OutRequest Priority	⩛	OutPermission
✉	GetRequest	✉	GetRequest Priority
⩛	GetPermission	⌂	PutRequest
⌂	PutRequest Priority	⩛	PutPermission
→	Sequence	⇢	Exception

표 6 ITS 의 아이콘

그림 2 는 ITL 에 대한 예제이다. 예제에서는 P, B, C, R1, R2 의 다섯 개의 프로세스가 있으며 P 는 R1 과 R2 를 포함하고 있다.

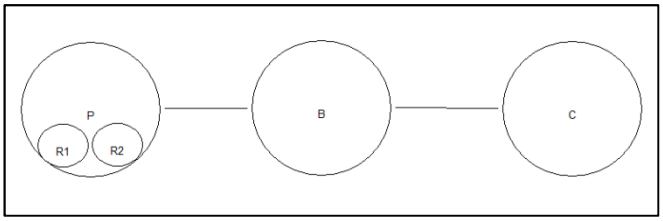


그림 2 ITL 의 예제

그림 3 은 ITS 에 대한 예제이다. 각각의 프로세스는 Process Lane 을 사용하여 구분되며, 각 프로세스가 실행할 액션들이 그 안에 표현되어 있다.

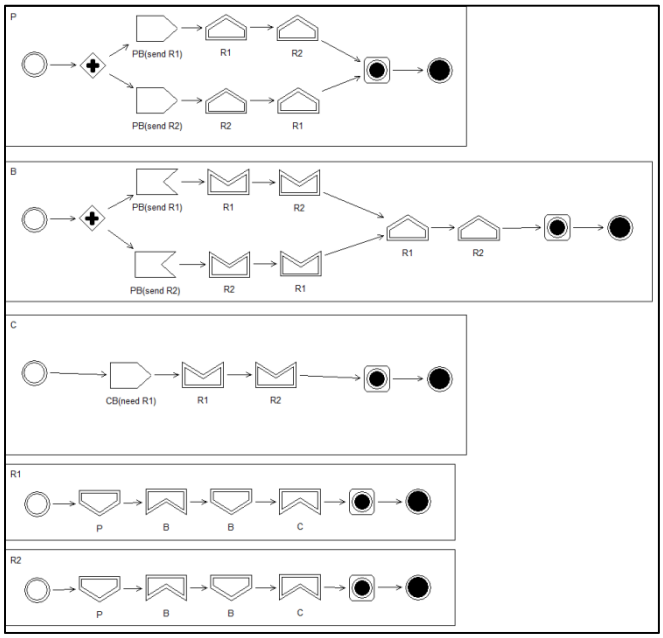


그림 3 ITS 의 예제

3.2 GTS Logic

A. GTS(Geo-Temporal Space)

GTS 는 프로세스의 공간 및 시간 정보를 그림 4 와 같이 표현한 것이다. δ-Calculus 와 유사한 방식으로 시스템 블록은 프로세스 블록들로, 프로세스 블록은 액션 블록들로 구성된다. 프로세스의 정의에 의해 프로세스 블록은 다른 프로세스 블록을 포함할 수 있다. 동기화를 위한 정의에 따라 서로 상호작용 하는 두 액션은 일정 시간에 대하여 서로 겹쳐있어야만 한다. 각각의 액션 블록은 준비 시간, 실행 시간 등과 같은 시간 속성을 지니고 있다.

그림 4 는 GTS 블록 다이어그램의 개념적인 그림이며, 실제 SAVE 에서의 다이어그램은 그림 9 에서 표현된다.

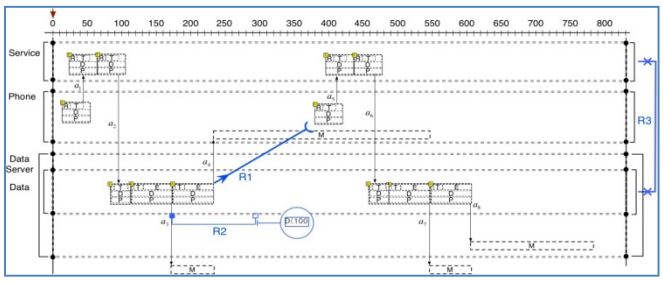


그림 4 GTS 블록 다이어그램의 예

B. GTS Logic

GTS Logic 은 시스템의 요구사항을 명세하고 검증하기 위한 1 차 논리를 시각화 한 언어이다. 시스템의 요구사항은 프로세스, 액션, 상호작용 블록간의 연관 관계로 표현되며, 요구사항이 시각화되어 GTS 블록 다이어그램에 표현된다. 표 7 은 GTS Logic 의 시각화 문법 및 의미를 나타낸다. 표의 A, B 는 프로세스를 나타내며 a, b 는 액션 또는 상호작용을 의미한다. 그림 4 에는 R1, R2, R3 라는 세 가지의 요구사항이 표현되어 있다.

	Syntax	Semantics
Geo-graphical Req's	A ○—○ B	A can be in B. B can be in A.
	A —×— B	A can be in B. B cannot be in A.
	A ○—× B	B can be in A. A cannot be in B.
	A —×—× B	A cannot be in B. B cannot be in A.
	A ——< B	B must be in A, always.
	A >—— B	A must be in B, always.
Temporal Req's	*a* →——(*b*	Action *a* must be performed before *b* in time.
	a)——→ *b*	Action *b* must be performed after *a*.
	a)——(*b*	Actions *a* and *b* can be performed concurrently in the same time period.
	a →→——←← *b*	Actions *a* and *b* can not be performed concurrently in the same time period.
Interval	⊓	Open interval
	⊓	Close interval
Conditions	—○ c	Condition for requirements: time, frequency, behavior, etc.

표 7 GTS Logic 의 문법

4. 기술 설명

SAVE 의 이중 접근 방법은 ADOxx 를 사용하여 개발되었다. ADOxx 는 비엔나 대학의 OMiLAB 에서 개발된 메타-모델링 플랫폼으로서 이를 기반으로 다수의 모델링 언어들[13, 14, 15, 16]이 개발되었다.

ADOxx 의 시각화를 위한 기술들은 다음과 같다.

- 메타-에디터: ADOxx 는 그래픽 정의 언어를 제공한다. 이 언어를 사용하여 모델링 클래스의 그래픽 표기 방식을 정의할 수 있다. 정의된 모델링 클래스의 표기 방식은 실제 모델링 도구에서 정의한 그대로 표기된다.
- 저장소: ADOxx 는 DB 를 사용하며, DB 에 접근하기 위한 기능을 제공하는 ADOxx 라이브러리를 제공한다. 모델은 모델링 도구를 사용하여 구성되며, 모델을 활용하는 여러 기능들은 라이브러리를 통해 저장소와 상호작용 함으로서 구현된다.
- 질의 시스템: 구성된 모델에 대하여 AQL 질의 언어를 사용하여 질의를 수행할 수 있다. AQL 은 ADOxx 에서 정의한 질의 언어이다.

5. 개발

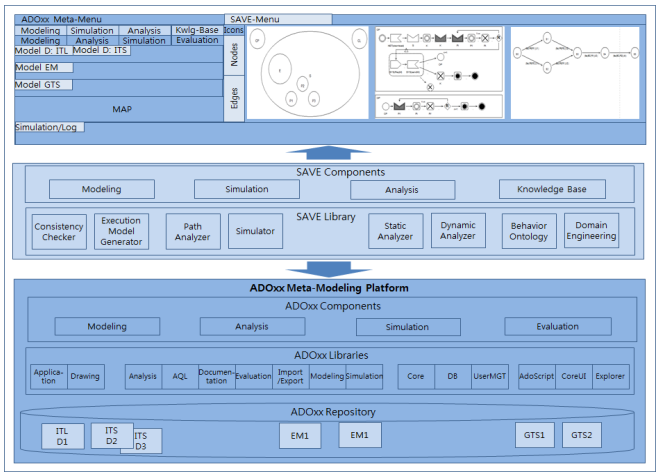


그림 5 SAVE 의 구조

그림 5 는 SAVE 의 구조를 나타낸다. SAVE 는 모델러, 시뮬레이터, 분석기, 검증기의 네 가지 기본적인 구성요소로 구성되어 있다. 각각의 구성요소는 ITL, ITS, EM, GTS 다이어그램을 기반으로 하고 있다. 각각의 구성요소에 대한 역할은 다음과 같다.

- 모델러: 모델러는 분산 이동 실시간 시스템을 명세하기 위한 기능을 제공한다. 모델러를 사용하여 ITL 을 생성하고, 각각의 프로세스에 대한 ITS 를 생설할 수 있다.
- EM 생성기: 모델러를 사용하여 명세한 ITL 과 ITS 를 기반으로 시스템의 모든 실행 경로를 표현하는 EM 그래프를 생성한다. 생성된 EM 그래프에서 하나의 경로를 선택하여 시뮬레이션을 할 수 있다.
- 시뮬레이터: EM 그래프에서 선택한 경로를 기반으로 시뮬레이션을 진행한다. 시뮬레이션의 결과는 자동적으로 GTS 블록 다이어그램으로 표현된다. 생성된 GTS 블록 다이

어그램은 분석 및 검증을 위해 사용된다.

- 분석기: 시스템의 여러 가지 특성을 분석하기 위한 질의 시스템을 제공한다.
- 검증기: GTS Logic 을 사용하여 시스템의 요구사항을 시각적으로 표현하고, 이를 검증하기 위한 기능을 제공한다.

SAVE 의 그래프 표현은 ADOxx 를 사용하여 개발되었다. 각각의 구성요소에서 제공되는 여러 기능들은 AdoScript 언어를 사용하여 개발되었다.

ADOxx 는 세 계층의 메커니즘을 제공한다.

- 1 계층: 모델링 도구의 기본적인 기능을 구현하기 위하여 ADOxx 에서 기본적인 기능들을 제공한다.
- 2 계층: AdoScript 를 사용하기 위한 400 여종의 API 를 제공한다. 이 API 를 사용하여 객체의 생성, 속성 값의 변경 등의 작업을 수행할 수 있다.
- 3 계층: ADOxx 의 외부와 상호작용을 하기 위한 기능들을 제공한다. 가장 기본적인 방법은 XML 파일을 사용하는 것이다.

SAVE 는 시각화를 위한 모델을 구현하기 위하여 1 계층의 기능을 사용하였다. 또한 EM 생성기, 시뮬레이터, 분석기, 검증기의 여러 기능을 구현하기 위하여 2 계층의 기능을 사용하였다.

현재 모델러와 EM 생성기, 시뮬레이터는 구현되었으며 분석기와 검증기는 개발중인 상태이다.

6. 예제

예제를 사용하여 SAVE 를 사용한 이중 접근방법에 대하여 설명한다. PBC(Producer-Buffer-Consumer)는 생산자와 소비자 간의 자원 전달이 발생하는 예제이다. 이 예제에는 P, B, C 의 세 프로세스 간에 자원 R1, R2 를 전달하는 형태로 구성된다. 그림 6 은 PBC 예제를 텍스트로 표현한 형태이다.

$$
\begin{aligned}
&PBC ::= P[R1 \parallel R2]\langle PB\rangle \parallel B\langle PB, BC\rangle \parallel C\langle BC\rangle \\
&P ::= ((PB(\overline{sendR\ 1}).put\ R1.put\ R2) \\
&\qquad +(PB(\overline{sendR\ 2}).put\ R2.put\ R1)).ext \\
&B ::= ((PB(sendR\ 1).get\ R1.get\ R2) \\
&\qquad +(PB(sendR\ 2).get\ R2.get\ R1)) \\
&\qquad .put\ R1.put\ R2.ext \\
&C ::= get\ R1.get\ R2.ext
\end{aligned}
$$

그림 6 PBC 예제의 명세

이러한 텍스트 명세를 기반으로 SAVE 에서는 시각화 모델을 생성한다. 그림 7 은 PBC 예제에 대한 ITL 을 생성한 것이다.

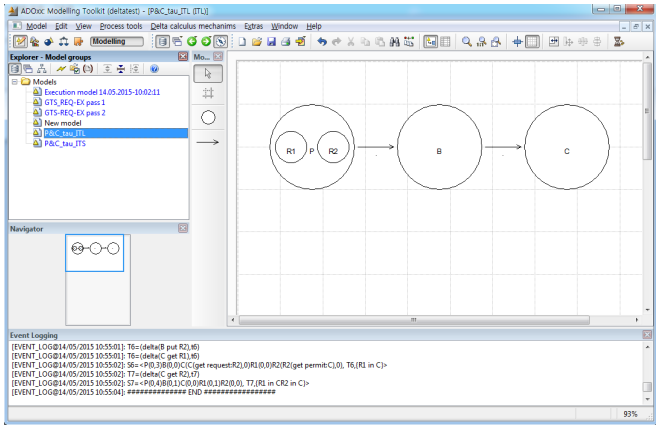


그림 7 PBC 예제의 ITL 표현

ITL 에서는 P, B, C 의 세 프로세스와, P 안에 R1, R2 가 포함되어 있음이 표현되어 있다.

프로세스 P 와 B 에는 Choice 연산에 따른 분기가 존재하며, 선택에 따라 다른 실행 결과가 발생한다. 두 프로세스에서 적절한 선택이 이루어지면 동기화에 의해 정상적으로 실행이 되지만, 선택이 잘못된다면 P 와 B 가 서로 동기화가 되지 않아 시스템이 정지하게 된다. 이러한 상황을 분석하기 위하여 EM 생성기를 통해 EM 그래프를 생성한다. 그림 8 은 PBC 예제의 ITS 와 EM 그래프를 나타낸다.

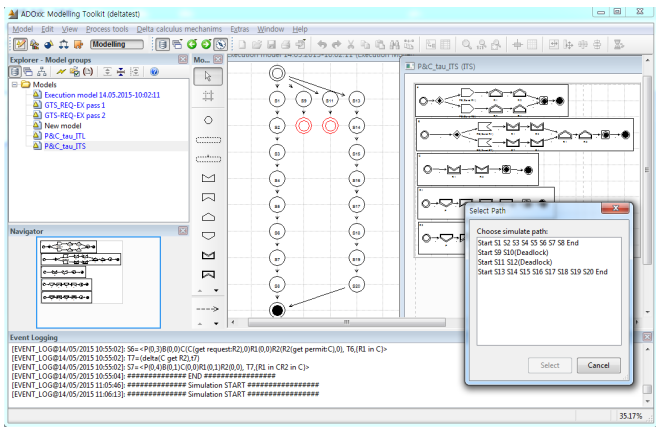


그림 8 PBC 예제의 ITS 와 EM 그래프

그림 8 을 살펴보면 총 네 가지의 경로가 존재함을 알 수 있다. 그 중 첫 번째와 네 번째의 경로는 정상적으로 작동하며, 두 번째와 세 번째는 비정상적인 작동을 하여 정지했음을 나타낸다. 안전한 시스템을 위해서는 모든 경로에 대하여 정상적으로 작동하도록 명세해야 한다. 이와 같은 분석을 통해 시스템의 기능적 요구사항을 만족하고 있는지의 여부를 파악할 수 있다.

그림 8 에서 표현된 네 가지의 경로 중 하나를 선택하여 시뮬레이션을 수행할 수 있다. 그림 9 는 첫 번째 경로에 대한 시뮬레이션을 수행한 뒤 그 결과를 GTS 블록 다이어그램으로 표현한 것이다.

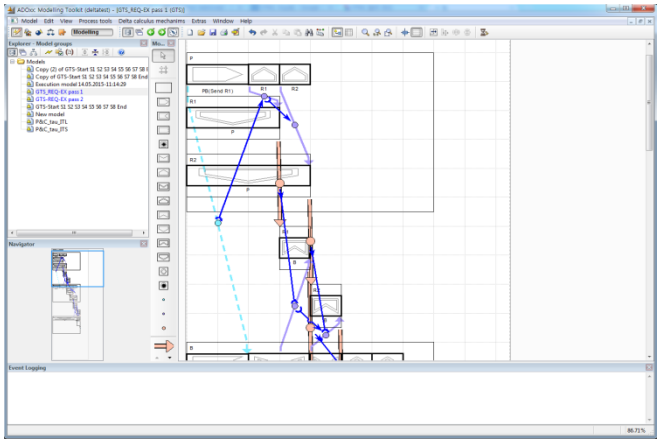


그림 9 PBC 예제의 GTS 블록 다이어그램과 GTS Logic 의 표현

어느 경로를 선택하던 P 는 제일 먼저 B 에게 메시지를 보낸다. 첫 번째 경로에서는 P 는 R1 을 먼저 보낸다는 메시지를 전송 후 R1 을 B 로 보낸다. 그 후에는 R2 를 B 로 보내는 행동을 수행한다. P 가 R1 와 R2 를 B 로 모두 전달하는 데에는 총 다섯 가지의 동기 연산이 사용된다. 이 다섯 가지의 행동을 표현하면 다음과 같다.

- τ_1: P 가 B 에게 메시지를 보낸다.
- δ_1: P 가 R1 을 밖으로 내보낸다.
- δ_2: P 가 R2 를 밖으로 내보낸다.
- δ_3: B 가 R1 을 가져온다.
- δ_4: B 가 R2 를 가져온다.

P 가 R1 을 보내기 전 먼저 메시지를 보내는 행동을 수행하므로 이에 대하여 정의되는 요구사항은 다음과 같다.

$$\tau_1 < \delta_1$$

a<b 관계는 a 가 b 보다 먼저 수행되어야 함을 의미한다.

첫 번째 경로에서는 P 가 R1 을 보낸 뒤 R2 를 보내야 하므로 이에 대하여 정의되는 요구사항은 다음과 같다.

$$\delta_1 < \delta_2$$

B 가 R1 과 R2 를 가져오기 위해서는 먼저 P 가 R1, R2 를 밖으로 내보내야 한다. 이에 대하여 정의되는 요구사항은 다음과 같다.

$$\delta_1 < \delta_3$$
$$\delta_2 < \delta_4$$

이러한 네 가지 요구사항에 대한 명세는 그림 9 에 표현되어 있으며, 이를 검증 한 결과 모두 만족됨이 확인되어 요구사항이 파란색으로 변경되어 표현되어 있음을 확인할 수 있다. 만약 요구사항을 만족하지 못하는 경우에는 붉은색으로 변경된다.

7. 결론

본 논문에서는 ADOxx 를 기반으로 개발된 SAVE 를 사용하여 시스템을 명세하고 검증하는 이중 접근 방법을 제안하였다. 이중 접근 방법은 δ-Calculus 를 사용한 명세와 GTS Logic 을 사용한 요구사항의 검증으로 구성된다.

현재 SAVE 는 모델러, EM 생성기, 시뮬레이터가 구현되어 있으며, 분석기와 검증기는 일부만 구현되어 있는 상태이다.

향후 연구에서는 분석기와 검증기의 구현, EM 그래프에서 발생될 수 있는 상태 증폭 문제의 해결을 수행할 것이다. 상태 증폭 문제는 syntactic sugar 를 사용하여 부분적으로 해결[17]된 상태이며, 루프나 재귀와 같은 복잡한 구성에서의 증폭 문제를 추가적으로 다룰 예정이다.

Acknowledgement

이 논문은 2010 년도 정부(미래창조과학부)의 재원으로 한국연구재단의 거대과학연구개발사업(NRF-2014M1A3A3A02034792) 및 정부(교육부)의 재원으로 한국연구재단의 이공분야기초연구사업(NRF-2015R1D1A3A01019282)과 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성 지원사업 (IITP-2015-H8501-15-1012)의 연구결과로 작성되었음.

참고문헌

[1] Haxthausen AE, Peleska Jan. Formal development and verification of a distributed railway control system. *Software Engineering IEEE Transactions on Vol. 26, No. 8*. August. 2000. pp. 687-701.

[2] Fill H, Karagiannis D. On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform. *Enterprise Modelling and Information Systems Architectures Vol. 8, No. 1*. Mar. 2013. pp. 4-25

[3] R Milner. *Communication and Concurrency*. 1989. Prentice hall New York, etc.

[4] R Milner. *Communicating and mobile systems: the pi calculus*. 1999. Cambridge university press.

[5] Luca Cardelli, Andrew D Gordon. Mobile ambients. *Foundations of Software Science and Computation Structures*. 1998. Springer Berlin Heidelberg. pp. 140-155.

[6] R Alur, TA Henzinger. Real-time logics: complexity and expressiveness. *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*. 1993. Academic Press. pp. 35–77.

[7] JS Ostroff. *Temporal logic for real-time systems*. 1989. John Wiley & Sons New York.

[8] Luís Caires, Luca Cardelli. A spatial logic for concurrency (part I). *Theoretical Aspects of Computer Software*. 2001. Springer Berlin

Heidelberg. pp. 1-37.

[9] Masato Notomi, Tadao Murata. Hierarchical reachability graph of bounded Petri nets for concurrent-software analysis. *Software Engineering IEEE Transactions Vol. 20, No. 5*. May. 1994. pp. 325-336.

[10] Dave Steinberg, et al. *EMF: eclipse modeling framework*. 2008. Pearson Education

[11] Stefan Berger, at al. Metamodel-Based Information Integration at Industrial Scale. *Model Driven Engineering Languages and Systems part 2*. 2010. Springer Berlin Heidelberg. pp. 153-167

[12] Ditigal Equipment Corpoeration. *DECDesign: User's Guide*. DEC, Maynard, MA, Part No. AA-PABRE-TE, Octber 1991.

[13] ADOxx. ADOxx Realization Case: BPMN. Last Access: http://www.adoxx.org/live/bpmn, 26-06-2015

[14] ADOxx. ADOxx Realization Case: UML. Last Access: http://www.adoxx.org/live/uml, 26-06-2015

[15] ADOxx. ADOxx Realization Case: OWL. Last Access: http://www.adoxx.org/live/owl, 26-06-2015

[16] ADOxx. ADOxx Realization Case: ER. Last Access: http://www.adoxx.org/live/er, 26-06-2015

[17] Woorim Choi, Yeongbok Choe, Moonkun Lee. A Reduction Method for Process and System Complexity with Conjunctive and Complement Choices in a Process Algebra, 39th Annual IEEE Computer Software and Applications Conference Workshops. 2015.

위치 기반 서비스에서 이동 경로의 정형 명세 및 모니터링

권혁, 정선일, 권기현

경기대학교 컴퓨터과학과
경기도 수원시 영통구 광교산로 154-42
{s191819, greffsozpkk, khkwon}@kyonggi.ac.kr

요약: 모형 검사(model checking) 기법은 모형의 모든 행위가 속성을 만족하는지를 결정한다. 모형의 모든 행위를 조사하기 때문에 무거운 검증 기법에 속한다. 한편, 실행 시간 검증(runtime verification)은 하나의 행위에 대해서만 속성 만족 여부를 결정한다. 모든 행위가 아니라 하나의 행위만을 다루기 때문에 가벼운 검증 기법이라고 할 수 있다.

본 논문에서는 위치 기반 서비스에서 핵심인 이동 경로의 명세와 모니터링에 실행 시간 검증 기법을 적용한다. 이동 경로는 GPS 신호로 구성되며, 속성 명세는 시간흐름뿐만 아니라 신호 값의 세기를 표현하는 신호 시제 논리를 사용한다. 신호 시제 논리로 이동 경로의 속성을 명세하기때문에, 현재 위치만을 고려하는 기존의 위치 기반 시스템보다 더 우수한 서비스를 제공할 수 있다.

핵심어: 시제 논리, 정형 명세, 정형 검증, 모니터링.

1. 서론

IT 융합 시대에서 정확하게 동작하는 소프트웨어의 중요성은 아무리 강조해도 지나치지 않다. 왜냐하면 소프트웨어의 오 동작으로 인한 피해 규모가 갈수록 증가하기 때문이다. 이론적으로 보자면, 정확도를 높이기 위해서는 시스템의 일 부분 보다는 시스템 전체를 전수 조사해야 한다. 비록 시스템을 추상화 한 모형을 다루지만, 모형 검사(model checking)는 모형의 모든 행위를 전수 조사한다[1]. 모형 검사란 "시스템의 추상 모형 $\mathcal{M}$과 속성 φ가 주어졌을 때, $\mathcal{M} \vDash \varphi$ 여부를 결정하는 기법" 이다. 이를 결정하기 위해서 모형의 행위를 전부 조사하기 때문에, 상태 폭발 문제 같은 위협에 시달린다. 그래서 모형 검사는 개념적으로는 우수함에도 불구하고, 산업체에서 그다지 활발히 사용되지는 못하고 있다.

"본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성 지원사업의 연구결과로 수행되었음" (IITP-2015-R0992-15-1014).

한편, 실행 시간 검증(runtime verification)이란 "하나의 행위 w와 속성 φ에 대해서 $w \vDash \varphi$ 여부를 검사하는" 기법이다[2]. 모형 검사는 모든 행위를 조사하는 기법인데 반해서, 실행 시간 검증은 한 행위만을 조사하기 때문에 모형 검사 보다는 가벼운 기법이다. 본 논문에서는 여러 이동 경로 중 하나의 경로(행위)만을 대상으로 실행 시간 검증을 이용해서 위치 기반 서비스에서 필수적인 이동 경로를 명세하고 모니터링한다. 이동 경로는 GPS 수신기의 위도 신호와 경도 신호로 얻는다. 이동 경로에 대한 명세는 시간흐름뿐만 아니라 연속 신호 값을 나타낼 수 있는 신호 시제 논리로 기술한다. 그런 후, 시제 모니터링을 통해서 연속 신호로 구성된 이동 경로가 명세를 만족하는지를 모니터링한다.

본 논문의 기여는 다음과 같다: 1) 현재 상용화된 위치 기반 서비스를 넘어서는 서비스를 제공함으로써, 어린 자녀, 치매 노인 등의 사회적 약자를 보호하고, 추후 전과자 및 죄수 등의 사회적 흉악범을 감시하여 국민의 안정성을 도모하고자 한다. 2) 정형 명세 시각화 도구로 쉽고도 빠르게 서비스를 명세할 수 있다. 3) 개발된 GPS 모니터링 시스템뿐만 아니라 다양한 신호등으로 확장 가능하다.

본 논문의 구성은 다음과 같다. 먼저 2 장에서는 관련 연구로 위치 기반 서비스의 정의와 기존에 제공되고 있는 위치 기반 서비스와 신호 시제 논리에 대해 설명하도록 한다. 3 장에서는 논문에서 제안하는 모니터링 시스템의 구성도를 설명하고, 실험과 결과 해석을 보이도록 한다. 4 장에서는 결론 및 향후 연구로 마무리를 한다.

2. 관련 연구

현재 다양한 위치 기반 서비스들이 GPS 신호를 통하여 제공되고 있다. 본 논문에서는 신호시제논리를 통하여 확장된 위치기반서비스를 명세하고, 이를 모니터링 하도록 한다. 따라서 본 논문을 이해하기 위해 기존에 제공되는 위치기반서비스를 조사하고, 신호시제논리에 대해 설명하고 이를 자동으로 판별하는 모니터링 시스템을 설명한다.

2.1 위치 기반 서비스

위치 기반 서비스는 휴대폰 속에 기지국이나 위성 항법장치(GPS)와 연결되는 칩을 부착해 위치추적 서비스, 공공안전 서비스, 위치기반정보 서비스 등 위치와 관련된 각종 정보를 제공하는 서비스를 일컫는다. 즉 유선·무선 통신망을 통해 얻은 위치정보를 바탕으로 여러 가지 서비스를 제공하는 것이 위치기반 서비스이다.[3] 위치 기반 서비스는 GPS 를 사용하는데 위성을 사용한 GPS 의 경우, 위성 신호의 특성으로 고층 건물이나 실내에서는 불확실한 정보를 제공한다. 본 논문에서는 GPS 신호의 정확성을 다루는 논문이 아니므로, GPS 신호는 정확하다고 가정하고 실험을 진행토록 한다.

현재 스마트 폰에 GPS 가 탑재 됨에 따라, 이동 통신사의 주도로 위치기반서비스는 모든 분야에 걸쳐 사용되고 있다. 특히, 어린아이들이나 노인, 애완동물 등을 대상으로 미아방지, 치매노인 및 애완동물 실종방지 등의 위치기반 서비스들이 많이 제공되고 있다. 일반적으로 위치 기반 서비스는 위치 정보를 제공하는 업체에 따라 분류할 수 있는데 이를 표로 요약하면 다음과 같다.[4,5,6]

Table 1 이동통신사별 위치 기반 서비스 기능

제공업체	기능
S 사	- 가족위치확인 - 이동경로확인 - 위험존 진/출입 파악 - 이동경로조회
L 사	- 자동위치받기 - 안심시간위치받기 - 발자취보기
K 사	- 안심지역설정 - 긴급호출 - 위치정보조회

이 외에도 위치 정보를 수집할 수 있는 스마트 장비들이 발전됨에 따라, 다양한 어플리케이션들이 개발되어, 서비스가 제공되고 있다.

2.2 신호 시제 논리

신호 시제 논리(Signal Temporal Logic, 줄여서 STL)는 선형 시제 논리(Linear Temporal Logic, 줄여서 LTL)에서 확장된 형태이다. 선형 시제 논리로는 연속적인 신호 값을 다룰 수 없다. 이벤트의 순서만을 중요시하기 때문이다. 또한, 이벤트가 일어나는 정확한 시점에 대해서도 생략한다. 예를 들어, “신호등이 노란불이면 차의 속도를 줄여야 한다” 라는 명제에 대해 선형 시제 논리는 얼마 시간 내에 속도를 줄여야 하는지, 속도를 얼마나 줄여야 하는지에 대한 정보는 무시한다. 따라서, 위와 같은 선형 시제 논리의 한계점을 극복하기 위해 나온 것이 신호 시제 논리이다. 신호 시제 논리는 신호를 다루기 위해 나온 시제논리로, 현재 다양한 신호의 속성을 명세 하는데 쓰이고 있다.

앞서 말했듯이, STL 식의 문법은 LTL 에서 확장된 형태를 보인다. STL 식의 문법은 다음과 같다.[7]

$$\varphi ::= \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid$$
$$\varphi_1 \mathrm{U}_{[t1,t2)} \varphi_2 \mid \Diamond_{[t1,t2)} \varphi \mid \Box_{[t1,t2)} \varphi$$

위 문법을 설명하면 다음과 같다. μ 는 관심이 있는 대상으로, LTL 과 가장 다른 점은 단순 명제가 아닌 술어라는 것이다. $\neg\varphi$ 는 논리식의 부정이다. $\varphi_1 \wedge \varphi_2$ 는 동시에 두 논리식이 참이 되는 경우에만 참을 만족한다. $\varphi_1 \vee \varphi_2$ 는 두 논리식 중 하나라도 참을 만족하면 참이 된다.

$\varphi_1 \mathrm{U}_{[t1,t2]} \varphi_2$ 는 논리식 φ_2이 참이 될 때까지(Until), 구간 t1, t2 사이에서 논리식 φ_1 는 항상 참이 되어야 한다. $\Diamond_{[t1,t2)}\varphi$는 언젠간(eventually) 구간 t1, t2 사이에서 논리식 φ 이 참이 되어야 한다. $\Box_{[t1,t2)}\varphi$ 는 항상(always) 구간 t1, t2 사이에서 논리식 φ이 참이 되어야 한다.

다음은 STL 식에 대한 만족도(Satisfaction)이다. STL 식의 만족도는 이진(Boolean Satisfaction)과 정량 만족도(Quantitative Satisfaction) 이 있다. 이진 만족도는 결과값을 참과 거짓으로 나타내고, 정량 만족도는 결과값을 실수 값으로 나타낸다. 이진 만족도는 정량 만족도에 포함될 수 있다. 정량 만족도의 결과 값이 양의 값을 가지면 이진 만족도는 참, 0 이나 음의 값을 가지면 이진 만족도는 거짓이 된다.

3. 본론

3.1 모니터링 시스템

본 논문에서 제안하는 모니터링 시스템의 구성도는 다음과 같다.

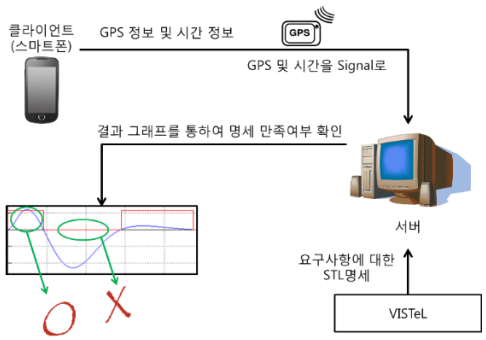


Figure 1 모니터링 시스템 구조

먼저, 모니터링 하고자하는 GPS 신호는 스마트폰을 통하여 받아와서 서버에 저장해둔다. 서버에 저장해 둔 GPS 를 바탕으로 VISTeL 이라는 모니터링 시스템을 통하여 요구사항을 STL 로 명세한다. 이때, 직접 STL 을 명세하기는 어려움이 있으므로, 시각화된 도구를 통하여 명세하도록 하였다. 시각화 도구의 형태는 다음과 같다.

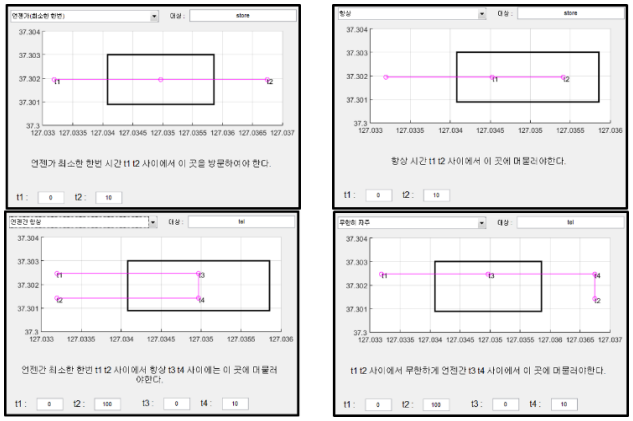


Figure 2 시스템에서 제공하는 명세 시각화 도구

위와 같은 시각화 도구를 통하여 명세를 한 뒤, 신호가 명세에 만족하는지를 자동으로 판별하는 알고리즘은 다음과 같다.

만족도의 관점에서 볼 때, 신호의 결과값이 0 보다 크면 1(TRUE)를 만족한다. 신호의 결과값은 명세에 따라서 달라지게 되므로, STL 에서의 각각 연산자의 의미를 예를 통해 설명하도록 한다.

먼저, 부정연산자는 신호 값의 반대 부호를 취해준다. 예를 들어, 신호 값이 0.5 였다고 한다면 부정연산자를 명세할 경우 -0.5 가 된다. 다음 논리곱, 논리합 연산자는 각각 신호 값에 대해 min, max 값을 취해준다. 예를 들어, 신호 값이 각각 0.3, 0.5 였다면, 이 둘의 논리곱은 min 값인 0.3 이 되고, 논리합은 max 값인 0.5 가 된다. 다음 ◇(Eventually) 연산자는 특정 구간에 대해 상한 값을 취하게 된다. 예를 들어, 특정 구간에서 (0.1, 0.2, 0.3, 0.4) 신호 값이었다면, 이 신호를 ◇(Eventually) 연산한 결과는 구간에서의 상한 값인 0.4 로 바뀌게 되어 (0.4, 0.4, 0.4, 0.4) 의 신호 값이 출력된다. □(Always) 연산자는 이와 반대로 하한 값을 취하게 되어, 위와 같은 신호에서 □(Always) 연산한 결과 값은 (0.1, 0.1, 0.1, 0.1) 의 신호 값이 출력된다. U(Until) 연산자는 $\varphi_1 U_{[t1,t2)} \varphi_2$ 일 때, 시간 t 부터 t+t2 사이에서 논리식 φ_1을 만족하는 실수 신호 값에서의 하한 값과, 시간 t + t1 과 t + t2 사이를 만족하는 논리식 φ_2의 실수 신호 값 중에서 min 값의 상한을 취한 값이 출력된다. 이를 간단한 수식을 바탕으로 표로 요약하면 다음과 같다.

Table 2 연산자에 따른 신호 값

연산자	신호 값이 S 일때
$\neg\varphi$	-S
$\varphi_1 \wedge \varphi_2$	최소값(S)
$\varphi_1 \vee \varphi_2$	최대값(S)
$\Diamond_{[t1,t2)}\varphi$	구간(t1,t2)내 S 의 상한(sup) 값
$\Box_{[t1,t2)}\varphi$	구간(t1,t2)내 S 의 하한(inf) 값
$\varphi_1 U_{[t1,t2)} \varphi_2$	구간(t,t+t2) 내 S 의 하한 값과 구간(t + t1, t + t2) 내 S 의 최소값의 상한 값

3.2 실험

본 실험에서는 신호 시제 논리를 통하여 모니터링 할 경우, 기존에 제공되던 미아방지나 치매노인 실종방지를 위한 위치기반 서비스등의 기본 기능을 제공할 뿐만 아니라 추가적인 기능을 기대할 수 있음을 보이도록 한다.

3.2.1 실험 데이터

본 실험에 쓰인 실험 데이터는 GPS 를 서버에 수집하는 간단한 어플리케이션을 가지고, 경기대학교를 직접 다니면서 수집한 GPS 데이터이다. 본 데이터를 통하여 실험자가 신호 시제 논리 명세에 맞는 경로로 움직였는지를 모니터링 해본다. 본 실험에서는 GPS 의 정확성을 다루지는 않는다. 따라서 측정한 GPS 값 중에서 특이값(너무 갑자기 튀는 값)에 대해서는 직접데이터를 수정한 뒤에 진행하였다.

3.2.2 실험 시나리오

실험 시나리오를 통해 보이고자 하는 것은 총 3 가지이다. 먼저 첫 번째는 기존 다른 이동통신사에서 제공하는 것과 동일한 기능을 보이는 것이다. 항상 안전지역을 방문해야 하는 것과, 특정지역을 언젠간 한번이라도 방문해야 하는 시나리오를 통하여 기존 위치 기반 서비스와 동일한 기능을 보이도록 한다.

다음으로는 신호 시제 논리를 활용하였을 때, 제공되는 확장된 위치 기반 서비스들을 보이도록 한다. 두 번째 시나리오는 기존 위치 기반 서비스보다 더 유연한 서비스를 보이고자 한다. 기존 위치 기반 시스템에서는 위험지역을 한번이라도 지나면 보호자에게 알림이 전송되었다. 하지만 특정상황에서는 한번만 지난다고 해서 무조건 위험한 상황이 아닐 수도 있다. 예를 들어, 집을 빨리 가기 위해서 위험지역인 PC 방앞을 우연히 한번 지나칠 수 있다. 하지만 실제로 이것보다는 그 지역에서 특정 시간 동안 머무르는 것이 더 위험한 상황이라고 판단할 수 있다. 신호

시제 논리를 사용하면 위와 같은 상황을 명세하여 서비스 할 수 있다.

또한, 신호 시제 논리를 활용하면 기존 위치 기반 서비스와는 달리 반복적인 행위도 명세할 수 있다. 예를 들어, 일정 시간 내에 지속적으로 반복 방문을 해야 하는 상황이라면, 기존 시스템에서는 현재 위치만을 기준으로 서비스를 제공하기 때문에 한 장소만을 방문했다는 것만 알 수 있다. 하지만 신호 시제 논리를 활용하여 명세하면, 등하교, 출퇴근 등과 같은 중복 방문을 명세할 수 있다.

3.2.3 실험 및 결과해석

먼저, 첫 번째 시나리오는 기존 다른 위치 기반 서비스와 동일한 기능을 보이는 시나리오다. 항상 특정 지역을 방문하거나 한번이라도 그 지역을 방문하여야 하는 시나리오이다. 먼저, 측정된 GPS 이동경로와 실제 명세 화면은 다음과 같다.

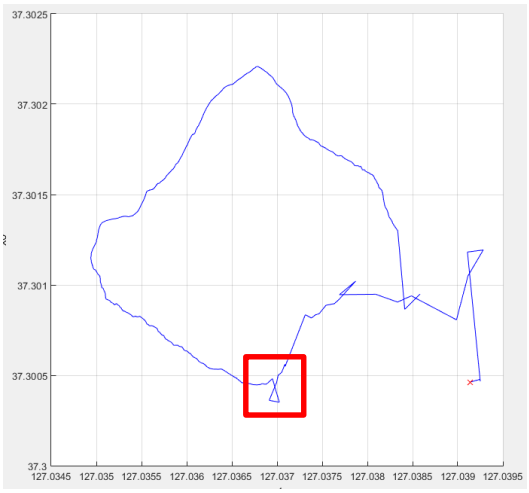


Figure 3 시나리오 1 의 이동경로

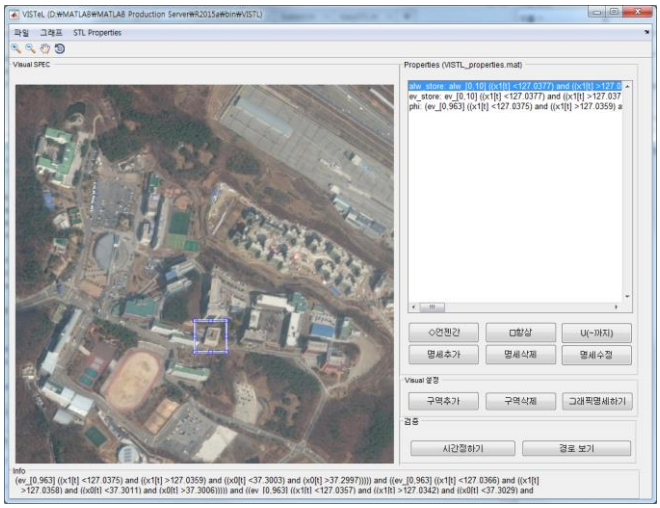


Figure 4 시나리오 1 의 실제 명세 화면

위에서 빨간 네모지역은 학교 편의점이다. 시나리오에 따라 명세를 항상 10 초동안 편의점을 방문하여야 한다와 최소한 한번 10 초안에 편의점을 방문하여야 한다를 시각화 도구를 통해 STL 명세한 뒤, 모니터링 시스템을 통하여 만족도를 자동으로 판정하면 다음과 같다.

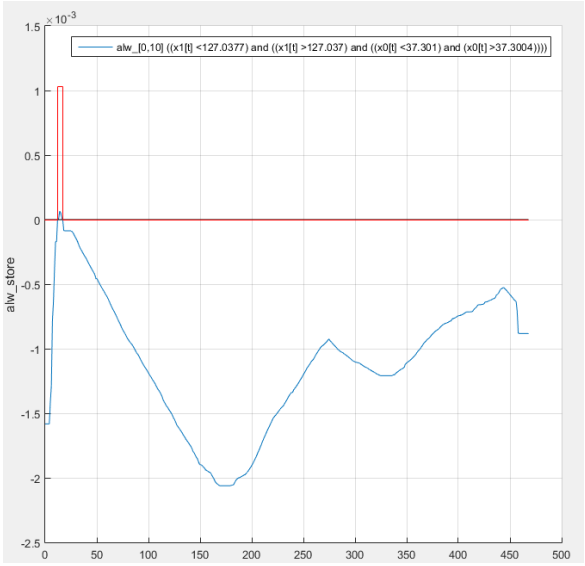


Figure 5 $\square_{[0,10]}$(**편의점**) 의 만족도 그래프

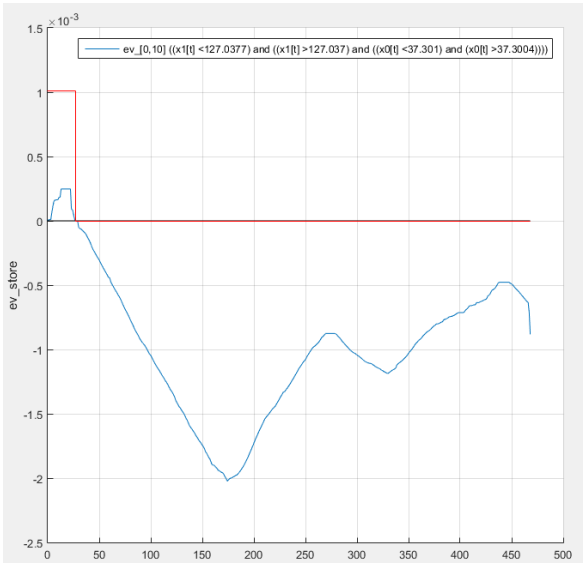


Figure 6 $\Diamond_{[0,10]}$(**편의점**)의 만족도 그래프

위 그래프를 해석하면 다음과 같다. 먼저 빨간색 그래프는 이진 만족도를 나타내고, 파란색 그래프는 정량 만족도를 나타낸다. 이해를 돕기 위해 빨간색 그래프를 기준으로 결과를 해석해보면 먼저 그림 5 에서 약 20 초 부근에서 항상 10 초동안 편의점에 가야 한다는 명세를 만족하므로 값이 1(TRUE)이 됨을

볼 수 있다. 마찬가지로 그림 6 에서도 시작부터 10 초안에 최소한 한번 편의점을 방문하므로, 명세를 만족하여 값이 1(TRUE)이 됨을 볼 수 있다. 그 이후로는 명세를 만족하지 않으므로 0(FALSE)임을 볼 수 있다.

다음은 신호 시제 논리를 통하여 명세를 하였을 때, 새롭게 적용 가능한 기능인 유연성이다. 기존 위치 기반 시스템에서는 한번이라도 위험지역을 지나면 알림을 울렸었다. 하지만 신호 시제 논리를 통하여 명세하였을 경우, 한번만 지나는 것이 아니라 몇 초 이상 머물러서는 안된다라고 명세를 할 수 있다. 먼저 시나리오 2 를 위해 측정된 이동경로와 실제 명세 화면은 다음과 같다.

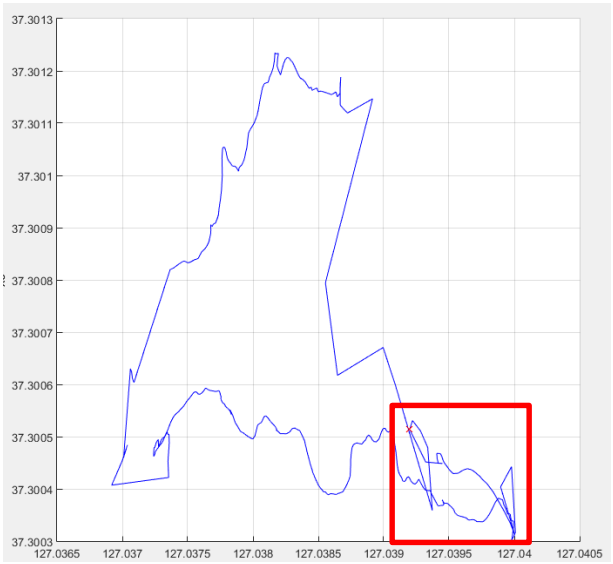


Figure 7 시나리오 2 의 이동경로

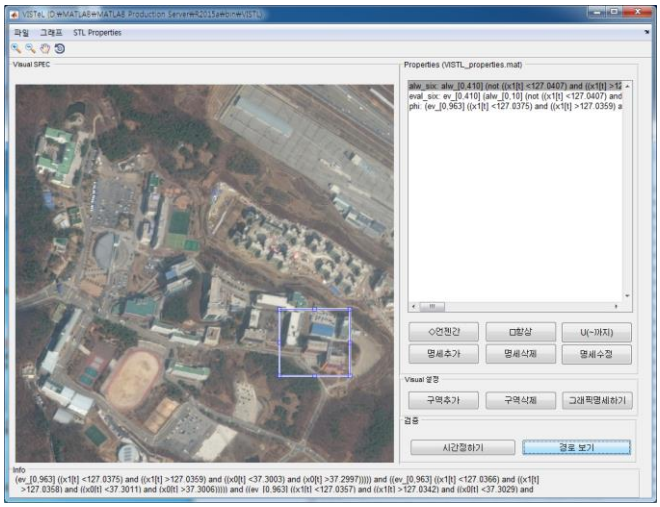


Figure 8 시나리오 2 의 실제 명세 화면

위에서 빨간색 네모 부분은 6 강의동이다. 기존 시스템에서는 6 강의동을 항상 지나면 안된다고 명세할 수 있었다. 하지만 신호 시제 논리를 활용하여 명세할 경우, 언젠간 항상 10 초동안 6 강의동에 머물러선 안된다라고 명세할 수 있다. 이를 시각화 도구를 통해 STL 명세한 뒤, 모니터링 시스템을 통하여 만족도를 자동으로 판정하면 다음과 같다.

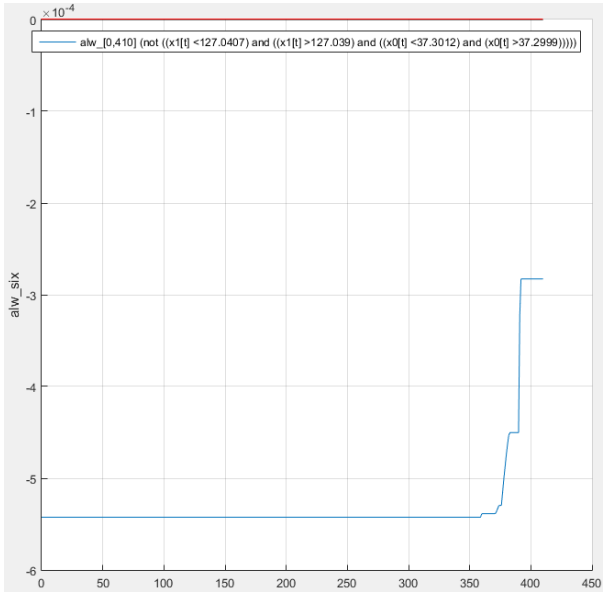


Figure 9 $\square_{[0,410]}\neg$(**6강의동**)의 만족도 그래프

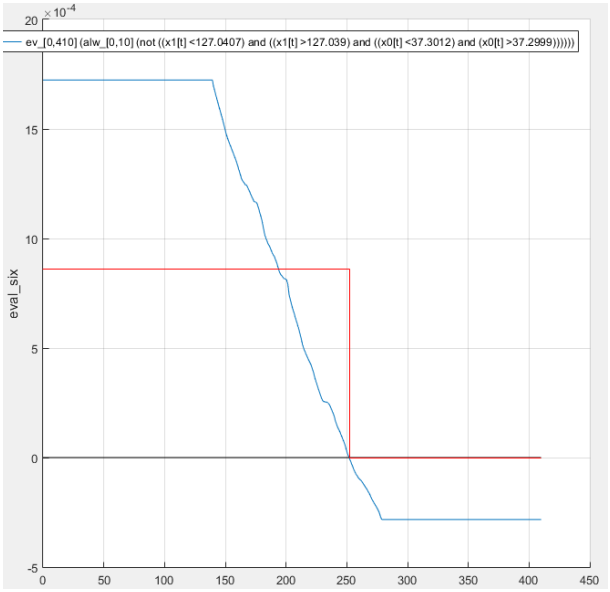


Figure 10 $\Diamond_{[0,410]}\square_{[0,10]}\neg$(**6강의동**)의 만족도 그래프

위 그래프를 해석하면 다음과 같다. 먼저 그림 9 에서 한번이라도 6 강의동을 방문하기 때문에, 명세를 만족하지 않아 0(FALSE)임을 볼 수 있다. 하지만 같은 이동 경로에 대해 그림 10 의 명세의 경우, 약 250 초까지 최소한 한번이라도 10 초이상 6 강의동에서 머무르지 않으므로, 명세를 만족하므로 값이 1(TRUE)이 됨을 볼 수 있다.

마지막으로 새롭게 적용 가능한 기능은 반복적인 방문을 찾는 명세이다. 기존 시스템에서는 현재 위치만을 기반으로 하기 때문에, 단일 방문만을 다뤘었다. 하지만 신호 시제 논리를 활용하여 명세를 하였을 경우, 반복적인 방문을 하는 경우를 다룰 수 있다. 먼저 시나리오 3 의 이동경로와 실제 명세 화면은 다음과 같다.

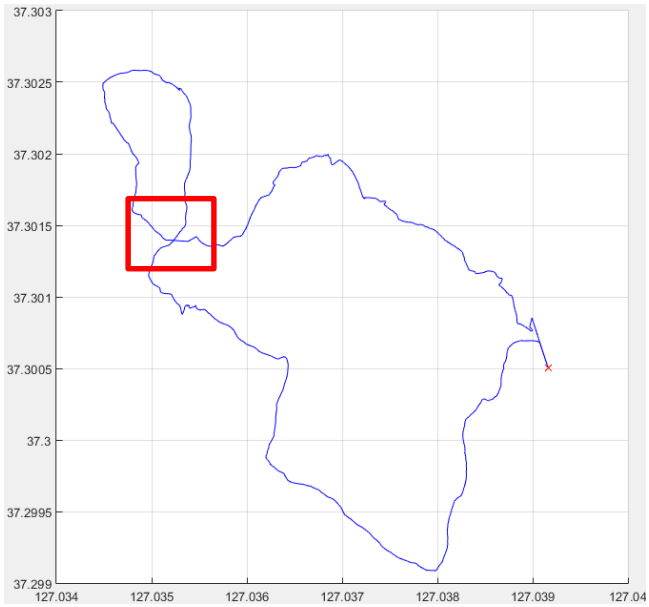


Figure 11 시나리오 3 의 이동경로

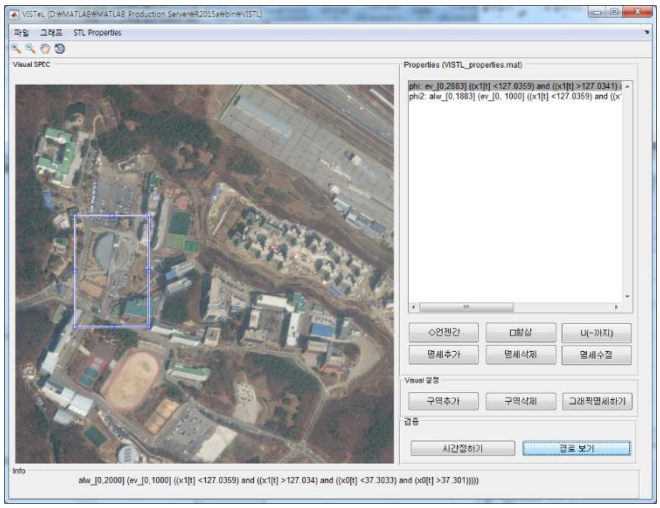


Figure 12 시나리오 3 의 실제 명세 화면

위에서 빨간색 네모 부분은 텔레컨벤션 센터이다. 기존 시스템에서는 최소한 한번 텔레컨벤션 센터를 방문해야 한다고 명세할 수 있었다. 하지만 신호 시제 논리를 활용하여 명세할 경우, 1883 초까지 무한하게 1000 초안에 최소한 한번 텔레컨벤션 센터를 방문해야 한다고 명세할 수 있다. 이를 시각화 도구를 통해 STL 명세한 뒤, 모니터링 시스템을 통하여 만족도를 자동으로 판정하면 다음과 같다.

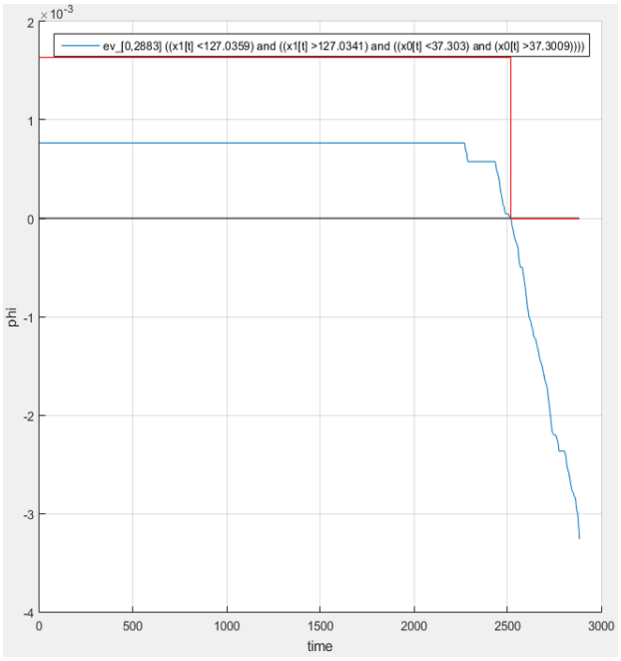


Figure 13 $\Diamond_{[0,2883]}$(**텔레컨벤션 센터**)의 만족도 그래프

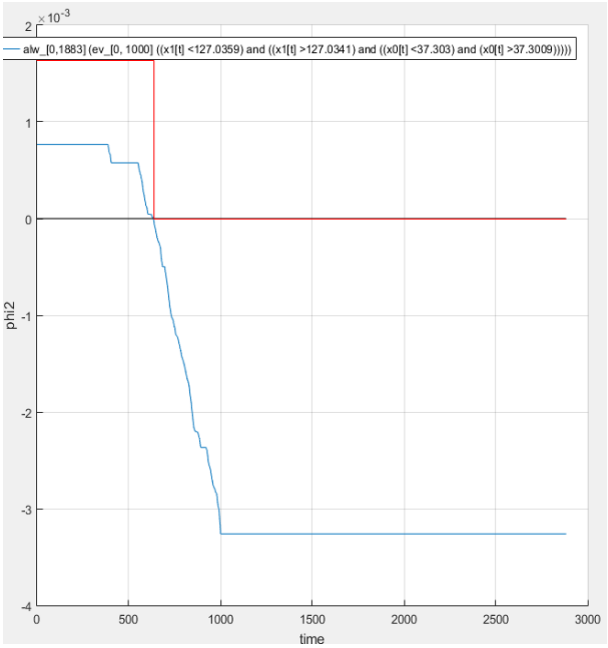


Figure 14 $\Box_{[0,1883]}\Diamond_{[0,1000]}$(**텔레컨벤션 센터**)의 만족도 그래프

위 그래프를 해석하면 다음과 같다. 먼저 그림 13 에서 약 2500 초까지 최소한 한번이라도 텔레컨벤션 센터를 방문하기 때문에, 명세를 만족하므로 값이 1(TRUE)이 됨을 볼 수 있다. 하지만 같은 이동 경로

에 대해 그림 14 의 명세의 경우, 1883 초까지 무한하게 1000 초안에 최소한 한번 텔레컨벤션 센터를 방문해야 한다는 명세를 약 700 초까지는 만족하므로 값이 1(TRUE)이 됨을 볼 수 있다. 즉, 그때까지는 텔레컨벤션 센터를 반복 방문하는 것이다. 하지만 그 이후부터는 단일 방문을 하여도 반복 방문은 하지 않으므로 0(FALSE)가 됨을 볼 수 있다.

4. 결론 및 향후 연구

본 논문에서는 실행 시간 검증을 이용해서 위치 기반 서비스에서 필수적인 이동 경로를 명세하고 이를 모니터링하였다. 이동 경로는 GPS 수신기의 위도 신호와 경도 신호로 얻었다. 이동 경로에 대한 명세는 시간흐름뿐만 아니라 연속 신호 값을 나타낼 수 있는 신호 시제 논리로 기술한다. 그런 후, 시제 모니터링을 통해서 연속 신호로 구성된 이동 경로가 명세를 만족하는지를 모니터링하였다. 신호 시제 논리를 통하여 이동경로를 명세하여 보이고자 하는 것은 다음과 같았다. 1) 현재 상용화된 위치 기반 서비스를 넘어서는 서비스를 제공함으로써, 어린 자녀, 치매 노인 등의 사회적 약자를 보호하고, 추후 전과자 및 죄수 등의 사회적 흉악범을 감시하여 국민의 안정성을 도모하고자 한다. 2) 정형 명세 시각화 도구로 쉽고도 빠르게 서비스를 명세할 수 있다. 3) 개발된 GPS 모니터링 시스템뿐만 아니라 다양한 신호등으로 확장 가능하다. 각각 보이고자 하는 바를 실험 시나리오를 통하여 보이고 입증하였다. 즉, 실험을 통해 위치 기반 서비스에서 신호 시제 논리를 통하여 이동 경로의 속성을 명세할 경우 기존의 위치 기반 서비스의 기능을 제공할 뿐만 아니라, 확장된 서비스를 제공할 수 있음을 보였다.

향후 연구로는 위치 기반 서비스에서 이동 경로의 속성을 신호 시제 논리를 사용하여 명세 하였을 경우 더 제공될 수 있는 서비스를 찾아볼 것이다. 또한 신호 시제 논리를 이동 경로의 속성을 나타내는 데만 국한되지않고, 더 나아가 다양한 신호에 적용하여 현재 쓰이는 서비스보다 더 나은 서비스를 제공하기 위해 노력할 것 이다.

참고문헌

[1] Christel Baier, Joost-Pieter Katoen, Principles of Model Checking, MIT, 2008

[2] Ezio Bartocci,“Runtime Verification”,MOVER2014

[3] 네이버지식백과,“위치기반서비스”,http://terms.naver.com/entry.nhn?docId=1232842&cid=40942&categoryId=32379, (2015-12-20)

[4] 네이버,“KT 올레”,https://ollehguard.olleh.com/olleh/main/main.asp, (2015-12-20)

[5] 네이버,“LG 유플러스”,http://www.uplus.co.kr/css/pord/cosv/cosv/RetrievePsMbSDmsgInfo.hpi?catgCd=51424&prodCdKey=C000000712, (2015-12-20)

[6] 네이버,“t 월드”,http://www.tworld.co.kr/normal.do?serviceId=S_PHOW0002&viewId=V_PHOW1001&cateCd=HD, (2015-12-20)

[7] Alexandre Donze, Oded Maler , Ezio Bartocci, Dejan Nickovic, Radu Grosu and Scott Smolka, "On Temporal Logic and Signal Processing", ATVA 2012

시스템 오브 시스템즈 목표 검증을 위한 시뮬레이션 사례 연구*

(Case study on Simulation for System of Systems Goal Verification)

서동원, 신동환, 박지훈, 지은경, 배두환

한국과학기술원 전산학과
대전광역시 유성구 대학로 291
{dwseo, donghwan, jhpark, ekjee, bae}@se.kaist.ac.kr

요약: 단일 시스템으로 대응할 수 없는 고차원적 목표를 달성하기 위하여 다수의 이종 단일 시스템을 연결하여 초대형 복잡 시스템을 구성하는 시스템 오브 시스템즈(SoS)에 대한 연구가 시작되고 있다. SoS 가 주어진 목표를 수행할 수 있는지 확인하기 위한 SoS 모델링 및 시뮬레이션 연구들이 제안되어왔으나, 해당 연구들의 SoS 예제는 SoS 를 설명하는 정의, 특성, 분류를 모두 포함하지 못해왔다. 본 연구에서는 SoS 의 정의, 특성, 타입을 포함하는 다중 손상 사고 SoS 예제를 제안하고, 목표 검증을 위한 시뮬레이션 모델을 해당 예제에 적용한다.

핵심어: 시스템 오브 시스템즈, 시뮬레이션, 이산 사건 시스템, 에이전트 기반 시뮬레이션, 시스템 다이내믹스

1. 서 론

사회, 경제, 산업 구조가 복잡해지고 고도화되면서 단일 시스템으로 대응하기 어려운 요구사항이 발생하고 있다. 예를 들어, 국가적 재난 상황이 발생했을 때 하나의 시스템이 해당 재난 상황을 모두 해결하는 것이 아니라, 여러 관련 시스템들의 긴밀한 공조를 통하여 문제를 해결해야 한다. 시스템 오브 시스템즈(System of Systems, 이하 SoS)는 이러한 문제를 해결하고자 제안되었으며[5], 전 세계적으로 활발히 연구가 시작되고 있다.

SoS 는 공통의 목표를 가진 다수의 요소 시스템(Constituent System, 이하 CS)의 집합으로 이루어지며, SoS 를 구성 및 실행하기 전에 모델링 및 시뮬레이션 과정을 통해 공통의 목표를 달성하는지 여부를 확인할 필요가 있다. 기존 연구들[1, 2]에서 SoS 모델링 및 시뮬레이션을 했지만, SoS 의 개념을 설명하는 많은 논문들의 SoS 에 대한 정의, 특성, 분류를 모델링 및 시뮬레이션에 포함하지 못해왔다.

본 논문에서는 SoS 의 개념을 설명하는 세 가지 방식인 정의, 특성, 타입으로 SoS 를 설명하고[3], 이 세 가지 방식의 설명에 부합되는 다중 손상 사고 대응 SoS 예제를 제안한다. 그리고 기존 연구에서 사용한 시뮬레이션 기법을 통해서 해당 SoS 예제를 모델링 한다.

본 논문의 구성은 다음과 같다. 2 장에서는 SoS 개념을 세 가지 분류에 따라 설명한다. 3 장에서는 SoS 목표를 검증하는 세 가지 시뮬레이션 모델을 소개하고, 4 장에서는 2 장의 분류에 적합한 다중 손상 사고 대응 SoS 예제를 제안하고, 세 가지 시뮬레이션 모델을 적용한다. 이어서 5 장에서는 결론 및 향후 연구에 대해 논의한다.

2. SoS 의 개념

"SoS 는 공통의 목표를 달성하기 위해 조직된 다수 시스템들의 집합이며[4], SoS 를 구성하는 각 CS 는 운용과 관리의 독립성을 갖는다[5]". SoS 를 정의하고 기존의 시스템과의 구별되는 점들을 분석한 많은 연구들이 있으며, 위와 같은 정의는 SoS 를 정의하는 연구들 간의 최소한의 공통 분모로 여겨진다[6]. 기존 연구들에서 SoS 를 설명하는 방법은 크게 세 가지로 나뉜다. SoS 를 정의를 통해 설명하는 방법, 기존의 대규모 시스템과의 차이점에 따라 설명하는 방법, SoS 를 운용, 관리하는 방법에 따른 SoS 타입에 따라 설명하는 방법이 있다. 본 장에서는 기존의 연구들에서 설명하는 방식에 따라 SoS 의 정의, 특징, 그리고 타입 순으로 SoS 의 개념을 설명한다.

2.1 SoS 의 정의

SoS 를 정의를 통해 설명하는 연구는 대표적으로 세 가지가 있다[4,5,7]. 첫 번째, Checkland 의 연구[4]에서는 SoS 를 독립적으로 정의된 다수의 시스템들이 공통의 목표를 수행하는 것으로써 정의한다. 두 번째, Maier 의 연구[5]에서는 운용의 독립성과 관리의 독립성을 가진 CS 들의 집합으로 SoS 를 정의한다. 세 번째, INCOSE 에서 발간한 책에 따르면[7] SoS 를 대규모 상호 협력 문제를 수반하는 복잡, 분산, 이기종 시스템으로 정의한다.

SoS 를 정의한 연구들은 두 지 공통적인 정의를 가지는데[6], 첫 번째는 SoS 를 하나의 공통적인 목표를 달성하기 위해 조직된 시스템의 집합으로 정의하는

* 이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No. R0126-15-1101, (SW 스타랩) 모델 기반의 초대형 복잡 시스템 분석 및 검증 SW 개발)

것이고, 두 번째는 CS 들의 운용 및 관리의 독립성의 여부에 따라 정의하는 것이다.

2.2 SoS 의 특성

Baldwin 의 연구[8]에서는 SoS 가 대규모 시스템과의 구별되는 SoS 의 다섯 가지 특성인 자율성(autotomy), 연대성(belonging), 연결성(connectivity), 다형성(diversity), 창발성(emergency)에 대해 설명한다. 이러한 특성 중 자율성, 연대성, 연결성은 CS 들이 주체가 되는 근원 특성이며 다형성, 창발성은 근원 특성으로부터 발현되는 SoS 가 주체인 결과 특성으로 분류 할 수 있다.

- **자율성:** 시스템의 각 부분(part)은 자율성을 갖지 않지만 SoS 의 CS 들은 자율성을 갖는다. 이는 각 CS 이 독립적으로 CS 의 목표를 달성하기 위해 자율적으로 운용, 관리됨을 의미한다.
- **연대성:** 시스템의 각 부분은 태생적으로 시스템에 소속 되어있지만 SoS 의 CS 들은 개별적인 목표 달성에 기반하여 자율적으로 SoS 에 소속된다.
- **연결성:** 시스템의 각 부분은 디자인에 따라 정적으로 연결 되어있지만, SoS 의 CS 들은 SoS 의 필요에 따라 동적으로 연결된다.
- **다형성:** 시스템은 설계 단계에서 형태가 정해지지만, SoS 는 목표 달성을 위해 CS 들의 연결성, 자율성, 연대성에 따라 다양한 형태를 가질 수 있다. 따라서 SoS 는 설계 단계에 형태를 예측 할 수 없으며 실제 운용되는 런타임에 이르러 예측이 가능하다.
- **창발성:** SoS 는 요소시스템들의 자율적인 행위에 따른 국부적 효과들의 중첩으로 창발되는 전역적 효과를 통해 목표를 달성한다. 따라서, SoS 가 갖고 있는 창발성이란 기존 시스템과 달리 설계 단계에서 예측 가능한 창발성이 아닌, 런타임에 발현되는 창발성을 의미한다.

2.3 SoS 의 타입

SoS 는 CS 들의 운용 및 관리의 방식에 따라 네 가지의 형태로 구분될 수 있다[9]. 아래의 절에서 네 가지 타입에 대해서 각각 설명한다.

2.3.1 Directed SoS

Directed SoS 는 SoS 수준의 소유자 및 관리자가 존재한다. SoS 수준의 소유자는, CS 들의 개발을 주관하는 주체로써 SoS 목표 달성을 위한 절대적인 권한을 갖게 된다. 따라서 이 타입의 SoS 수준 관리자는 CS 들의 구성, 연결, 협업에 대한 명령을 할 수 있는 권한을 갖는다. 하지만 각 CS 은 SoS 목표를 달성하기 위해서 명령을 받지 않는 상황에서는 개별적인 관리, 운용의 권한을 가지며, CS 의 개별 목표 달성을 위한 행위를 지속적으로 추구한다.

2.3.2 Acknowledged SoS

Acknowledged SoS 는 SoS 수준의 관리자는 존재하지만 소유자는 존재하지 않는다. 따라서 SoS 의 목표를 달성하기 위해 CS 들에게 강제적인 권한을 행사하지 못한다. SoS 목표를 달성 하기 위해서 CS 들에게 목표 달성을 위한 행위를 요청할 수 있지만, CS 들은 스스로 판단하여 개별 목표 반하지 않는 수준에서 SoS 목표를 위한 관리자의 요청을 수행한다.

2.3.3 Collaborative SoS

Collaborative SoS 는 자율적인 협업을 통해 공통의 목표를 달성하는 SoS 이다. 이 타입은 SoS 의 수준의 소유자도, SoS 수준의 관리자도 없다. 하지만 CS 들 간 동의된 공통의 목표가 있기 때문에, 목표를 달성하기 위해 CS 들이 자율적으로 협업한다.

2.3.4 Virtual SoS

Virtual SoS 는 Collaborative SoS 와 마찬가지로 SoS 수준의 소유자도, 관리자도 존재 하지 않는다. 뿐만 아니라 CS 들 간의 동의된 공통의 목표가 없기 때문에, SoS 목표 달성은 CS 들의 자율적인 행위에 따라 결정 된다. 따라서 Virtual SoS 는 공학적인 접근 방법이 적용되지 않는다.

이와 같은 네 가지 타입의 SoS 는 표 1 의 세 가지 기준에 따라 구분 될 수 있다.

표 1. SoS 의 타입

SoS 타입	SoS 수준 소유자	SoS 수준 관리자	SoS 목표
Directed	O	O: 명령	O
Acknowledged	X	O: 권고	O
Collaborative	X	X	O
Virtual	X	X	X

3. SoS 시뮬레이션 기반 검증

시스템 공학에서 검증을 위해서 사용되는 대표적인 시뮬레이션 기법으로는 이산 시간 시스템, 에이전트 기반 시뮬레이션, 시스템 다이내믹스가 있다. 기존의 [1, 2, 10]의 연구에서 SoS 예제 대상으로 시뮬레이션을 하기 위해 위의 기법들이 사용되었으며, 본 장에서는 각 기법에 대해 설명한다.

3.1 이산 시간 시스템

이산 시간 시스템(Discrete Event System, 이하 DEVS)는 프로세스를 모델링 하는 것을 목표로 한다. DEVS 는 atomic 모델과 coupled 모델로 나뉘며 atomic 모델에서는 하나의 시스템의 프로세스를 상세히 모델링 하며, coupled 모델에서는 여러 시스템간

의 상호 관계를 모델링 한다. 낮은 수준까지 모델링 할 수 있는 것이 DEVS 의 장점이다. DEVS 의 시뮬레이션 결과 변화 요인은 이벤트이다. 따라서 DEVS 는 CS 의 상세한 행위 및 CS 간 상호 작용 시뮬레이션에 적합한 기법이다[1].

3.2 에이전트 기반 시뮬레이션

에이전트 기반 시뮬레이션(Agent Based Simulation, 이하 ABS)는 개별 에이전트를 모델링 하는 것을 목표로 한다. 각 에이전트는 규칙을 가지고 있고, 규칙을 기반으로 자율적인 판단을 내리며, 다른 에이전트들과 상호 작용한다. 따라서 시뮬레이션 결과의 변화 요인은 에이전트의 규칙에 따른 결정 및 상호작용에 의존한다. ABS 는 자율성 및 사회적인 측면을 표현하기에 적합한 시뮬레이션 기법으로서 SoS 의 근원 특성인 자율성을 표현하고 검증에 적합하다[2].

3.3 시스템 다이내믹스

시스템 다이내믹스(System Dynamics, 이하 SD)는 관찰의 관점에서 시스템을 모델링 하여 내부에서 사용되는 자원들의 추세를 관찰할 수 있다. 상세한 프로세스는 모델링 하지 않기 때문에 추상적인 수준에서만 시뮬레이션한다. SD 는 자원을 저장할 수 있는 제한된 공간인 stock 과 자원을 시스템으로 보내는 전송률인 flow 로 구성되며, 반복적 실행을 통해 시간에 따른 자원의 사용률 추세를 관찰한다. 시뮬레이션 결과의 변화 요인은 자원의 사용률에 대한 피드백이다. SD 는 자원의 사용률에 대한 결과를 분석하고 검증하는데 적합한 기법이다[10].

기존 연구들[1, 2]에서는 SoS 를 모델링 하기 위해 한가지 기법 만을 사용하였으나, 본 논문의 사례연구에서는 Ross 의 연구[10] 방법과 같이 ABS, DEVS, SD 세 가지 기법을 함께 활용하여 SoS 를 다양한 측면에서 검증한다.

4. SoS 예제에 대한 사례 연구

4.1 구체적인 SoS 예제

SoS 검증 연구를 위해 다중 손상 사고 (Mass Casualty Incident, 이하 MCI) 대응 SoS 예제를 제안한다. MCI 는 의료 자원 대비 많은 환자가 동시에 발생하는 상황에서, 의료 자원들을 활용하여 환자를 구조하는 상황을 말한다.

MCI 대응 SoS 는 응급환자 이송 시스템, 병원/응급실 관리 시스템, 도로교통 제어 시스템, 인명구조 시스템들로 구성될 수 있다. 본 논문에서는 응급환자 이송 시스템을 중심으로 MCI 대응 예제 시나리오를 제안하고, 모델링 및 시뮬레이션 기법을 적용해본다. 본 절에서는 MCI 대응 예제에 대해 SoS 의 정의, 특성, 타입에 따른 예제 적합성에 대해 설명한다.

4.1.1 SoS 정의에 따른 예제 적합성

MCI 대응 SoS 는 전체 환자 생존율 향상이라는 공통의 목표를 이루기 위해 CS 인 응급 환자 이송 시스템의 집합으로 이루어져 있다. 각 CS 인 응급환자 이송 시스템의 운영 및 관리는 각 지방 자치 단체에 의해 독립적으로 수행 됨으로써 SoS 정의에 따른 운용 및 관리의 독립성을 만족한다.

4.1.2 SoS 특성에 따른 예제 적합성

SoS 특성에 따른 MCI 대응 SoS 의 예제 적합성은 아래와 같다.

- **자율성:** 각 응급 환자 이송 시스템은 운영 및 관리의 독립성을 갖는다.
- **연대성:** 응급환자 이송 시스템이 각 지역 환자 이송 뿐만 아니라 SoS 수준의 목표를 함께 연대적으로 수행한다.
- **연결성:** MCI 상황이 아닐 시 각 응급환자 이송 시스템은 각자의 목표에 따라 움직이지만, MCI 상황 발생 시 이에 대응하기 위해 동적으로 연결되어 SoS 를 구성한다.
- **다형성:** SoS 목표에 따른 응급환자 이송 시스템의 구성이 미리 정해져 있지 않다.
- **창발성:** 각 환자 이송 시스템의 자율적인 행위에 따라 SoS 목표가 달성되는 창발적 행위가 일어날 수 있다.

4.1.3 SoS 타입에 따른 예제 적합성

MCI 대응 SoS 에서 SoS 의 타입을 표현하기 위해, rate SoS 를 정의한다. rate SoS 는 MCI 지역의 응급환자 이송 시스템이 지방 자치 단체의 응급환자 이송 시스템으로 구급차를 요청했을 때, 이를 수락할 확률을 의미한다. Directed SoS 는 CS 의 소유 및 관리의 권한을 SoS 관리자가 지니고 있기 때문에, 지방자치 단체의 응급환자 이송 시스템은 MCI 지역의 요청을 반드시 수락해야 한다. Acknowledged SoS 는 SoS 수준 소유자가 없기 때문에, 관리의 권한으로 구급차 지원을 요청하지만, 지방 자치 단체의 응급환자 이송 시스템은 이를 반드시 수락할 필요는 없다. Collaborative SoS 는 요청에 의해서 지원하는 형태가 아니라 CS 의 판단에 의해서 MCI 지역으로 구급차를 지원한다.

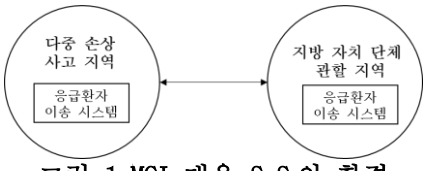


그림 1 MCI 대응 SoS 의 환경

그림 1 은 MCI 대응 SoS 의 환경을 보여준다. 평시에는 각 응급환자 이송 시스템은 CS 로서 각자의 관

할 지역에서 환자 이송을 담당하고, MCI 발생 시 SoS 타입에 따른 전략에 따라 구급차를 지원한다.

4.2 시뮬레이션 모델 적용

MCI 대응 SoS 를 대상으로 3 장에서 소개한 검증 방법인 시뮬레이션 기법을 기반으로 해당 예제를 모델링 한다. 시뮬레이션은 DEVS, ABS, SD 를 함께 사용하는 멀티 패러다임[10] 기법을 적용한다.

MCI 대응 SoS 에 대한 모델링은 크게 CS 모델링과 CS 간 협력 모델링으로 나눌 수 있다. 환자 이송 시스템과 환지 이송 시스템간 협력 부분은 DEVS 모델로 표현 되며, 구급차, 환자는 ABS 모델로, 환자 생존율, 응급차 사용률, 환자 발생률은 SD 모델로 표현될 수 있다.

4.2.1 이산 시간 시스템(DEVS) 모델링

DEVS 는 atomic 모델과 coupled 모델을 함께 사용하였다. 각 지역의 atomic 모델은 환자 발생 이벤트부터 환자의 사망 또는 치유 이벤트까지의 프로세스를 표현한다. Coupled 모델은 각 atomic 모델 간의 구급차 요청 및 지원을 프로세스를 표현한다. 그림 2 는 MCI 대응 SoS 의 DEVS 모델링을 보여준다.

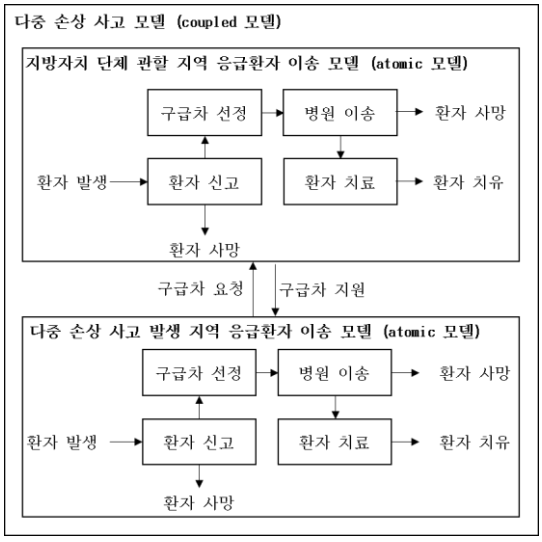


그림 2. MCI 대응 SoS 대상 DEVS 모델

4.2.2 에이전트 기반 시뮬레이션(ABS) 모델링

MCI 대응 SoS 의 에이전트에 해당하는 요소는 구급차다. 구급차는 환자의 상태에 따라, 사망 상태까지 더 적은 시간이 남은 환자를 우선으로 병원에 이송한다.

4.2.3 시스템 다이내믹스(SD) 모델링

시뮬레이션 진행 중 MCI 발생 지역과 지방자치 단체 관할 지역의 총 환자 생존율을 확인하기 위해 SD 모델링이 사용될 수 있다. 환자 생존율은 환자 발생률과 응급차의 사용률에 따라 달라진다. 환자 발생률은 MCI 발생 지역을 지방자치 단체 관할 지역보다 높게 설정한다. 응급차의 사용률은 구급차의 요청 및 수락의 정도에 따라 달라질 수 있는데, 시뮬레이션 결과의 반복을 통해 응급차의 사용률을 100%에 가깝게 할수록 환자 생존율을 높일 수 있다.

5. 결 론

본 연구에서는 SoS 개념을 설명하는 세 가지 방식에 기반하여 MCI 대응 SoS 예제를 제안하고, 예제를 대상으로 모델링 및 시뮬레이션을 수행하였다. 향후 연구로는 시뮬레이션 도구에 모델을 적용하여 MCI 대응 SoS 의 공통의 목표인 환자 생존율을 검증할 예정이다. 그리고 본 연구에서 소개한 시뮬레이션 기반 검증이 아닌 소프트웨어공학 기술인 통계적 모델 체킹을 통한 SoS 목표 검증을 연구 할 예정이다.

참고문헌

[1] Sahin, Ferat, Mo Jamshidi, and Prasanna Sridhar. "A discrete event xml based simulation framework for system of systems architectures.", System of Systems Engineering (SoSE), 2007.

[2] Acheson, Paulette, Cihan Dagli, and Nil Kilicay-Ergin. "Model Based Systems Engineering for System of Systems Using Agent-Based Modeling." Procedia Computer Science 16 (2013): 11-19.

[3] Nielsen, C. B., Larsen, P. G., Fitzgerald, J., Woodcock, J., & Peleska, J., Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions, ACM Computing Surveys (CSUR), 48(2), 18, 2015.

[4] Checkland, P. B. Systems Thinking, Systems Practice. Chichester, UK: John Wiley & Sons Ltd, 1999.

[5] Maier, M. W. "Architecting principles for systems-of-systems." Systems Engineering, the Journal of the International Council on Systems Engineering (INCOSE) 1 (4), 1998.

[6] SEBok, System of Systems(18th, December, 2015), http://sebokwiki.org/wiki/System_of_Systems_(SoS)_(glossary).

[7] INCOSE. 2012. Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, version 3.2.2. San Diego, CA, USA: International Council on Systems Engineering (INCOSE), INCOSE-TP-2003-002-03.2.2.

[8] Baldwin, W. Clifton, and Brian Sauser. "Modeling the characteristics of system of systems.", System of Systems Engineering (SoSE), 2009.

[9] Dahmann, Judith S., and Kristen J. Baldwin. "Understanding the current state of US defense systems of systems and the implications for systems engineering." Systems Conference, 2008.

[10] Ross, Weston, Mihaela Ulieru, and Alex Gorod. "A multi-paradigm modelling & simulation approach for system of systems engineering: A case study.", System of Systems Engineering (SoSE), 2014.

에어컨 실외기 제품의 소프트웨어 재사용성 향상을 위한 피처 기반 제품라인공학 기술 적용 연구

권혁상*, 강성원**, 한영훈**

*㈜삼성전자, KAIST 소프트웨어대학원
** KAIST 전산학부
{ozvoy, sungwon.kang, younghun.han}@kaist.ac.kr

요약: 제품라인공학(Software Product Line Engineering, 이하 SPLE) 방법의 많은 성공 사례들이 과거에 소개되었고 SPLE 가 소프트웨어 개발의 생산성과 품질 향상에 효과적임을 보여주었다. 가전제품에 탑재되는 임베디드 소프트웨어 분야에서도 재사용성 향상을 위해 제품라인 공학 기술은 높은 관심을 끌고 있다. 본 연구에서는 에어컨 실외기 제품에 피처 기반의 제품라인공학 방법론을 적용한다. 도메인 엔지니어링을 통해 도메인의 공통점과 차이점을 분석하여 핵심 자산을 확보하고, 어플리케이션 엔지니어링을 수행하여 확보된 도메인 자산을 바탕으로 실제 제품을 구성하여 소프트웨어 재사용성 향상시키는 사례연구를 수행한다.

핵심어: 소프트웨어 제품라인 공학, 피처모델, 재사용성, 도메인공학, 응용공학, 에어컨 실외기

1. 서 론

임베디드 시스템은 일반 범용의 컴퓨터와는 달리 특정 목적에 맞도록 설계된 하드웨어와 이를 제어하고 사용하기 위한 소프트웨어로 구성된다. 에어컨은 실내기와 실외기, 이를 제어하는 소프트웨어로 구성된 임베디드 시스템이다. 이 중 실외기는 열교환 효율을 높이기 위해 압축기, 센서, 스텝모터 등의 부하가 마이크로프로세서에 탑재된 임베디드 소프트웨어에 의해 제어된다. 에어컨에 내재된 소프트웨어는 열교환을 위해 냉매 순환을 하는 기본 동작 이외에 소비자, 설치자, 공장시험자 등 이해관계자들의 요구사항을 처리하기 위하여 소프트웨어 활용하여 제어한다.

과거 가전제품은 기구와 회로 중심으로 제품이 개발되었으며, 증가하는 고객의 요구사항을 충족하기 위해 최근에는 소프트웨어 기술이 이를 대체하고 있다. 또한, 사물인터넷(IoT) 시장으로 진출하기 위해 에어컨 소프트웨어도 모바일 기기와 연동하는 기능이 추가하는 등 그 규모가 복잡해지고 있다. 따라서, 국내 기업에서도 가전제품에 내재된 임베디드 소프트웨어를 효율적으로 적기에 개발하기 위하여, 소프트웨어 구성요소의 재사용을 높이기 위한 많은 노력을 기울이고 있다.

본 논문에서는 에어컨 실외기 제품에 소프트웨어 제품라인 공학(이하 SPLE) 기술을 적용하여, 향후 에어컨 실외기 제품군 개발에 있어서 소프트웨어 재사용성을 크게 높일 수 있도록 하고자 한다. 제품라인 구축의 출발점으로 기존의 개발 산출물들을 활용하였고, 제품라인 구축을 위해서 다양한 SPLE 개발 방법론 중에서 [4]의 방법을 선정하였다. 논문 [4]는 기존 제품을 재사용하는 방법으로 기업에서 SPLE 기술을 처음으로 도입하기 위해 사용하는데 적합한 방법론이다. 이 연구에서는 [4]의 개발 프로세스를 따라 에어컨 실외기 제품의 공통성과 가변성을 분석하고 가변성이 포함된 아키텍처와 컴포넌트를 도출한다.

이하 논문의 구성은 다음과 같다. 2 절의 배경지식에서 소프트웨어 제품라인 공학의 방법론의 종류를 알아본다. 3 절에서는 본 연구에서 사용하는 프로세스를 정의한다. 4 절에서는 도메인 공학에 따라 제품을 분석하고, 응용 공학에서 도메인공학에서 정의된 자산을 활용하여 제품을 설계한다. 5 절에서는 본 연구로부터 얻은 교훈을 논의한다. 6 절에서는 결론을 맺는다.

2. 배경지식

2.1 소프트웨어 제품라인 공학

소프트웨어 제품라인 공학은 소프트웨어의 생산성과 품질을 향상시키기 위한 방법론으로 도메인공학과 응용공학으로 구성되어 있다. 도메인 공학(Domain Engineering, 이하 DE)에서는 제품 설계를 통해 해당 제품 도메인의 공통점과 차이점을 분석하여 핵심 자산을 확보하고, 응용 공학(Application Engineering, 이하 AE)에서는 제품자산의 스코핑(scoping)을 통해 제품을 정의하여 특정 제품을 구성하는 product line life cycle 을 수행한다[1]. SPLE 는 여러 가지 방법론이 소개되고 있으며 각각의 방법론의 핵심 요소와 특성에 따라서 개별 방법론의 장점이 있다. 대표적인 SPLE 방법론과 그 특징은 다음과 같다[2].

- FORM(Feature Oriented Reuse Method)

  FORM 은 SPLE 를 위한 초기 마케팅 및 제품 계획

단계부터 정해진 의사결정을 바탕으로 피처 모델을 구성하고 추출식(Extractive), 선행적(Proactive) 방식으로 개발하기에 적합한 기법이다.

- PLP(Product Line Practice)

미국 CMU/SI 의 방법론으로서, SPLE 전체에 걸친 기술을 종합하여 다양한 접근이 가능하도록 하였고 특히 아키텍처 명세와 평가를 강조하였다.

- FAST(Family-Oriented Abstraction, Specification, and Translation)

Lucent Technology 의 FAST 는 PASTA (Process and Artifact State Transition Abstraction) 모델을 제시하여 수행될 프로세스에 반복과 재사용이 용이한 프로세스를 제시하였다.

- AHEAD(Algebraic Hierarchical Equation for Application Design)

University of Texas 의 방법론으로 반복적인 피처 정련(refinement)을 통해 SPLE 을 점차 확대해 나가는 반응식(reactive) 방식을 주로 사용한다.

- GP(Generative Programming)

University of Ilmenau 의 방법론으로 상세한 피처 모델을 바탕으로 프로그램 코드를 생성시키는 방식을 제시한다.

- PuLSE(ProdUct Line Software Engineering)

독일 Fraunhofer IESE 의 방법론으로, 재공학(re-engineering)을 통해 기존 제품들로부터 SPLE 을 구성하는 추출식(Extractive) 방식을 적용하기에 용이하다.

2.2 FORM 방법론

2.1 절에서 열거한 방법론 중에서 피처 기반의 SPLE 방법론을 적용할 때 얻을 수 있는 장점은 다음 세 가지이다[3].

- 피처 지향성(feature orientation)

피처는 사용자나 개발자가 식별할 수 있는 시스템의 구분되는 특징을 의미하는 것으로, 피처 중심의 도메인 내 시스템 분석은 분석 단계에서 시스템의 공통점과 차이점을 파악함으로써 시스템의 재사용성을 높여주게 된다. 시스템 개발에 참여하는 여러 사람들, 즉 사용자, 분석자, 설계자, 개발자 간의 관심 도메인에 따라 네 개 계층의 피처로 분류하고 이들의 관계를 체계적으로 설정함으로써 분석 단계의 모델로서만 그치는 것이 아니라, 설계 및 구현 단계에도 영향을 미치는 소프트웨어 생명 주기 전반에 영향을 주는 모델이다.

- 객체 지향(object orientation)

객체지향은 도메인이 안정됨에 따라 도메인 내 서비스는 변화가 거의 없고, 새로운 기술이나 운영 환경의 변화에 따라 소프트웨어를 유지 보수하는 경우가 대부분인데, 이러한 변화 가능성이 있는 운영 환경 요소나 특정 기술을 나타내는 알고리즘을 객체로 한정시킴으로써 변화에 쉽게 대처할 수 있도록 한다. 기존의 객체 지향 방법에서는 단일 시스템에 대한 요구사항을 분석하여 객체를 찾아내며 도메인 내 공통점과 차이점을 고려하지 않기 때문에 새로운 요구사항이 생기거나 새로운 기술이 도입될 경우 이를 쉽게 수요하기 어려울 수 있는데, 반면 피처 모델은 도메인 내 시스템의 공통 부분과 변화 가능한 부분을 체계적으로 구조화한 모델이므로 효과적으로 객체를 찾아낼 수 있고, 도메인 내 여러 시스템에 적용 가능한 중요한 객체가 된다.

- 관심 분리(separation of concern)

시스템 개발에는 사용자, 분석자, 설계자, 개발자 등이 참여하게 된다. 서로 다른 지식 배경과 관심 분야를 가진 사람들이 같은 목적을 가지고 일을 하기는 매우 어려움에 따라 소프트웨어 개발 방법론은 이러한 다양한 배경을 가진 사람들의 지식을 효과적으로 표현하고 이들에게 원활하고 용이한 의사소통 수단을 제공해야 한다.

FORM 방법론을 통해 참여자들 각각의 관점에서 모델을 제공하여 각자의 관심 분야를 정확하고 체계적으로 모델링하고 참여자들 상호 간에 효율적으로 의사 소통할 수 있는 모델을 제시한다.

2.3 FORM 엔지니어링 프로세스

SPLE 는 DE 와 AE 으로 구성된다. DE 는 정해진 도메인의 시스템들에 대한 참조 모델을 만드는 공정이고, AE 은 DE 의 산출물인 참조모델을 바탕으로 새로운 시스템을 만드는 공정이다. 도메인 공학과 응용 공학은 그림 1 과 같이 각각 DE1~DE4 와 AE1~AE4 로 이루어진다.

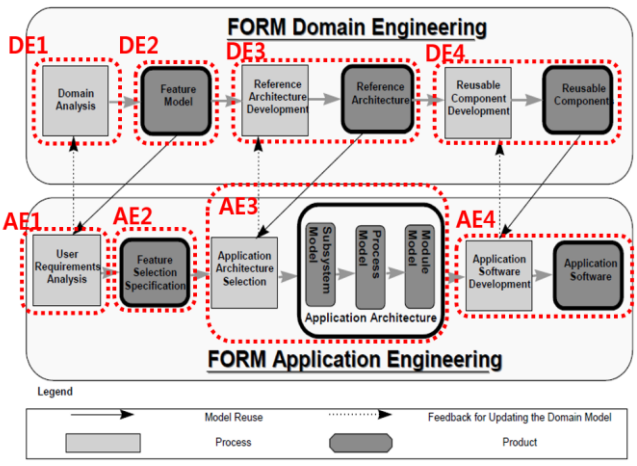


그림 1. FORM 프로세스[3]

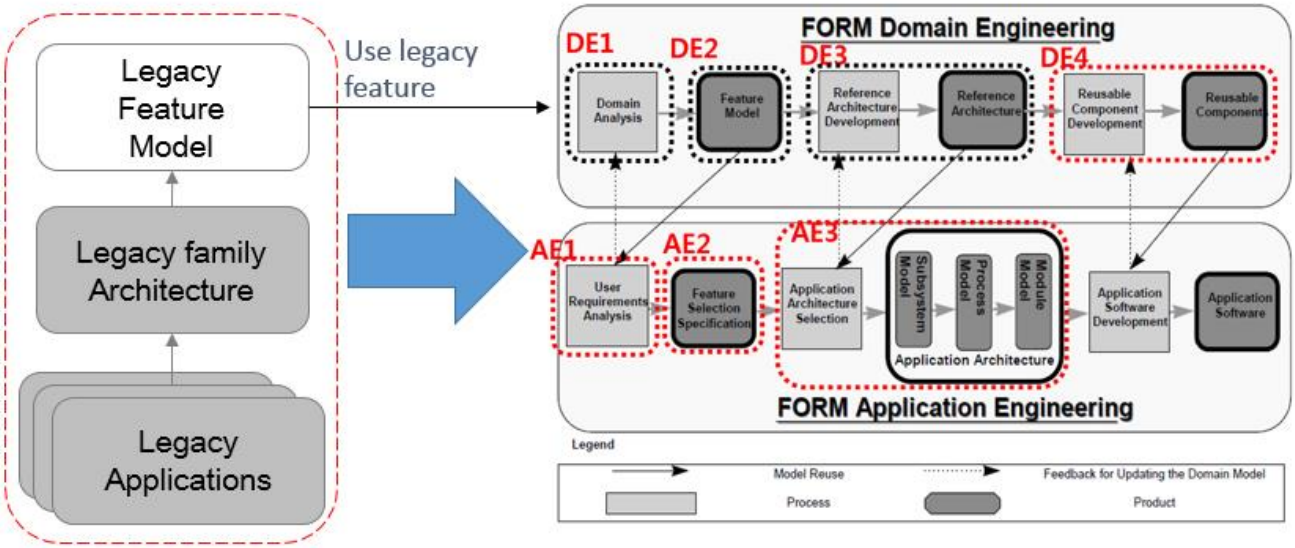


그림 2. 기존제품의 분석이 추가된 FORM 프로세스

표 1. OVDL 표현 예[6]

PL	가변점에 의한 가변부의 추상화	BatteryTube — vpLM
	가변점의 가변성 정의	vpLM : RedLM , GreenLM

2.4 피처의 가변성 표현 방법

피처의 가변성을 표현하는 여러 방법이 제시되었는데[8], Thiel et al.[10]과 Pohl et al.[11]의 독립적 가변성 모델을 이용한 표현 방법과 Gomma 의 UML 의 스테레오타입을 확장하여 다양한 UML 모델에 가변성을 표현한 방법, Moon et al.[12]의 제품라인 아키텍처 컴포넌트와 인터페이스, 함수에 가변성을 나타내는 방법 등 여러 가지 방법들이 있다. 이 중에서 가변성을 본 연구에서는 OVDL (Orthogonal Variability Language) [7]로 표현한다. 표 1 은 OVDL 표현의 예이다.

3. 기존 제품 분석 단계로 확장한 FORM 엔지니어링 프로세스

새로운 제품을 기획하는 경우가 아닌 기존 제품을 바탕으로 FORM 방법론을 적용하는 과정은, 그림 2 와 같이 기존제품(Legacy Product)의 피처 분류를 참조하여[4] 프로세스에 적용하는 과정이 추가된다.

그림 2 에서 왼쪽부분은 기존제품으로부터 피처 수준까지 재구축하는 단계이다. 이 단계에서는 제품들로부터 제품 아키텍처가 추출 되고, 제품 아키텍처에서 피처 분석을 하여 참조할 수 있도록 전처리 하는 과정이 추가 된다.

도메인 엔지니어링은 다음 네 단계로 이루어진다.

- DE1(Domain Analysis)
  도메인 분석을 수행
- DE2(Feature Modeling)
  피처 모델링을 수행하기 위해 사용자 요구사항을 통한 분석을 수행.
- DE3(Reference Architecture)
  참조 아키텍처를 Subsystem model, Process model, Module model 의 3 개의 모델로 구축하고 [3], OVDL 을 사용하여 아키텍처의 각 파트의 가변성 표현
- DE4(Reusable Components)
  위의 DE1~DE3 을 통해 얻어진 도메인 공학의 결과에 따라 재사용 콤포넌트를 구성

응용 공학은 다음 네 단계로 이루어진다.

- AE1(User Requirement Analysis)
  사용자 요구사항 분석 수행
- AE2(Feature specification selection)
  피처 선택은 DE2에서 수행한 피처 모델에서 선택
- AE3(Application Architecture selection)
  아키텍처를 선택하는 단계는 DE3 에서 작성된 도메인 아키텍처에서 아키텍처의 가변부에서 가변점을 선택하여 구성
- AE4(Application Software Development)
  실제 제품 소프트웨어 개발

표 2. 기존 제품의 피처 분석

구분	대분류	소분류	컴포넌트 개수
기능	디바이스	스텝모터	6
		센서	
		저장소	
		디스플레이	
	인터페이스	통신	3
	기능제어	에러	8
		보호제어	
		로직	
		운전	
	서비스	테스트	2
		서비스운전	
	UI	GUI	1
비기능	공통	성능	4

4. 사례연구

기존제품들의 어프리케이션을 바탕으로 구성된 제품군의 아키텍처를 이용하여 제품의 피처 모델을 분석하여 FORM 방법론의 도메인 엔지니어링의 피처 모델링에 참조할 수 있도록 한다.

기존 제품의 분석을 통하여 파악된 기존 제품의 피처는 표 2 와 같다.

4.1 도메인 공학

- DE1(Domain Analysis)

에어컨 실외기는 기구, 부품, 회로의 관점에서의 분석, 소프트웨어의 관점에서의 분석의 2 가지로 구성이 되며, 각각 표 3 에서처럼 13 가지 OFP(Outdoor Function Part)로, 표 4 에서처럼 16 가지의 OSP(Outdoor Software Part)로 분류된다.

- DE2(Feature Modeling)

분석결과는 그림 3 과 같다. 요구사항 분석을 통해 피처들을 추출하고 그림 4 와 같이 피처 카테고리로 나누었다. 피처 카테고리는 FORM 에서 제시된 4 가지 계층에 따라서 카테고리를 나누고 해당되는 피처를 배치하였다.

생성된 피처 카테고리에 따라서 만들어진 피처 모델은 그림 5 에 있다. 피처 모델에서는 각 피처에 대하여 Mandatory/Optional 과 Alternative 의 관계를 표시하고, 각 피처 간의 가변성 의존관계를 표시하였다.

표 3. 기구, 부품, 회로 관점의 분석

ID	실외기 기능 부품
OFP1	팬모터
OFP2	열교환기
OFP3	압축기
OFP4	냉매량 조절 밸브
OFP5	압력 센서
OFP6	온도 센서
OFP7	Step motor
OFP8	Display
OFP9	통신선
OFP10	밸브
OFP11	냉매
OFP12	Option 장치
OFP13	중앙제어장치

표 4. 소프트웨어 관점의 분석

ID	S/W 기능
OSF1	통신
OSF2	기본 운전
OSF3	냉매 분류
OSF4	보호제어
OSF5	기타 기능
OSF6	특수 운전
OSF7	Error 처리
OSF8	Storage
OSF9	성능 제어
OSF10	공장 테스트 기능
OSF11	서비스 운전
OSF12	Device 제어
OSF13	통신 Protocol
OSF14	OS
OSF15	SW Platform
OSF16	보안 기능

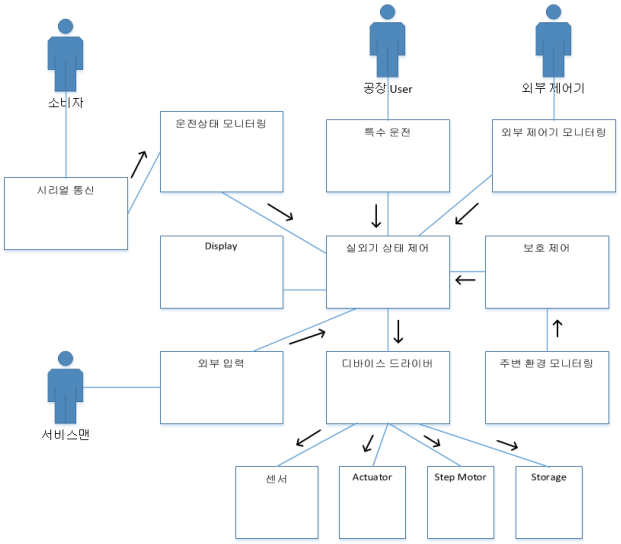


그림 3. Use case 요구사항 분석

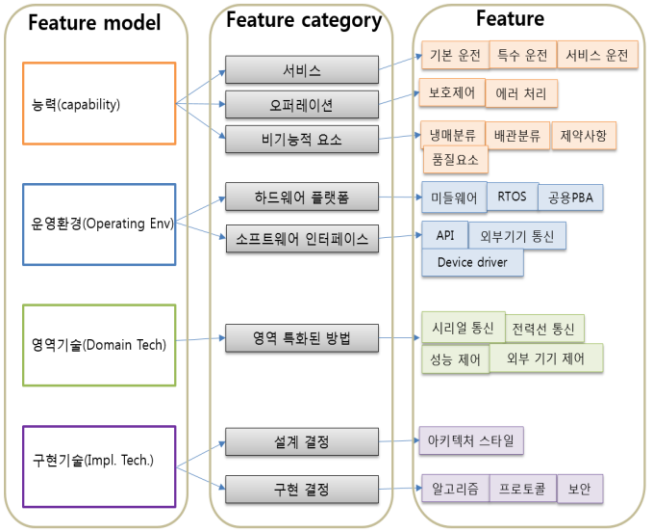


그림 4. 피처 카테고리

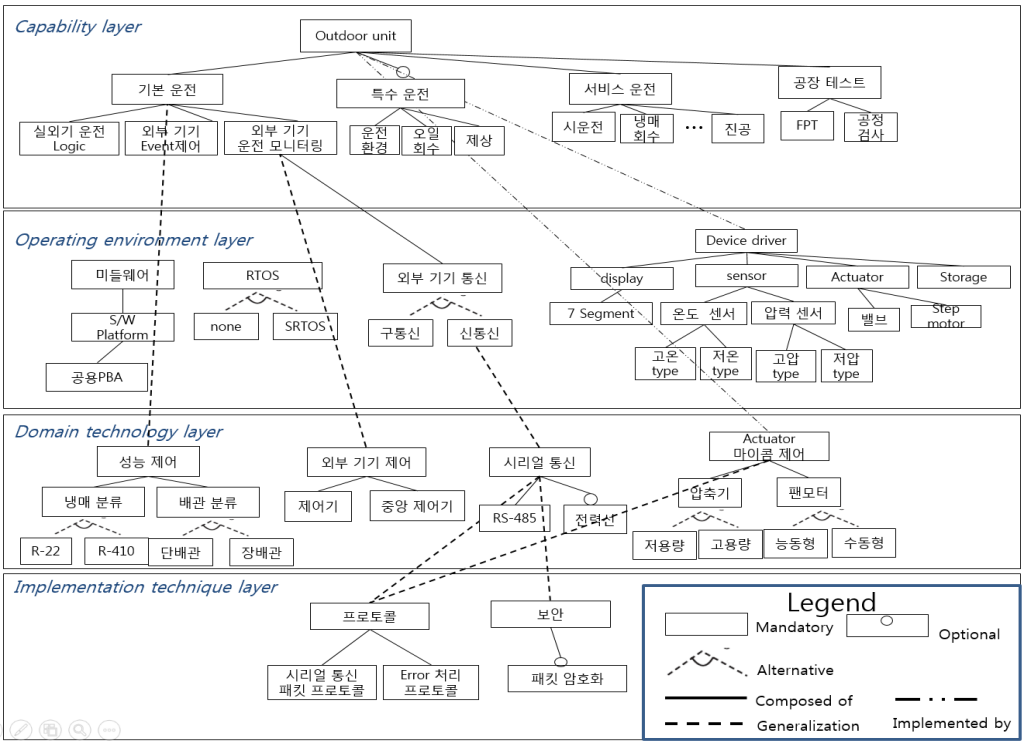


그림 5. 피처 모델

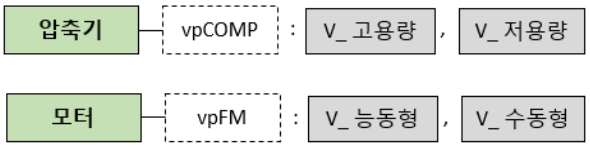


그림 6. OVDL 가변성 표현 예

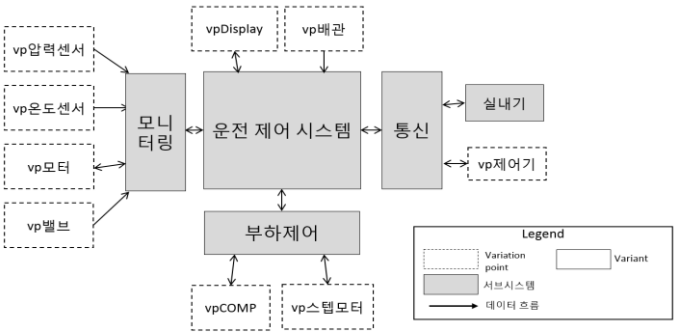


그림 7. 서브시스템 모델

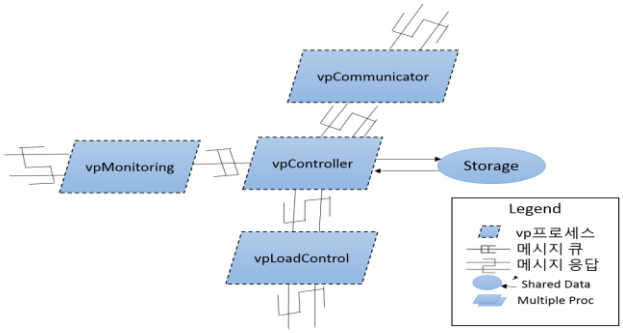


그림 8. 프로세스 모델

그림 9. 가변점 구성의 예

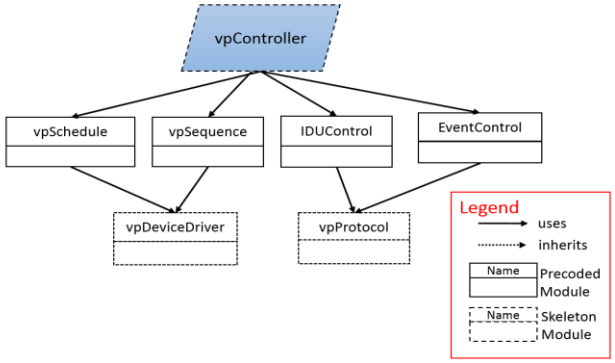


그림 10. 모듈 모델

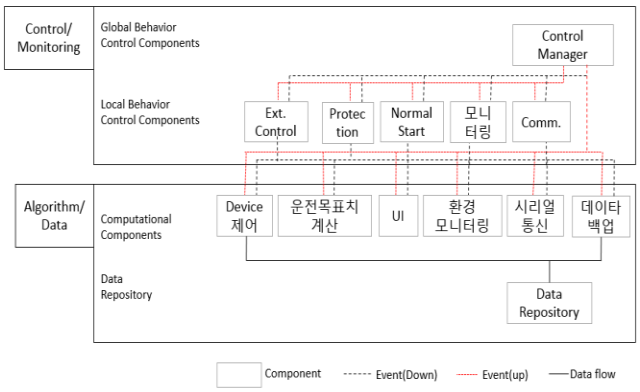


그림 11. 재사용 컴포넌트

- DE3(Reference Architecture)

OVDL 방법을 이용하여 가변성을 표현하기 위해 vp(variation point)를 표시한 결과는 그림 6 과 같다..

1) Subsystem Model

Subsystem 모델은 그림 7 와 같이 구성되며, 운전제어 시스템을 중심으로 센서, 밸브, 모터등의 Subsystem 으로 구성된다.

2) Process Model

프로세스 모델도 Subsystem 모델과 같이 OVDL 을 이용한 가변성을 표현하고자 vp 로 표시하였다. 그림 8 는 가변점 구성의 예를 보여준다.

3) Module Model

모듈 모델은 그림 10 과 같이 구성이 된다. vpMonitoring, vpCommunicator 모델과 가변점 구성의 예는 일대일 매칭이여서 생략하였다.

- DE4(Reusable Components)

재사용 콤포넌트는 DE3 에서 수행된 아키텍처 구성에 따라 legacy products 의 기정의된 콤포넌트를 바탕으로 그림 11 과 같이 구성된다. 제시된 재사용 콤포넌트 설계는 응용 공학 수행시 참조 모델로 사용할 수 있다.

4.2 응용 공학

응용 공학은 도메인 공학 단계에서 만든 피처 모델과 아키텍처 그리고 재사용 콤포넌트를 바탕으로 제품의 스코핑(scoping)을 수행하여 실존하는 제품을 재사용을 통해 구성했다.

응용 공학은 AE1~AE4 의 네 가지 단계로 3 절의 프로세스에서 제시되어 있으나, 이 절의 사례연구는 기존의 제품으로부터 구성되므로 AE4 의 Application Software development 부분은 제외한다.

- AE1(User Requirement Analysis)

제품 요구사항 분석은 기존 제품 중에서 선택하였으며 요구사항분석 결과는 표 5 와 같다.

- AE2(Feature specification selection)

피처 결과는 표 6 와 같다. DE2 의 결과로 구성된 피처 모델에서 피처를 선택하여 제품에서 구성할 피처 모델을 구성할 수 있었으며 그림 12 에서 같이 점선으로 표시된 피처는 삭제되는 피처이며 굵은 네모로 표시된 피처는 선택된 피처이다.

- AE3(Application Architecture selection)

도메인 아키텍처에서 아키텍처의 가변부에서 가변점을 선택하여 구성한 결과는 그림 13, 14, 15 과 같다.

표 5. 요구사항 분석

요구사항	기능 연결
팬	전면 토출 방식
모터	마이콤 내장 능동형 모터
압축기	8 마력 단일 압축기
냉매 타입	R410
압력 조절	고압/중압/저압 센서
온도 조건	냉방 운전 전용
통신	시리얼 통신, 실외기 대수 - 1 대
운전 기능	냉매 냉각 기본 기능, 보호제어
특수 운전	압축기 오일회수
서비스 운전	공장 자동 테스트 운전

표 6. 피처 선택 개수

구분	대분류	소분류	대표 spec	상세 spec
2	6	10	42	91

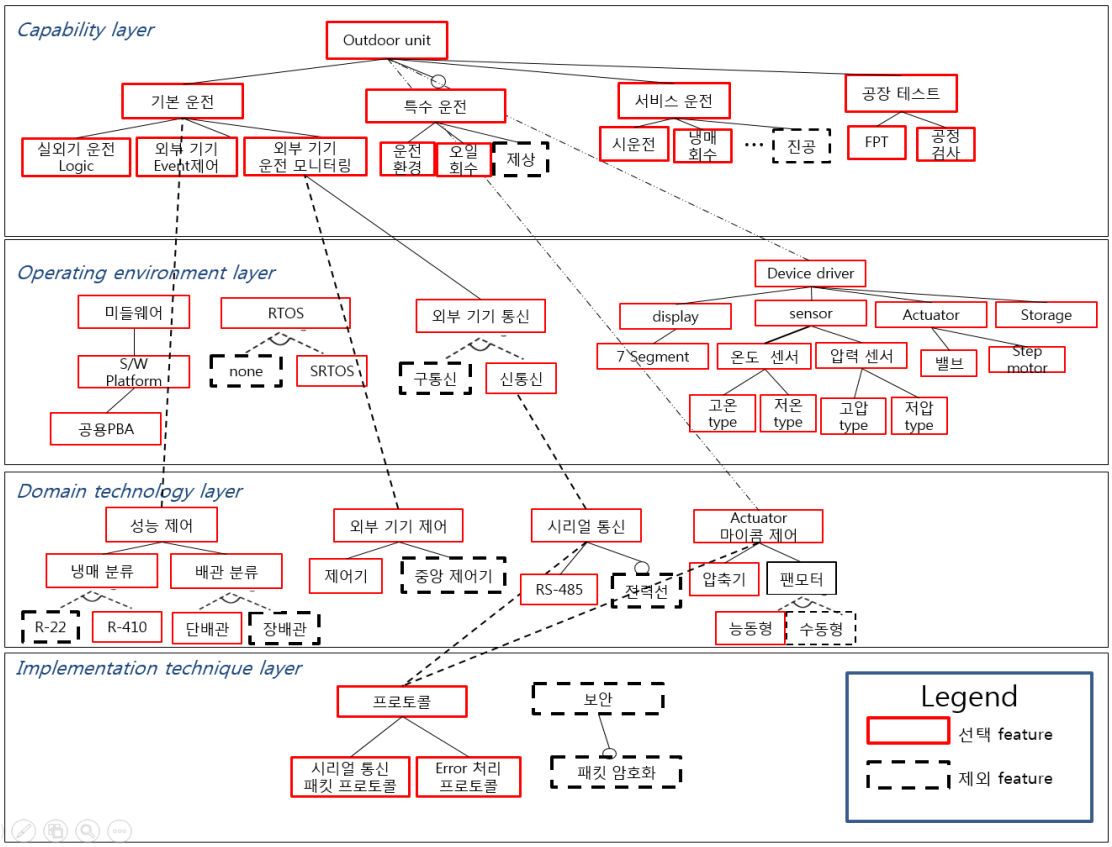


그림 12. 선택된 피처 모델

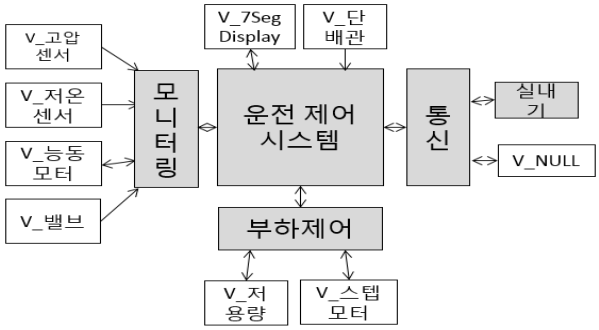


그림 13. 선택된 서브시스템 모델

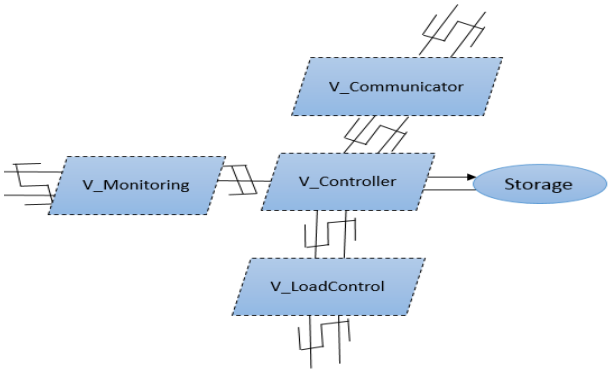


그림 14. 선택된 프로세스 모델

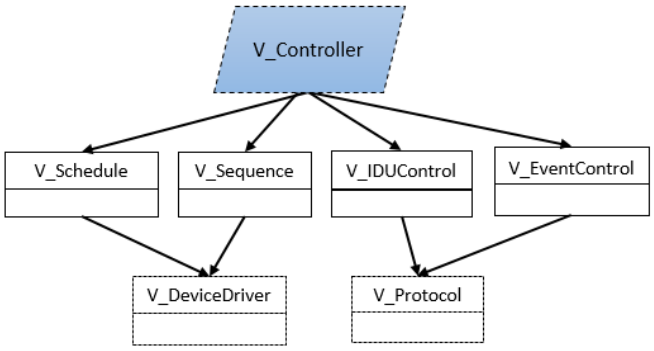


그림 15. 선택된 모듈 모델

5. 교훈

이 과정에 우리는 기존 제품을 활용하는 체계적인 가이드라인 부재로 인한 어려움을 경험하였으며, SPLE 이 재사용에 대한 효율성을 입증하려면 방법론을 정량적으로 측정할 수 있는 지표가 필요하다는 결론을 얻었다.

사례 연구를 수행하면서 두가지 교훈을 얻게 되었다.

첫번째로, 도메인 자산은 한번에 수립되지 않고 어플리케이션 엔지니어링 과정과 함께 지속적인 정련을 통해 완성해 나가는 과정이라는 것이다.

FORM 방법론의 프로세스에 따라 3 절과 4 절에서 기존의 제품의 분석을 바탕으로 실외기 제품의 도메인 엔지니어링을 통해 공통부와 가변부를 정의를 하여 피처 모델, 참조 아키텍처 작성을 하고, 도메인 엔지니어링의 결과를 바탕으로 어플리케이션 엔지니어링 과정을 수행하여 정의된 자산을 바탕으로 피처를 선택하여 제품의 피처 모델과 아키텍처 구성을 시도해 보았는데, 도메인 엔지니어링 과정을 수행할 때 처음부터 모든 제품을 포괄하는 피처와 아키텍처 자산을 정의 하고자 하였으나, 어플리케이션 엔지니어링 과정을 수행하며 많은 도메인 엔지니어링 결과가 수정이 되었다.

두번째로, 방법론 적용의 효용성을 인정받기 위해서는 정량적이고 실질적인 지표가 필요하다. 본 연구의 결과로 정성적으로는 개발 구성원들의 설문 평가와, 정량적인 결과로 피처의 응집도(cohesion)[9]로 측정하고자 하였으나, 체계적인 측정 지표로서 부족함을 보여 결론에서는 생략을 하였다.

6. 결론

본 연구는 SPLE 방법론 중 FORM 방법론을 에어컨 실외기 제품에 적용하여 제품라인 공학의 적용 가능성의 여부를 살펴보고, 그리고 피처 모델과 참조 아키텍처 설계를 통해 소프트웨어 재사용성을 향상시키는 방법을 찾고자 하였다. FORM 방법론의 프로세스를 준수하되 기존 제품이 있는 경우에 해당 방법론을 적용하는 경우 기존제품의 피처 구성을 작성하여 도메인 엔지니어링에서 피처 모델링에 참조하도록 프로세스를 부분 추가하였고, 실제 모델의 구성까지 시도해 보았다.

오랜 시간과 많은 시행착오를 통해 FORM 방법론의 적용을 수행해본 결과 지속적인 정련의 필요성과 설득력 있는 지표를 통한 정량적인 지표 측정 방법으로 결과를 측정하였을 때 해당 방법론의 적용이 설득력을 가질 수 있음을 교훈으로 얻었다.

또한, 얻어진 교훈을 통해 본 연구의 결과를 바탕으로 현업 개발자로서 새로운 제품을 구성하며 도메인 엔지니어링의 결과물을 정련을 지속적으로 수행할 것이며, 정량적인 지표를 얻고자 연구를 지속할 예정이다

참고문헌

[1] Martin Becker, "Product Line Engineering Lecture - Scoping(3)," Fraunhofer IESE.

[2] 한국소프트웨어진흥원, 소프트웨어 제품라인 공학기술 적용가이드, 2009 년 7 월.

[3] Kyo C, Kang, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," 1998.

[4] Kyo C, Kang, "Feature-Oriented Re-engineering of Legacy Systems into Product Line Assets - a case study", The 9th International Conference on Software Product Lines, pp. 45-56, 2005.

[5] 최현식, 이혜선, 조윤호, 강교철, "기존 시스템 기반의 소프트웨어 제품라인 공학 기법: 케이블 셋톱박스 소프트웨어 사례," 정보과학회 논문지, 소프트웨어 및 응용, 제 36 권 7 호(2009.7).

[6] 강성원, *체계적인 소프트웨어 제품라인 개발 이론*, pp 74-78, 2015.

[7] 이지현, 강성원, "소프트웨어 프로덕트라인 가변성 기술 기법 OVDL," 정보처리학회논문지 - 소프트웨어 및 데이터 공학, v.2, no.11, pp.739 - 746, 2013.2.

[8] 이혜선, 조성배, 강교철, "소프트웨어 제품라인 아키텍처 모델에서의 가변성 표현 방법 비교 연구," 소프트웨어 공학 소사이어티 저널, 24 권 3 호, pp. 79-89, 2011. 9.

[9] 김세훈, 김정아, "소프트웨어 제품라인의 피처모델과 구성요소간 가변성에 대한 일관성 검증 규칙," 정보처리학회논문지- 소프트웨어 및 데이터 공학, 3 권 1 호, pp.1-6, 2014.

[10] S. Thiel and A. Hein, "Systematic Integration of Variability into Product Line Architecture Design," 2nd International Software Product Line Conference (SPLC 2002), LNCS 2379, pp.130-153, 2002.

[11] K. Pohl, G. Böckle, and F. van der Linden, "Software Product Line Engineering: Foundations,Principles, and Techniques," Springer-Verlag New York Inc., 2005.

[12] M. Moon, H.S.Chae, and K.Yeom," A Metamodel Approach to Architecture Variability in a Product Line," International Conference on Software Reuse (ICSR 2006), LNCS 4039, pp. 115-126, 2006.

A Software Development Framework for Industrial Computer Vision Systems

Mesfin Abebe*, †Cheol-Jung Yoo*
* Chonbuk National University

567 Baekje-daero, Deokjin-gu, Jeonju-si,
Jeollabuk-do, Republic of Korea

mesfinabha@gmail.com, cjyoo@jbnu.ac.kr

† *Corresponding Author*

Abstract: Computer vision systems are potentially powerful tools to inspect the quality of industrial products in the modern manufacturing processes. However, computer vision is a broad field that involves several techniques, topics, and high-dimensional data. Hence, it is challenging to build a computer vision system. In this study, we proposed an industrial computer vision system development framework that systematically organized the activities and tasks. Though we incorporated only few examples; the framework was used to design, implement and test an industrial computer vision inspection system to validate its feasibility. The result shows that the proposed framework successfully supports the modeling of the system requirements, behaviors and structures of a computer vision system. In addition, it also facilitates the validation and verification of the system development process.

Keywords: Computer vision, development framework, software development life cycle.

1. Introduction

The application of computer vision in the industries has been increased considerably in the recent days. Its application covers a wide range of activities such as process control, navigation, event detection, information organization, object modeling, and automatic inspection [1]. One of the areas where its application spreads is in the manufacturing industry to inspect the quality of the products. Computer vision is the automatic extraction, analysis and understanding of the content of images to produce useful information that facilitates a decision making process [2]. The union of image analysis and other technologies to support the automation of the industrial manufacturing activity is also known as machine vision [3].

A Software development process describes the organization of the activities and tasks of the software development. Furthermore, it provides the precise steps required to accomplish the tasks and to identify the method and tools that are adopted for the project [4]. After the invention of the first software development process: *waterfall process*; many development processes has been created such as *Spiral*, *RUP*, *Incremental*, *agile* and etc. [5]. Nevertheless, these processes aren't good enough to fit the activities and tasks of a computer vision system requirements and features. Therefore, a systematic approach is required that maps the activities and tasks of a computer vision system to a selected development process. Generally, this requires the structuring and reorganizing of the activities and tasks of the vision system development into the different phases of the software development life cycle (SDLC). Hence, the main objective of this study is to establish a computer vision system development framework that facilitates the design, implementation and testing phases using existing development process, method and modeling language.

The rest of the paper organizes as follows; *Section 2* explains an overview of computer vision system. *Section 3* describes the previous works. *Section 4* provides the detail of the framework. *Section 5* presents the result and discussion of the study. Finally, *Section 6* concludes and indicates the future study areas.

2. Overview of Computer Vision System

Vision is one of the most important and advanced senses of human beings. More than 70% of the information we receive from the external world is acquired through our vision system [6]. Researchers have studied for decades to understand the internal process of human vision; though, they are understand few things. Currently, computer vison systems become an important source of information in computer-aided design (CAD) and computer-aided manufacturing (CAM). This brings a paradigm shift of thinking, where images are considered useful not only for information source but also for decision making and performing

actions [7].

A computer vision system is an economical, consistent and objective technique of quality inspection. Its high accuracy satisfies the continuous escalating demand of the industrial quality inspection need. However, an effective computer vision system development requires an understanding of the hardwares, software requirements and the application domain. Though the hardware components vary across the application domains; most of them include the following fundamental components: *power source*, *image acquisition devices*, *computing unit and control* and *communication cables*.

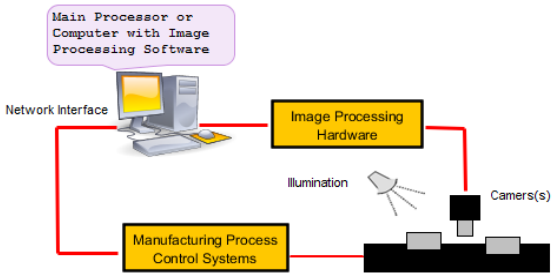


Figure 1: A typical industrial vision system components

These days, there are many software development methodologies, which standardize the activities of the software development. These methods support the conventional software system development phases such as; requirement engineering, design and modeling, implementation, verification and validation and Maintenance. However, they do not fit directly to the unique features of computer vision systems, which consist of *system engineering* and *software Engineering*. Thus, the development of a computer vision system required the reframing of the tasks and activities. This needs a thoughtful consideration of the hardware, software, domain specific requirements and constraints, and also the complex, concurrent and secure nature of the computer vision systems.

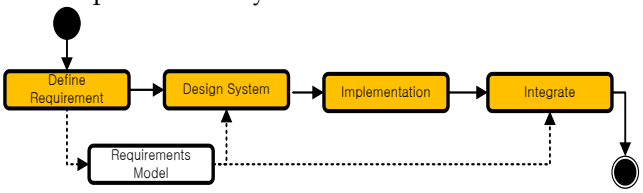


Figure 2: Software development lifecycle

3. Related Woks

Computer vision systems have been used for quality inspection process in the past 30 years. Since then, it evolved rapidly and becomes as one of the most promising technologies. Though there are many computer vision studies; we mentioned here only studies that related to the purpose of our study. Giacomo and Cesare described the use of W model in combination with the concept of model based development. Their approach is based on Model-based system engineering to simplify the complexity of a computer vision system development [8]. Likewise, another study provided a SysML-based behavior modeling method for mechatronic system [9]. They also described the SysML as a uniform behavior model that transform to different simulation platform. Bassi et al. proposed a design methodology that generates a hierarchy of models that describe the system at different levels of abstraction [10]. The models are arranged in such a way to make the mapping of each abstraction level simple. Chami et al. introduced a SysML based method for Intelligent Conceptual design evaluation of mechatronic systems [11].

Our study proposes a framework based on the V-shape process, COMET (Concurrent Object Modeling and Architectural Design Method) method and SysML modeling language. The framework reduces complexity by improving the tractability through the SysML diagrams and COMET guidelines and concepts.

4. Proposed Development Framework

4.1 Overview of the Proposed Framework

Building a computer vision system is beyond the reach of many peoples unless they have a good understanding of image processing, system modeling and computer programming [12]. Though these three concepts are equally important, in this study we consider the system modeling, which is the most neglected concept in computer vision systems study. In reality, our proposed framework has to support the *embedded*, *real-time* and *concurrent* nature of the computer vision systems. Therefore, we intermingled and adopted two of the existing standard process and method to build the development framework, which is suitable to the domain area and operational environment of an industrial computer vision system.

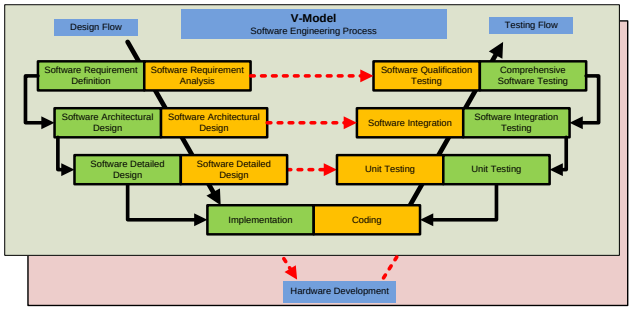


Figure 3: V Shaped development process [12]

This study unified various concepts and techniques of software development to overcome the challenges in computer vision system development. While computer vision systems have unique characteristics, they also follow similar development phases of the conventional SDLC: *Requirement definition*, *Architectural design*, *Detail design*, *Implementation*, *Unit testing*, *Integration testing* and *Software Testing*. We adopted the V-shape process to efficiently *describe*, *manage* and *control* the phases, activities and tasks of the software development process as shown in *Figure 3*. Along with V-shape, the framework includes the best features of SysML and COMET to define and communicate the activities and tasks using the highly traceable SysML diagrams and COMET concepts and guidelines. Therefore, the framework can create good understanding and communication among the practitioners. It also takes into consideration the unique features of the computer vision system and the corresponding industrial products and engineering process.

Though the COMET method uses UML notation, we prefer to apply SysML instead of UML; since SysML is more suitable to represent the hardware and software requirements effectively. Moreover, SysML is good to enhance traceability among the models at various phases with the help of its *requirement diagram*. In contrast, the COMET method provides design principles that simplify the design of *concurrent*, *real time* and *distributed objects* of the software system. The COMET method strongly supports only the first three SDLC phases: *Requirement modeling*, *Analysis modeling* and *Design modeling*. Therefore, we used the use cases and the state chart diagrams to generate test cases to stretch the development framework to cover the validation and verification phase of the SDLC. Both the V-shape process and the COMET method follow an iterative approach. They also provide a detail description of their life cycle. Accordingly, the COMET method is compatible with the V-shape process that can be used together [13]. The incremental software integration and system testing activities of COMET also easily mapped to the validation and verification phase of the V-shape development process. Generally, since a computer vision system is a component of a large hardware and software system, the framework also supports the system molding and software engineering activities of the development task.

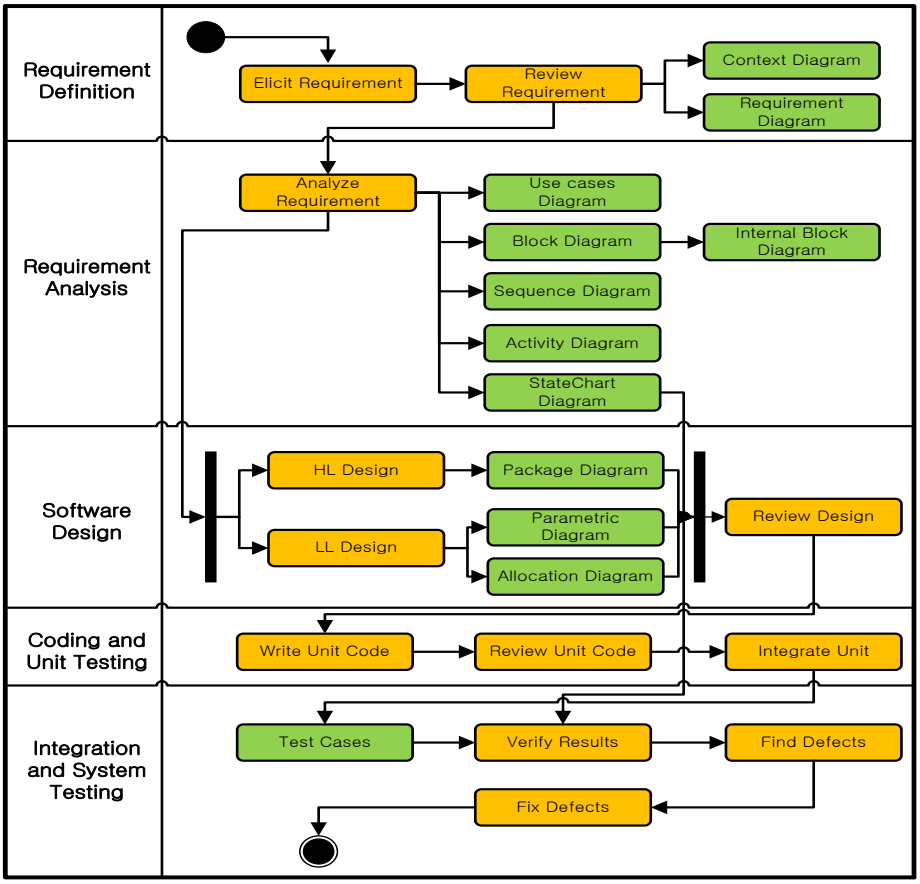


Figure 4: Mapping the activities, SysML diagrams and the development phases

4.2 Detail of the Development Framework

Figure 4 depicts the activities and SysML diagrams of the framework under each phase the development process. In this section, we briefly explained how the development framework associates the SysML diagrams, the COMET model and the V-shape process.

Requirement Definition: this phase describes the activities required to elicit and present the functional requirements, nonfunctional requirements and constraints of the system. This phase has two sub phases as it is adopted from COMET method:

- ***Requirements Elicitation:*** involves the identification and gathering of the functional and nonfunctional requirement using interview, observation etc. techniques to extract the requirements from the customers. It also includes the domain specific requirements.
- ***Requirements Specification:*** is the process of documenting the requirement from the requirement elicitation process. It includes all the requirements that describe the purpose and quality of the software system. The requirements are specified and organized in a document known as Software Requirement Specification (SRS) that provide a complete view of the system behavior.

An industrial computer vision system is usually part of a large *Hardware* and *Software* system, which also has a system requirement analysis and specification. The SysML *Requirement diagram* and context diagram are used for the system and software analysis. The COMET *software system context diagram* is used to define the external agents' interaction with the system in the scope of the system boundary. This describes the association between the different hardware and software of the system as shows in *Figure 5*.

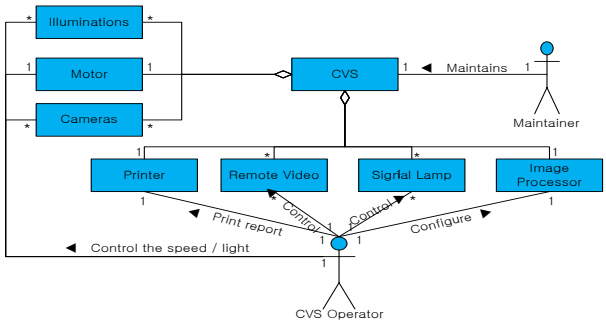


Figure 5: System context diagram of an inspection system

Requirement Analysis: in this phase, static and dynamic models are depicted to define the structural and dynamic nature of the system. The SysML *Black diagram* and *Internal block diagram* are used to define the structural relationships of the hardware and software comments at different stages of the abstraction process as shown in *Figure 6*. The dynamics models *Use case diagrams*, *Sequence diagrams*, *Activity diagrams* and *State chart diagrams* are define the dynamic nature of the system. The use cases diagram describes the users that participate in the use cases and the interactions between the users and the system. The objects and their association is depicted with the help of the sequence diagrams. Moreover, in the dynamic model state-dependent objects are defined using the State chart diagrams.

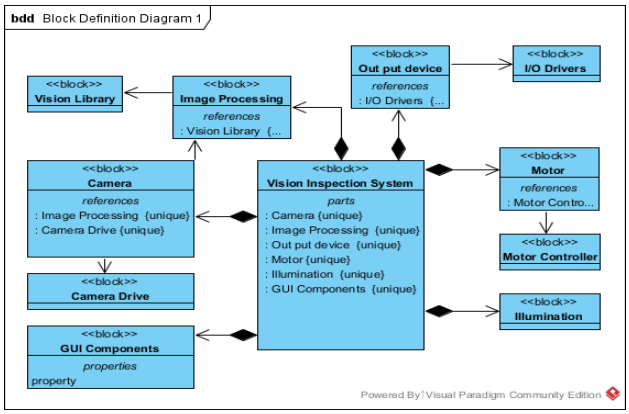


Figure 6: An example Block definition diagram

Software Design: this phase emphasizes more on the design of the solution. We adopted the following two phases of the COMET method:

- ***High Level Design:*** at this point the system is separated into subsystems in terms of components and interfaces using the COMET subsystem structuring criteria as a package diagram in Figure 7. The approach considers the concurrent tasks of the industrial computer vision systems and the object oriented concepts of information hiding, classes and inheritance. The development framework follows the COMET concurrency guidelines principles and use architectural structure patterns, communication patterns and design patterns to avoid unnecessary redesign effort and activities.

Furthermore, the parametric diagram represent the relationship between the system constraints. The constraints are important to develop a rule based specification that describes the behavior, performance and other constraints. The allocation diagram indicates the relationship between the modeling elements of the system. Specifying the allocation of the system parts is used to understand the inter-relationship among the functionalities, software modules and hardware blocks.

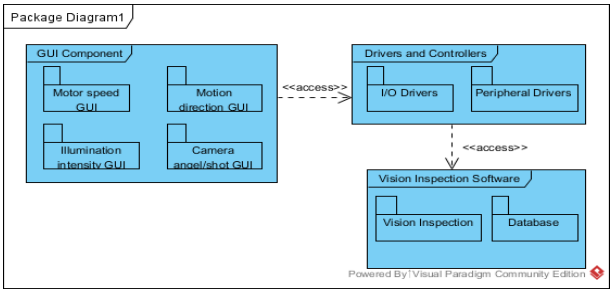


Figure 7: A Package diagram of an inspection system

- ***Low Level Design:*** this section describes how the system should be implemented at low level. The algorithm details of the software components are defined using a *Program Design Language* (PDL) notation, which is also known as Structure English or Pseudocode. This is done based on the object and class structuring criteria of the COMET method. Therefore, the system components can be explained in detailed to create good understanding without using a specific program language; but, this makes simple to map the software components to the implementation language.

As an example of the low level design, we showed the detail design of the Inspection class as follows. The Inspection class is one of the four subclasses that specialized the Process class. *Figure 8* shows the abstract Process superclass and the four subclasses, which are part of the vision inspection module.

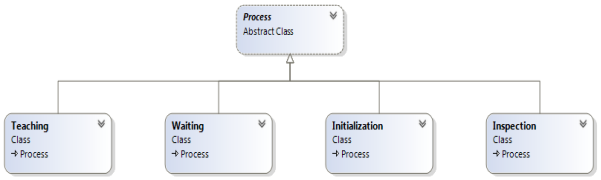


Figure 8: Example abstract superclass and subclasses

The Inspection class interface and its operations are described as follows using PDL:

Information Hiding Class: Inspection Class

Information Hidden: Encapsulate the Motor axis control attributes and their values.

Class structure criteria: Control class

Assumptions: Checking the image

Superclass: Process (Abstract Class)

Inherited operations: Command, Do, End, Init

Operations provided:

1. ***Inspection*** (in axis : AxisController, processManager : processManager, mModelName : string)
   - *Function*: Assign the value of the axis
   - *Precondition*: The three inputs has to be provided
   - *Post condition*: Assign the value of the axis
   - *Input parameters*: axis, processManager, modelName
   - *Operation used*: None
2. ***InspectMove () :*** Void
   - *Function:* Acquiring the illuminated images
   - *Precondition*: object should be under the camera
   - *Input parameter*: None
   - *Output parameters*: None
   - *Operations used*: ChangeValue, WaitCommand, SendCommand
3. ***InspectPost(in SocketServer ss):*** void
   - *Function:* Analysis the image to store
   - *Precondition*: the location should be identified
   - *Invariant*: the ID of the image must unchanged
   - *Post condition*: analysis, sane and write
   - *Input parameter*: SocketServer
   - *Output parameter*: void
   - *Operations used*: GetAverage, GetInstance

Coding and Unit Testing: the framework adopted the incremental software construction process of the V-shape process and COMET method. Each iteration of the incremental software construction covers a detail design, coding and unit testing of a subset of the use cases. At the end of each iteration, the software product is verified and validated before proceeding to the next iteration.

Integration and System Testing: it includes the integration and system testing of the system. The V-shape process provides an explicit phase for validation and verification, which emphasize on the quality of the software artifacts. To facilitate this, test cases are generated from the use case and state chart diagram for validation of the unit and integration test.

5. Results and Discussions

This study indicates the use of the SysML language, COMET method and V-shape process to enhance the development of a computer vision system. The combination of the SysML and COMET is used to handle all the system requirements at the early stage and to improve the communication among the practitioner. In addition, the V-shape process is used to control and management the activities and tasks. The framework maps the activities and tasks with the SysML diagrams to benefit from the advantage of the mode driven software engineering.

The framework begins with requirement definition and followed by the modeling of the system structure and behavior. Then, it defines the constraints, parametric and internal detail design of the system to implement the software modules. The SysML diagrams helps the developers to understand the system

hardware and software requirements. The use of the structural criteria such as objects and subsystem structure criteria of the COMET simplifies the analysis and design of the system. Although COMET uses UML, SysML is more preferable in this study due to the hardware oriented nature of the industrial computer vision system. SysML has the advantage of representing the structural behavior, dynamic behavior, software, hardware and data of a system at different abstraction level.

Though the framework is evaluated by applying it to develop a computer vison quality inspection system, we include only few examples to make suit the size of the paper for this conference. However, the result proves that the framework is good enough to support the activities and tasks of an industrial computer vision system. Generally, the purpose of the framework is to show the benefits of the SysML modeling language, COMET method and V-shape process in computer vision system development. *Table 1* shows the comparison of our framework with existing studies.

Table 1: Comparison to related studies

Criteria	***Our approach***	***Existing approaches***
Objective	To create a framework that supports both the system and software engineering	To show the benefit of the model driven approach [8]
		To show the use of SysML [9, 10, 11]
Techniques used	V-shape process, COMET method and SysML modeling language	W-process and SysML models [8]
		SysML models [9, 10, 11]
Framework	Identify the activities, tasks and the SysML diagrams at each phase	No

6. Conclusion and Future work

In this study, we proposed a software development firework based on SysML, COMET and V-shape process, which support an effective design and developed of a computer vision system. The framework supports the entire life cycle of the system and software development process that lead to reusable, reliable and consistent software models. Most importantly, the SysML diagrams and the COMET guidelines principles create an easy communication and understanding among the practitioners at the early stage of the development process.

In the future, we would like to present how to apply the development framework using a full computer vision software development project. By doing this, we can show the detail of the framework more clearly and visibly.

References

[1] Erick Saldana, Raul Siche, Mariano Lujan, Roberto Quevedo, *Review: Computer Vision Applied to the Inspection and Quality Control of Fruits and Vegetables*, Brazilian Journal of Food Technology, Vol. 16, No. 4, pp. 254-272, 2013.

[2] http://www.bmva.org/visionoverview

[3] http://en.wikipedia.org/wiki/Computer_vision

[4] Zhoushan, Zhejiang, *Research on the Rule of Evaluation of Software Development Process model*, IEEE International Conference, 2010.

[5] Roger S Pressman, *Software Engineering: A Practitioner's Approached*, Ed 7th, New York: McGraw-Hill, pp 77-99, 2005.

[6] S. Satorres Martinez, J.Gomez Ortega et al, *An industrial Vision System for Surface Quality Inspection of Transparent Part*, International Journal Adv. Manufacturing Technology, 2013.

[7] Alvaro Rodriguez Tajes, *A Methodology to Develop Computer Vison System in Civil Engineering: Application in Material Testing and Fish Tracking* (Doctor Thesis), 2014.

[8] Giacomo Barbieri, Cesare Fantuzzi, Roberto Borsari. *A model-based design methodology for the development of mechatronic system, Model-Based Mechatronic System Design*, Vol. 24, Issue 7, 2014.

[9] Yue Cao, Yusheng Liu et al., *Sys-based uniform behavior modeling and automated mapping of design and simulation for mechatronics*, Elsevier, 2013.

[10] Bassi L, Secchi C, Bonfe M, Fantuzzi C. A SysML based methodology for manufacturing machinery modeling and design. Mechatronics International Journal, 2011.

[11] Chami M, Bou Ammar H, Voos H, Tuyls K, Weiss G., *A nonparametric evaluation of SysML-based mechatronic conceptual design*. In: Conference on artificial intelligence; 2012.

[12] Software Engineering Center, *ESPR Embedded System Development Process Reference Guide*, Information-Technology Promotion Agency, Japan, 2012.

[13] Krishna Kumar Patel, A. Kar, S.N.Jha, M.A.Khan, *Machine Vision System: a tool for quality inspection of food and agriculture products*, Journal of Food Science and Technology, pp. 123-141, 2012.

구문적 노이즈 제거를 이용한 소프트웨어 아키텍처 복원 자동화 프레임워크

이기성, 이찬근

중앙대학교 창의 ICT 공과대학 컴퓨터공학부
서울특별시 동작구 흑석로 84
goory00@gmail.com, cglee@cau.ac.kr

요약: 소프트웨어 아키텍처 복원은 소프트웨어 저장소로부터 획득 가능한 정보를 이용하여 아키텍처 모듈-뷰를 생성하는 재공학 기술이다. 본 논문은 아키텍처 모듈-뷰 복원을 위한 프레임워크를 제안하며 기존 아키텍처 복원 연구에서 다루지 않았던 문제를 해결하고자 한다. 이와 관련하여 (1)시맨틱 노이즈 제거를 통한 복원 정확도 향상, (2)계층적 아키텍처 구조의 비교평가, (3)진화적 소프트웨어의 모듈성 품질 측정에 대한 이론적, 실증적 접근을 다룬다.

핵심어: 아키텍처 복원, 시맨틱 품질, 노이즈 제거, 계층적 구조 비교, 모듈성 측정

1. 연구배경 및 제안

소프트웨어 아키텍처 문서는 고수준의 소프트웨어 구조를 제공해 주지만, 시간이 지남에 따라 적절히 업데이트 되지 않거나 변질 혹은 분실되기 쉽다. 이에 소프트웨어 저장소로부터 데이터를 분석하여 아키텍처 모듈-뷰를 복원하는 연구가 활발하다[3].

그러나 기존 연구의 문제점으로, (1)소프트웨어의 의미적 관계를 고려한 복원은 코드의 구문적 품질에 민감하게 영향을 받지만 기존 연구들은 품질에 대한 고려가 없었다. (2)소프트웨어 아키텍처는 계층적인 형태임에도 불구하고 대부분의 연구가 평면 구조를 대상으로 복원 결과를 평가하였다. (3)아키텍처의 모듈성 품질 측정을 위한 공통된 합의가 존재하지 않아 모듈성 품질을 객관적으로 평가하기 어렵다.

본 연구는 소프트웨어 아키텍처 복원 자동화 프레임워크를 제안하며, 구문적 정보에 기반하여 아키텍처를 복원하며, 복원된 아키텍처의 모듈성 품질을 평가한다. 또한 프레임워크 내 각 단계에서는 앞서 언급한 한계점들의 해결방안을 다룬다. 그림 1 은 제안하는 아키텍처 복원 프레임워크의 개요이다.

2. 소스코드 분석 및 클러스터링

대부분의 아키텍처 복원 연구는 엔티티 간 구조적, 구문적, 혹은 논리적(개발활동 관련) 관계를 고려한다. 본 연구는 구문적 정보를 분석 대상으로 하며, 코드 내 구문정보를 추출하여 코드 간 시맨틱 유사도를 측정한다. 소스코드의 시맨틱 정보 분석에는

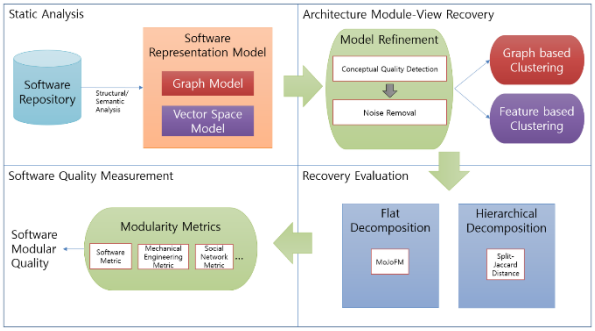


그림 1 아키텍처 모듈-뷰 복원 프레임워크 개요

LDA(Latent Dirichlet Allocation)을 사용하며, 이를 통해 각 문서가 관심있게 다루는 토픽의 분포를 추출할 수 있다. 표 1 은 토픽 분포의 예시를 보여준다. Doc는 소스코드를 의미하며 각 Doc가 5개의 토픽에 대하여 어느 정도의 관련성을 갖는지 파악한다.

표 1 토픽 분포 예시

	Topic1	*Topic2*	*Topic3*	*Topic4*	*Topic5*
Doc1	0.5	0.1	0.1	0.2	0.1
Doc2	0.1	0.6	0.1	0.1	0.1
Doc3	0.2	0.2	0.1	0.3	0.2

위와 같은 토픽 정보를 토대로 유사한 소스코드끼리 그룹화하여 모듈-뷰를 복원한다. 유사도 계산에는 코사인 유사도를 사용하며, 복원을 위한 클러스터링에는 그래프 기반 클러스터링과 특성(feature) 벡터 기반 클러스터링을 모두 활용한다.

3. 시맨틱 품질 측정을 이용한 복원 향상

구문적 정보 기반 소프트웨어 분석은 코드내의 클래스 명, 메소드 명, 변수 명, 주석 등의 정보를 분석한다. 그러나, 코드 명명은 개발자의 판단에 의해 결정되므로 그 품질이 상이하다. 따라서 이를 정량화할 필요가 있으며 본 연구는 코드의 시맨틱 품질을 측정하기 위해 CQ(Conceptual Quality)를 제안한다.

$$CQ(D_i) = \sum_{k=1}^{h} ConceptualWeight(P(D_i), HighRankedT(D_i, k)) \times T^{D_i}_{rank_k}$$

이는 LDA 로 추출한 각 소스코드의 토픽 분포 중 상위랭크 토픽 비율 $T^{D_i}_{rank_k}$ 와, 해당 토픽이 패키지에서 차지하는 비율인 *Conceptual Weight* 를 가중치로 하여 계산된다. 소스코드의 상위랭크 토픽은 해당 문서가

어떠한 컨셉트를 설명하고 있는지를 대변해 주므로 이러한 상위랭크 토픽이 현재 소스코드가 존재하는 패키지의 컨셉트와 얼마나 관련성이 있는지 측정한다. CQ 값이 크면 소스코드와 패키지의 컨셉트가 서로 일치함을 나타내며 반면 값이 작으면 소스코드의 의미적 내용이 패키지의 컨셉트와는 이질적임을 나타낸다. 본 연구는 CQ 를 시맨틱 품질측정 도구로 사용하여 시맨틱 품질이 낮은 소스코드들을 판별한다.

평가를 위해 의도적으로 생성된 구문적 저품질 문서 3 종류(A~C.java)를 Hadoop-core 0.19.0 에 삽입후 시맨틱 품질을 측정하였다. A.java 는 위키피디아에서 java 검색으로 나타난 일반문서이며, B.java 는 클래스명, 변수명 등이 의미없는 단어로 수정된 파일이다. 반면 C.java 는 정상적인 java 파일이지만 전혀 다른 성격의 패키지로부터 위치를 옮겼다.

비교를 위해 CQ 외에 k-NN(k-Nearest Neighbors) 기반 이상치 검출을 시도하였으며 이는 전체 문서에서 이질적인 성격의 문서를 파악해준다. 실험 결과는 표 2 와 같이 CQ 의 문서 품질 평가 정확도가 비교적 높은 것으로 나타났다.

표 2 CQ 와 k-NN 이상치 검출 비교

File	*CQ* rank	*k-NN* based outlier rank		
		$k = 10$	$k = N/2$	$k = N\text{-}1$
A.java	**1**	1	1	1
B.java	**5**	32	8	6
C.java	**27**	57	51	60

또한 시맨틱 품질을 측정하여 아키텍처 복원에 응용하였다. 시맨틱 품질이 낮은 문서를 제거한 후 복원을 수행한 결과 복원 결과가 향상되는 효과를 얻었다. 그림 2 와 같이 노이즈(저품질 문서) 제거 비율 0.7 구간 이하에서는 노이즈 제거율을 높일수록 복원 정확도(MojoFM[5] 수치)가 향상되고 있다.

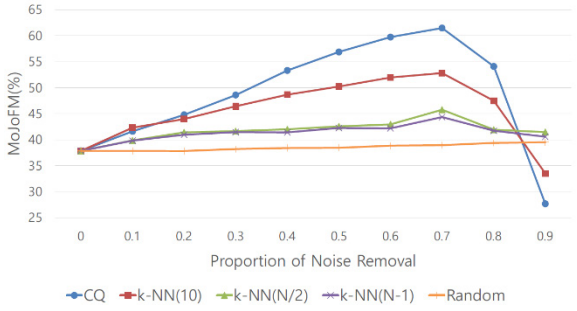


그림 2 아키텍처 복원시 노이즈 제거 효과 비교

그림 2 에서 시맨틱 품질을 전혀 고려하지 않은 랜덤 문서 제거는 복원 정확도 향상에 전혀 기여하지 못함을 보여준다. 반면 k-NN 으로 찾아낸 저품질 문서의 제거는 복원 정확도를 어느정도 높이고 있으며 k 값이 작게 설정된 경우 더 좋은 성능을 보인다. 연구에서 제안하는 CQ 의 경우 노이즈 제거를 이용한 아키텍처 복원 정확도 향상에 가장 좋은 성능을 보인다. 이는 전체 문서를 고려한 노이즈 제거보다 패키지 단위를 고려한 노이즈 제거 방식이 더 효과적임을 보여준다.

4. 계층적 아키텍처를 고려한 복원 평가

일반적으로 복원 정확도 평가를 위해서는 전문가가 분석한 정답지와 비교하며, 가장 많이 사용되는 것이 mojo 계열 비교 메트릭이다. 그러나 이 기법은 각 그룹을 평면 계층에서 비교하므로 소프트웨어와 같은 계층적 구조를 간과한다. 반면 Tree edit distance 는 계층화 되어있지만 노드의 순서가 중요한 영향을 미치므로 소프트웨어에 적합하지 않다.

본 연구에서는 계층적인 아키텍처 비교를 위해 비정렬 트리에 기반하는 스플릿-자카드 거리 (Split-Jaccard Distance) [2]를 제안한다. 스플릿[4]은 두 노드가 공통으로 갖는 가장 가까운 부모(혹은 조상) 노드를 의미하며, 스플릿을 루트(root)로 하는 서브트리간 유사도를 비교하여 두 아키텍처 구조의 거리를 측정한다. 이는 다음과 같은 식으로 표현할 수 있다.

$$D(T_1, T_2) = 1 - \frac{1}{\binom{n}{2}} \sum_{i=l_1}^{l_n} \sum_{j=i}^{l_n} d(i, j\,; T_1, T_2)$$

앞의 식에서 $d(i,j;T_1,T_2)$는 노드 i 와 노드 j 의 스플릿을 루트로 하는 T_1 의 서브트리와 T_2 의 서브트리 간 자카드 유사도를 나타하며 이 값의 총합을 0~1 사이로 정규화한 후 1 에서 뺌으로써 거리화 한다. 해당 수식은 계층화된 두 구조간 차이를 정량화하며 수학적 메트릭(metric)으로 사용 가능하다.

스플릿-자카드 거리는 두 트리간 계층적 구조의 상이함을 측정할 수 있으므로 소프트웨어 비교에 효과적이다. 현재는 계층정보를 담고있는 참-모듈 뷰의 부재로 인해 실질적 적용을 하지 못하고 있으나, 계층화된 복원 결과와 현재 아키텍처 구조와의 편차 측정 등에 응용될 수 있다.

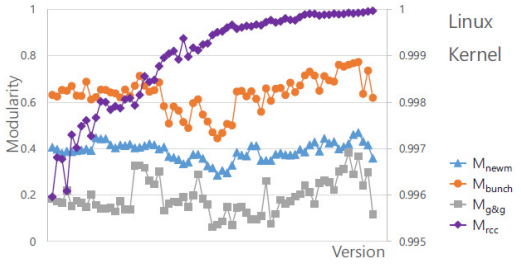
5. 소프트웨어 모듈성 메트릭 비교

복원된 아키텍처를 이용해 모듈성 품질을 평가할 수 있으나 모듈성 측정 방법은 매우 다양하다. 본 연구는 모듈성 측정을 위해 제안된 다양한 메트릭들을 이용하여 소프트웨어의 모듈성을 측정하여 비교한다. 표 3 은 논문에서 고려한 모듈성 메트릭을 보여준다.

표 3 모듈성 측정 메트릭

명칭	모듈성 측정식	요약
M_{newm}	$\frac{1}{2m}\sum_i\sum_j\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(g_i, g_j)$	사회적 네트워크 모듈성[6]
M_{bunch}	$\sum_{i=1}^{k} \frac{2\mu_i}{2\mu_i + \sum_{j=1}^{k}(\epsilon_{i,j} + \epsilon_{j,i})}$	소프트웨어 모듈성[7]
$M_{g\&g}$	$\frac{\sum_{i=n_k}^{m_k}\sum_{j=n_k}^{m_k} R_{ij}}{\sum_{k=1}^{M}(m_k - n_k + 1)^2} - \frac{\sum_{i=n_k}^{m_k}\left(\sum_{j=1}^{n_k-1} R_{ij} + \sum_{j=m_k+1}^{N} R_{ij}\right)}{(m_k - n_k + 1)(N - m_k + n_k - 1)}}{M}$	기계제품 모듈성[8]
M_{rcc}	$1 - \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} DependencyCost(i,j)}{N^{2\lambda}}$	소프트웨어 클러스터 비용[9]

실험을 위해 각기 다른 모듈성 메트릭을 이용하여 리눅스 커널의 67 개 버전의 모듈성을 측정하였다. 먼저 각 버전마다 소스파일 간 유사도를 이용하여 아키텍처 모듈-뷰 복원을 수행하였고, 복원 결과를 대상으로 모듈성을 측정하였다. 측정을 위한 도구는 java 언어를 이용하여 직접 제작하였다. 그림 3 은 리눅스의 진화적 모듈성 변화와 모듈성 메트릭간 상관계수를 보여준다[1].



Project		M_{newm}	M_{bunch}	$M_{g\&g}$	M_{rcc}
Linux Kernel	M_{newm}	–	0.783**	0.492**	-0.171
	M_{bunch}	0.783**	–	0.670**	0.059
	$M_{g\&g}$	0.492**	0.670**	–	0.089
	M_{rcc}	-0.171	0.059	0.089	–

그림 3 리눅스의 모듈성 변화 및 모듈성 상관성

그림 3 에서 M_{newm}, M_{bunch} 의 변화가 유사하며 피어슨 상관계수가 0.7 이상으로 높게 나타난다. 또한 $M_{g\&g}$ 역시 앞선 메트릭들과 유사한 성향을 가진다. 반면 M_{rcc} 는 모듈성 평가 기준이 상이하여 서로 관계성이 적은 것으로 파악된다. 위와 같이 각 모듈성 메트릭의 반응성이 조금씩 다르므로 단일 메트릭에 의존적인 평가 보다는 종합적인 측정이 바람직하다.

6. 결 론

본 논문은 소프트웨어 아키텍처 복원을 위한 프레임워크를 제안하였다. 이와 관련하여 구문적 노이즈 제거를 이용하여 복원 정확도를 향상시키고, 소프트웨어에 적합한 계층적 비교 메트릭을 제안하였으며, 소프트웨어의 진화적 모듈성 품질 측정 및 비교를 수행한 기여를 갖는다.

감사의 글

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성 지원사업(IITP-2015-H8501-15-1012)과 한국연구재단 기초연구사업(과제번호 NRF-2014-005519)의 지원을 받았습니다.

참고문헌

[1] K. S. Lee and C. G. Lee, “Comparative Analysis of Modularity Metrics for Evaluating Evolutionary Software,” IEICE TIS, Vol. E98-D, No. 2, pp. 439–443, 2015.

[2] K. S. Lee et al., “Split-Jaccard Distance of Hierarchical Decompositions for Software Architecture,” IEICE TIS, Vol. E98-D, No. 3, pp. 712–716, 2015.

[3] M. Shtern and V. Tzerpos, “Clustering Methodologies for Software Engineering,” Hindawi ASE, Vol. 2012, No. 1, 2012.

[4] Q. Zhang et al. “Split-Order Distance for Clustering and Classification Hierarchies,” SSDBM, Vol. 5566, pp. 517–534, 2009.

[5] Z. Wen and V. Tzerpos, “An effectiveness measure for software clustering algorithms,” Proc. IWPC, pp.194-203, 2004.

[6] M. E. J. Newman, “Modularity and community structure in networks,” Proc. NAS of USA, pp. 8577-8582, 2006.

[7] S. Mancoridis et al., “Bunch: A clustering tool for the recovery and maintenance of software system structures,” Proc. IEEE ICSM, pp. 50-59, 1999.

[8] F. Guo and J. K. Gershenson, “A Comparison of Modular Product Design Methods on Improvement and Iteration,” Proc. ASME, pp. 261-269, 2004.

[9] R. Milev et al., “Design evolution of an open source project using an improved modularity metric,” Proc. IFIP, pp. 20-33, 2009.

실시간 연동을 위한 LVC 통합연동시스템 아키텍처 설계

오지현†, 김천영, 장영찬, 지철규, 홍영석
국방과학연구소

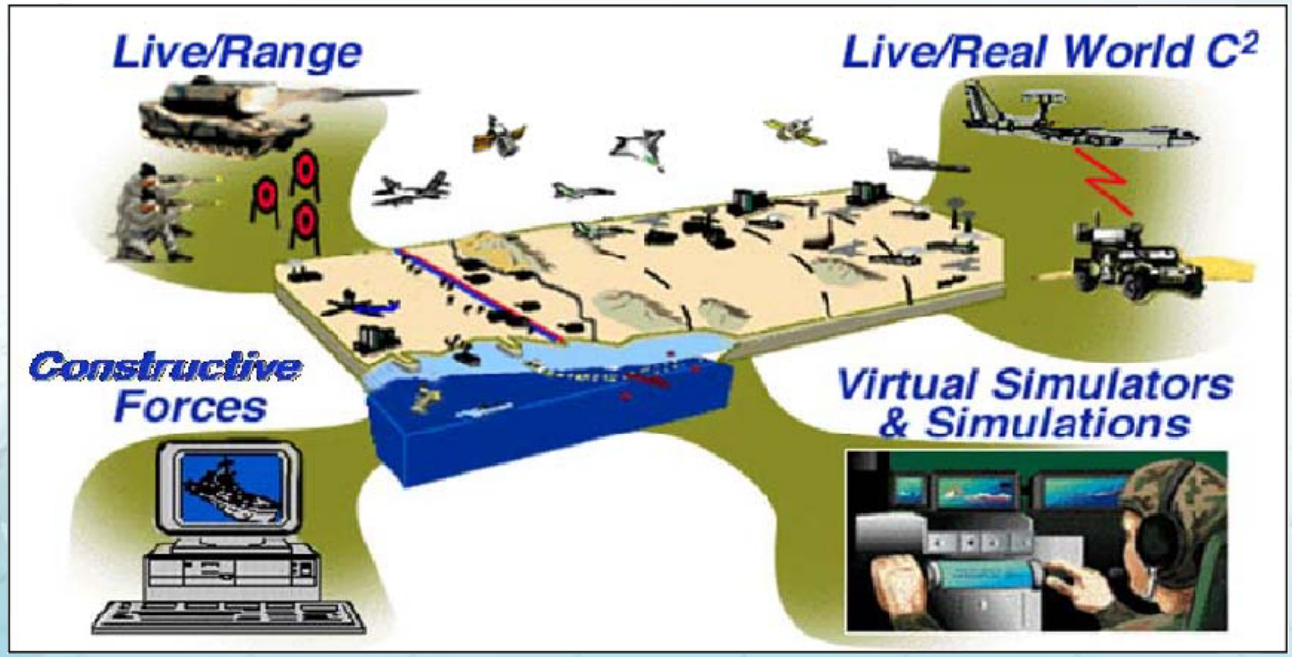
† 오지현 : 국방과학연구소
ohjh@add.re.kr

국 방 과 학 연 구 소
AGENCY FOR DEFENSE DEVELOPMENT

목 차

서론

- ‘LVC 통합연동시스템’ 이란?
  - Live, Virtual, Constructive 각 체계를 하나의 훈련환경으로 통합 운용하기 위한 연동시스템
    - Live : 실기동 체계 (유인/무인 전투기 등)
    - Virtual : 시뮬레이터 장비
    - Constructive : 임무 및 교전 모의 모델, Game engine 등



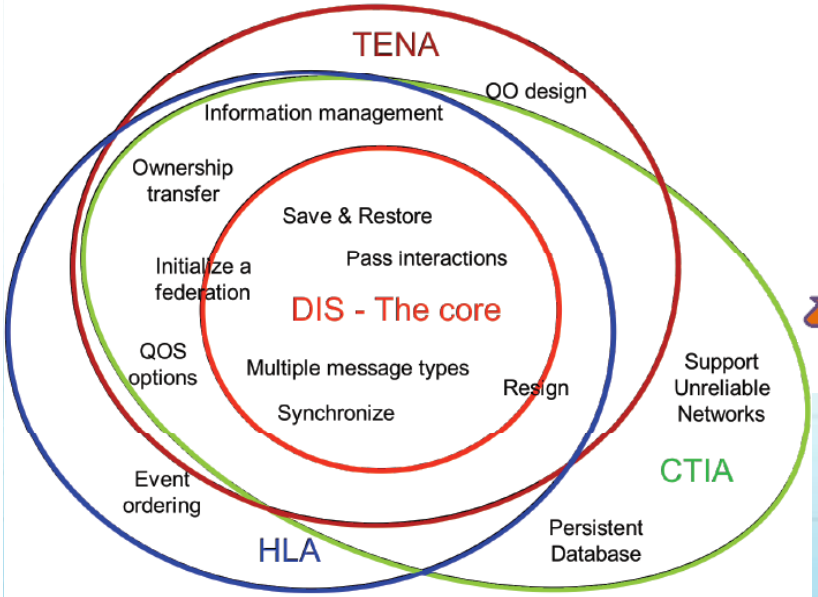
서론

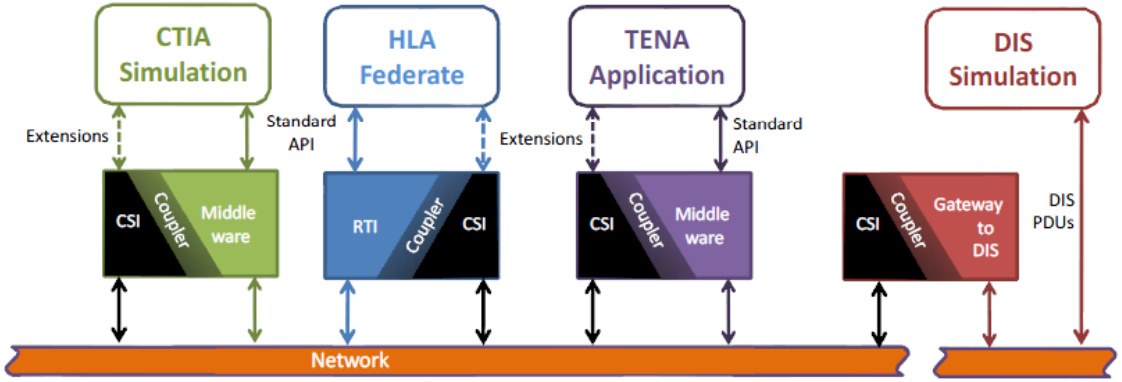
- LVC 통합연동시스템의 필요성
  - 훈련 체계에서 실기동급의 Live 훈련은 훈련의 범위에 제약이 있으며, 고비용 및 위험이 따름
  - 실기동급 훈련만의 제약을 극복하고자 Virtual 및 Constructive 체계를 연동한 LVC 합성전장환경의 전투 실험 및 훈련체계 구축이 요구되고 있음
  - LVC 통합연동시스템을 통해 독립적으로 운용되고 있는 각 LVC 체계를 하나의 합성전장훈련환경으로 제공할 수 있음

- 실시간 연동을 위한 LVC 통합연동시스템 아키텍처 설계
  - 기존에 독립적으로 운영되었던 L, V, C 체계를 통합 운용하기 위하여 각 운용 환경을 분석하고 상호운용하기 위한 통합 아키텍처가 필요함.

국내외 개발 동향

- 상이한 환경에서 운용되는 LVC 체계를 연동하기 위한 통합 아키텍처에 대한 표준 제정과 이를 활용한 개발 및 적용 연구가 국내외로 활발히 진행중
- 분산 환경에서의 상호 운용성을 지원하기 하기 위한 아키텍처로서 HLA/RTI, DIS, TENA 등이 있으며, 단일 아키텍처를 연동하기 위한 Multi-Architecture에 대한 연구 진행중





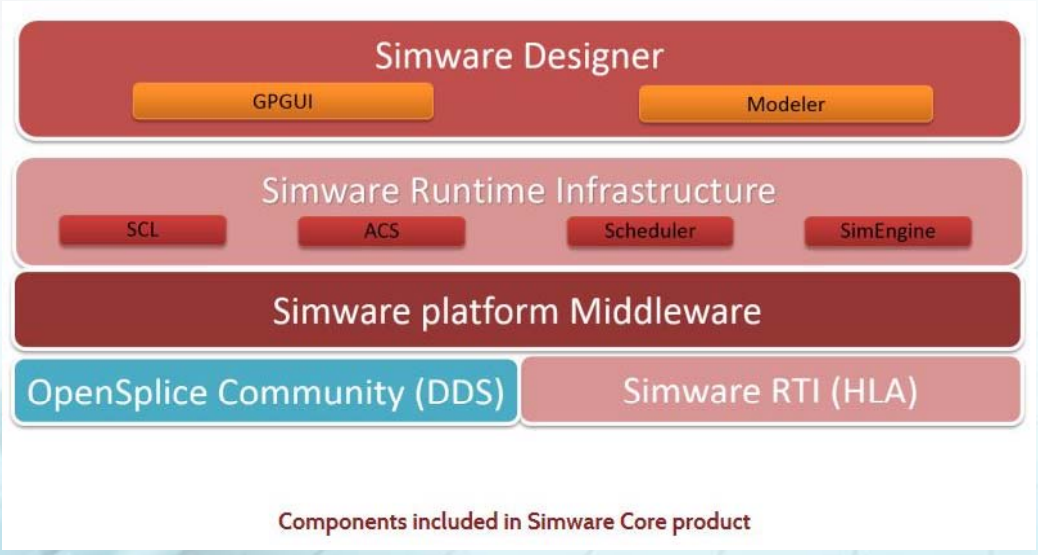
Common Distributed Architecture Overview

※ HLA/RTI : High Level Architecture/Run-time Infrastructure
※ DIS : Distributed Interactive Simulation
※ TENA : Test and training Enabling Architecture

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

국내외 개발 동향 – 미국

- **Simware – NADS**
  - **LSA(Layered Simulation Architecture) 기반의 Data-Centric Architecture를 통한 다양한 플랫폼을 연동**
    - **DDS, HLA, CBML, JAUS 등**
  - **Live, Virtual, Constructive 연동**



Components included in Simware Core product

※ DDS : Data Distribution Service
※ CBML : Coalition Battle Management Langugate
※ JAUS : Joint Architecture For Unmanned Systems

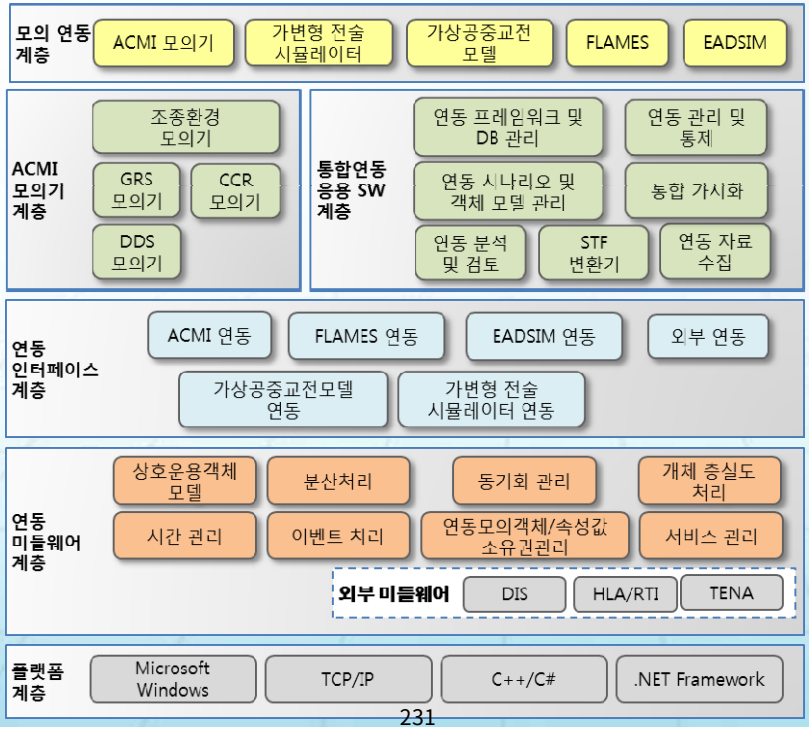
국내외 개발 동향 – 유럽

- **IMAGE**
  - 2003년 프랑스 IRIT 연구소를 중심으로 European project 시작
  - 이기종 시뮬레이션 간의 상호운용성을 위한 LVC 연동 미들웨어를 개발
  - CORBA, HLA 기반의 미들웨어
    - 이기종 시뮬레이션 사이의 연동 지원
    - 동기 및 비동기 통제
    - 분산된 시뮬레이션 참가자의 각 객체 상태를 동일한 합성환경으로 공유
    - 시뮬레이션들 사이의 데이터 교환
    - 분산 환경에서 운용되는 각 시뮬레이션 모델의 성능을 유지

※ CORBA : Common Object Request Broker Architecture

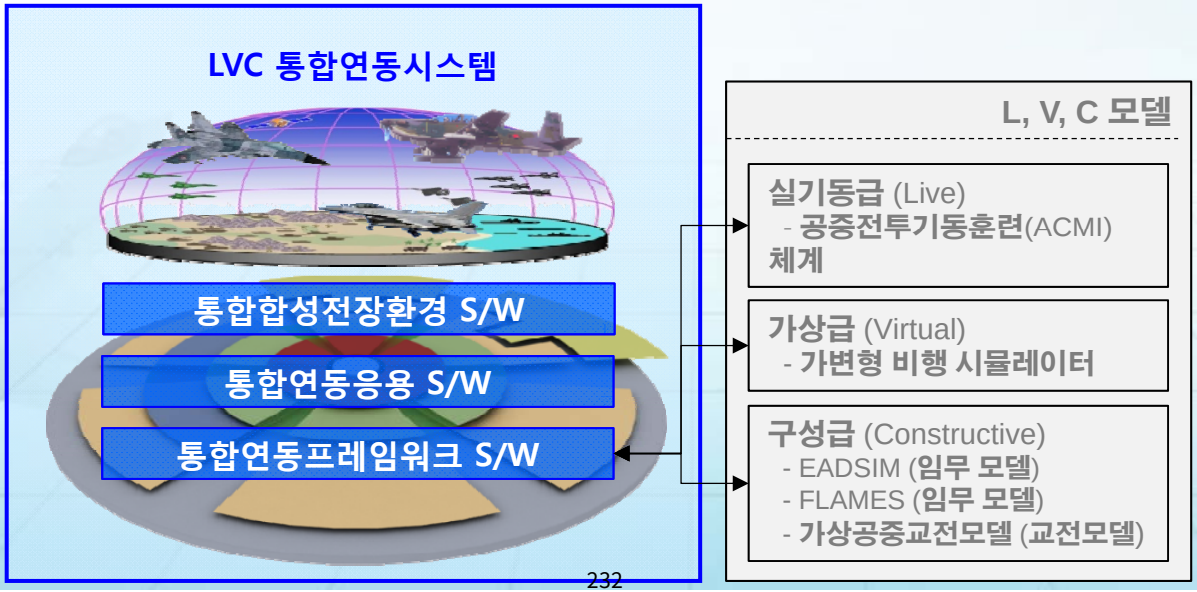
국내외 개발 동향 – 한국

- **항공무기체계 임무/가상/ACMI 시뮬레이션 연동 기술**
  - '12 ~ '14 국방과학연구소에서 항공무기체계에서의 LVC 연동 기술 응용연구 수행을 통해 항공무기체계 LVC 통합 아키텍처 기술을 확보
  - LVC 연동기반의 통합훈련환경을 구축하고, 공대공/공대지 훈련 시나리오를 작성하여 통합훈련을 수행
  - 한국 공군의 훈련체계에 적용하기 위한 다음 단계 사업을 진행 중



LVC 통합연동시스템 – 운용 개념

- 기 운용 모델 및 상용 모델을 하나의 합성전장훈련환경으로 통합 운용
  - 실시간으로 운용되는 Live와 Virtual, Constructive와의 연동을 수행
- 상이한 모델간의 교전 모의, 무장의 피해평가 처리 및 시뮬레이션 정보 처리를 위한 연동 프레임워크 구현
- LVC 통합연동시스템의 운용 지원을 위한 응용 S/W 와 합성전장환경표현을 위한 S/W 제공



LVC 통합연동시스템 – 연동 환경

- **ACMI 체계**
  - 비행 훈련의 실시간 및 사후 브리핑을 위한 것으로 전용 ICD를 구현하여 CCR 모의기, GRS 모의기, 및 DDS 모의기와의 데이터 교환을 수행
  - CCR 모의기를 통하여 Live와 타 체계와의 연동을 지원
- **가변형 비행 시뮬레이터**
  - 내부 데이터 연동을 위한 전용 ICD를 구현하여 데이터 교환을 수행
  - 구성급 가상공중교전모델과의 연동을 위한 Bridge를 개발하여 VC 연동을 지원
- **구성급**
  - 시뮬레이션 사이의 상호연동을 위한 HLA/RTI, DIS 기반 연동을 지원

<table>
<tr><th rowspan="2">연동 지원 환경</th><th>Live</th><th>Virtual</th><th colspan="3">Constructive</th><th>Game</th></tr>
<tr><th>ACMI</th><th>가변형 비행 시뮬레이터</th><th>가상공중교전모델</th><th>FLAMES</th><th>EADSIM</th><th>VBS3</th></tr>
<tr><td rowspan="2">Data format</td><td rowspan="2">ICD</td><td rowspan="2">ICD</td><td>ViCS RPR2-D18v</td><td>FLAMES RPR2-D12v</td><td>SOM</td><td rowspan="2">PDU’s</td></tr>
<tr><td>PDU‘s</td><td>PDU‘s</td><td>PDU’s</td></tr>
<tr><td rowspan="3">Infrastructure</td><td rowspan="3">-</td><td rowspan="3">-</td><td rowspan="2">MAK RTI</td><td>MAK RTI</td><td rowspan="2">DMSO RTI</td><td rowspan="3">DIS</td></tr>
<tr><td>DMSO RTI</td></tr>
<tr><td>DIS</td><td>DIS</td><td>DIS</td></tr>
</table>

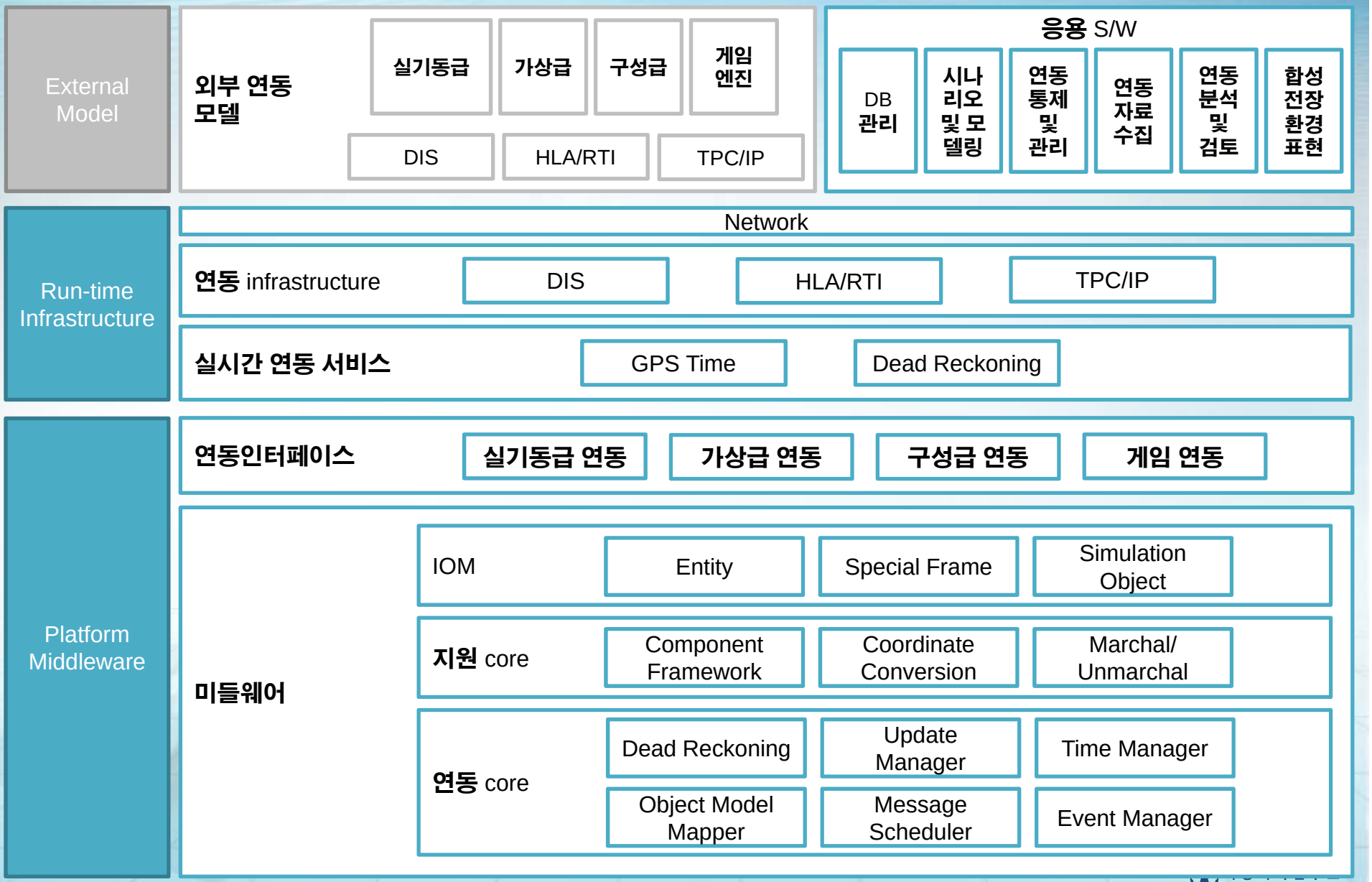
※ ICD : Interface Control Document
※ CCR : Central Control Room
※ GRS : Ground Relay Station
※ DDS : Display Debriefing Station
※ RPR : Real-time Platform Reference
※ PDUs : Protocol Data Units
※ SOM : Simulation Object Model

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

LVC 통합연동시스템 아키텍처 설계-1

- 실시간 연동 보장을 위한 LVC 통합연동시스템 아키텍처 설계
  - 지상의 GPS 정보 및 실시간 OS 기반의 시간 동기화 지원
- LVC 각 체계의 운용 플랫폼에 독립적인 미들웨어 설계
  - 각 모델의 연동 인터페이스 구현을 통한 상호운용성 지원
- 시뮬레이션에 참여한 각 LVC 체계에서 모의되는 객체 정보의 이벤트 및 시뮬레이션 정보를 처리하기 위한 미들웨어 기반의 아키텍처 설계

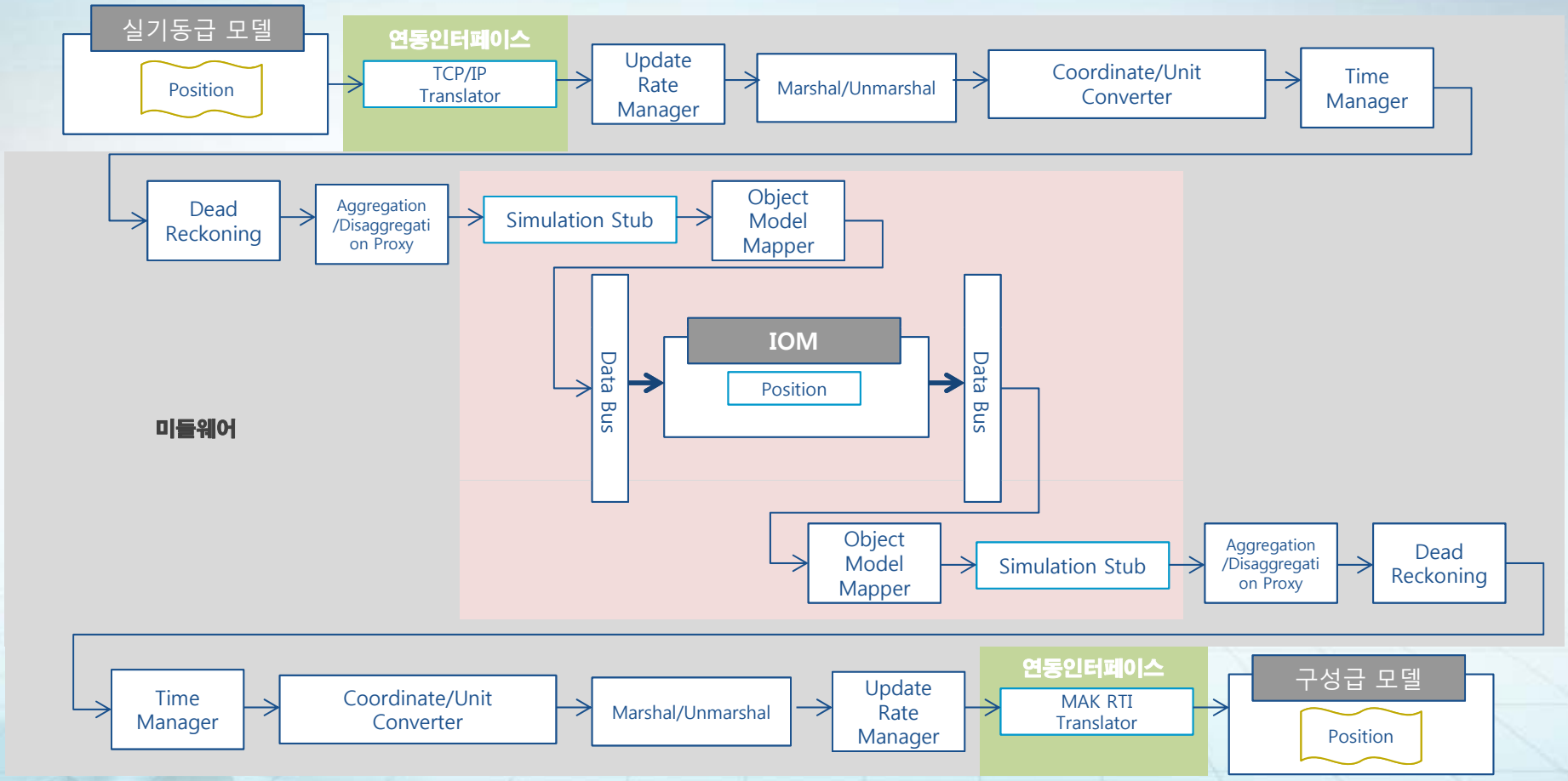
LVC 통합연동시스템 아키텍처 설계-2



※ IOM : Interoperability Object Model

LVC 통합연동시스템 아키텍처 설계-3

Platform middleware 프로세스 설계



향후 개발 계획 및 결론

- 본 연구를 통해 설계된 아키텍처를 기반으로 LVC 통합연동시스템을 개발
- 향후 요구되는 훈련 및 분석체계를 상호 운용할 수 있는 기반을 마련
- 현 공군에서 사용하는 LVC 체계를 단일 합성전장훈련환경으로 구현하여 향후 다양한 훈련체계 구축에 활용 예정

참고문헌

- Henninger A. E., et al. “Live Virtual Constructive Architecture Roadmap (LVCAR) Final Report”, Institute for Defense Analysies, M&S Co Project, 2008.09
- Richbourg R., et al. “Live Virtual Constructive Architecture Roadmap(LVCAR) Comparative Analysis of the Architectures”, M&S CO Project, 2008.09
- Saunders R., et al. “Live-Virtual-Constructive Architecture Roadmap Implementation Convergence Final Report”, Johns Hopkins University, 2010.06
- “Guide for Multi-Architecture Live-Virtual-Constructive Environment Engineering and Execution”, Johns Hopkins University, 2010.06
- www.simware.es
- Kim, C. Y., et al. “A Design of System Architecture for the Interoperability of Mission/Virtual/ACMI Simulation based on Aircraft Weapon System”, KSAS conference 2013.04.
- Kim, C. Y., et al. “A Development of an Integrating Architecture for the Interoperability System of Mission/Virtual/ACMI Simulation based on Aircraft Weapon System”, KSAS conference 2013.07.
- Jang, Y. C., et al. “A Design of Interoperability System based on Mission/Virtual/ACMI Simulation Integrating Architecture of Air Weapon system”, KIMST 2013.03.
- Oh, J. H., et al. “A Development of an Interoperability system based on Integrating Architecture for Mission/Virtual/ACMI Simulation on Aircraft Weapon System”, KIMST 2014.06.

KCSE 2016 논문집 18권 1호

무기체계 SW 신뢰성 통합관리 프레임워크 개발을 위한 프로세스에 관한 연구

A Research on the Process towards developing the Framework on Systematic Investigation for Reliability Improvement Upon Software(SIRIUS)

박삼준, 이태호, 백옥현*
오정섭, 서달미**

* 국방과학연구소 SW신뢰성기술실
** ㈜NSE 융합솔루션개발팀

순 서

연구 개요

연구 배경

- **무기체계 요구 기능 변화**
  - 무기체계의 복잡화, 첨단화, 무인화, 자율화, 네트워크화 추세
  - 무기체계 제반 기능의 상당 부분을 SW로 구현함에 따라 SW 비중/중요성 증가

- **SW 결함에 의한 사고**
  - 패트리어트 미사일 방어시스템(1991)
    - 걸프전에서 운영되던 미사일 방어시스템의 SW 버그로 미군 28명 사망
    - 수치를 누적하는 시스템 시계에서 작은 시간 오류발생으로 추적시스템 오동작 발생
  - European Space Agency의 Ariane 5(1996)
    - 발사 37초 후에 자체 폭발
    - 경로 변경과정에서 numerical overflow로 인하여 on-board computer가 잘못된 고도 전달
  - Some Recent Software Failures Caused by Software Bugs !!!
    - http://www.sereferences.com/software-failure-list.php

- **무기체계 소프트웨어의 결함 발생 시,**
  - 무기체계 파손
  - 인명 및 비용 손실 등 치명적인 결과 초래

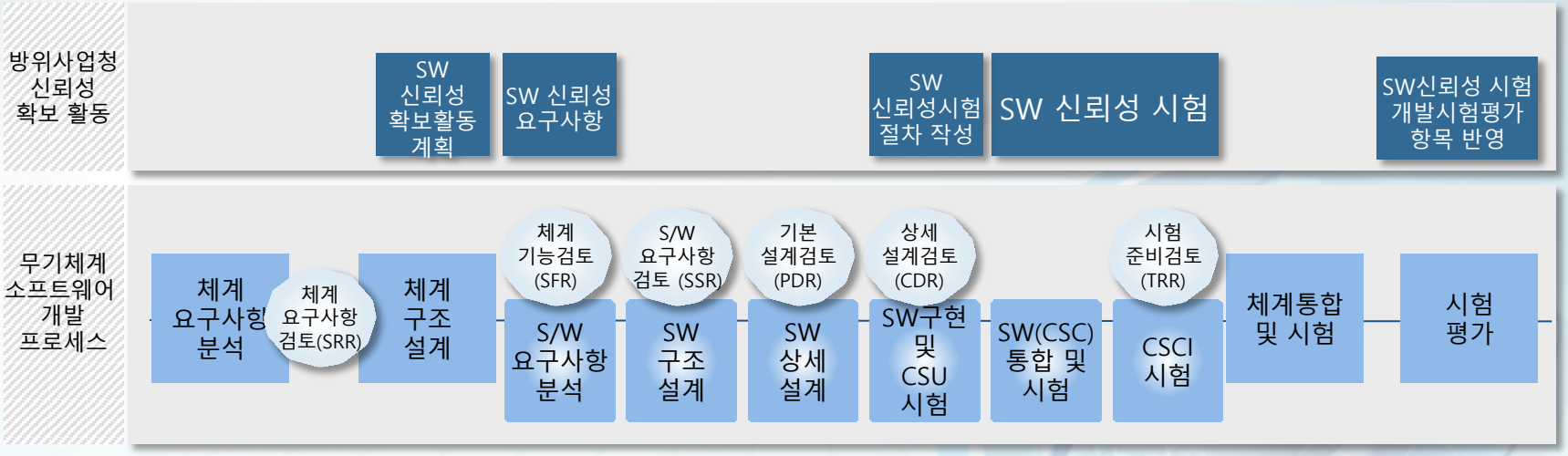
소프트웨어 신뢰성 확보로 결함의 최소화

SW 신뢰성 확보 활동

SW 신뢰성 확보 활동 관련 제도

- 명시 규정/지침서 : 방위사업관리규정, 무기체계 SW 개발 및 관리 매뉴얼, 실무지침서
- SW 신뢰성 확보 활동 : 무기체계 소프트웨어가 정확하고 일관성 있게 수행될 수 있도록 소스코드 내의 잠재적 결함을 최소화시키는 활동

☞ 구체적인 가이드라인 미제시



- SW 신뢰성 시험 : 소스코드에 대한 시험, 시험대상/기준 등 IPT 협의 결정
  - 코딩규칙
  - 실행시간오류
  - 코드실행률

구 분	내 용
정적시험 (Static)	SW의 소스코드 결함을 검출하는 코딩규칙 검증, 실행시간오류(Runtime Error) 검출 시험 등
동적시험 (Dynamic)	SW의 코드실행률(Code Coverage) 확인 시험

Motivation

- (現)SW 신뢰성 제도
  - 개발과정에서 정적/동적시험 수행에 따른 결함제거 긍정적 효과 존재
  - 정적/동적시험(SW 테스팅) 만으로는 SW 신뢰성 평가 불충분
    - SW 잔존 결함 수?
    - 시험 종료 시점?
    - SW 품질수준 파악 가능?
    - Resource problem? (개발기간, 시험예산 및 인력)
  - SW 신뢰성 확보를 위한 구체적인 활동 제시 부족

- SW신뢰성을 개발 전(全)단계에서 지속적으로 정량적 관리 필요
  - 신뢰성 척도 및 모델을 활용하여 SW 신뢰성을 분석/평가
  - 선진국 수준의 SW 신뢰성 통합관리 프레임워크 구축

사업개요

- **사업명**

  무기체계 SW 신뢰성 확보 및 검증 기술

- **사업목표**

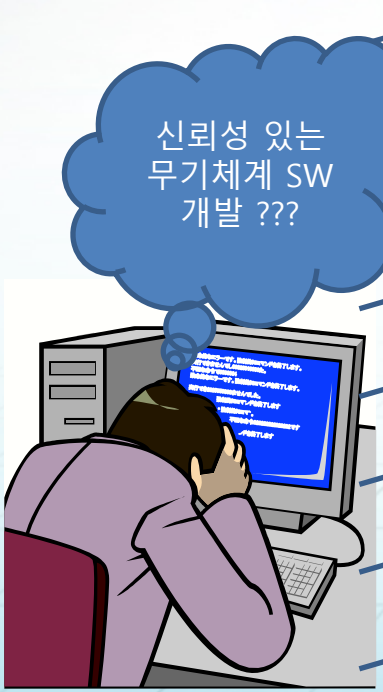
  무기체계 개발 전(全) 단계에서 SW 신뢰성의 정량적인 측정 및 평가를 위한 요소기술과 통합관리시스템을 개발하여 무기체계 SW 신뢰성 확보 및 검증 기반환경 구축

- **기간/예산/주관형태/사업단계**

  - 기간/예산 : ' 15. 10 ~ ' 18. 9 (36개월) / 00억원
  - 주관 형태 : 국과연 주관
  - 사업 단계 : 응용연구(핵심SW사업)

사업 필요성

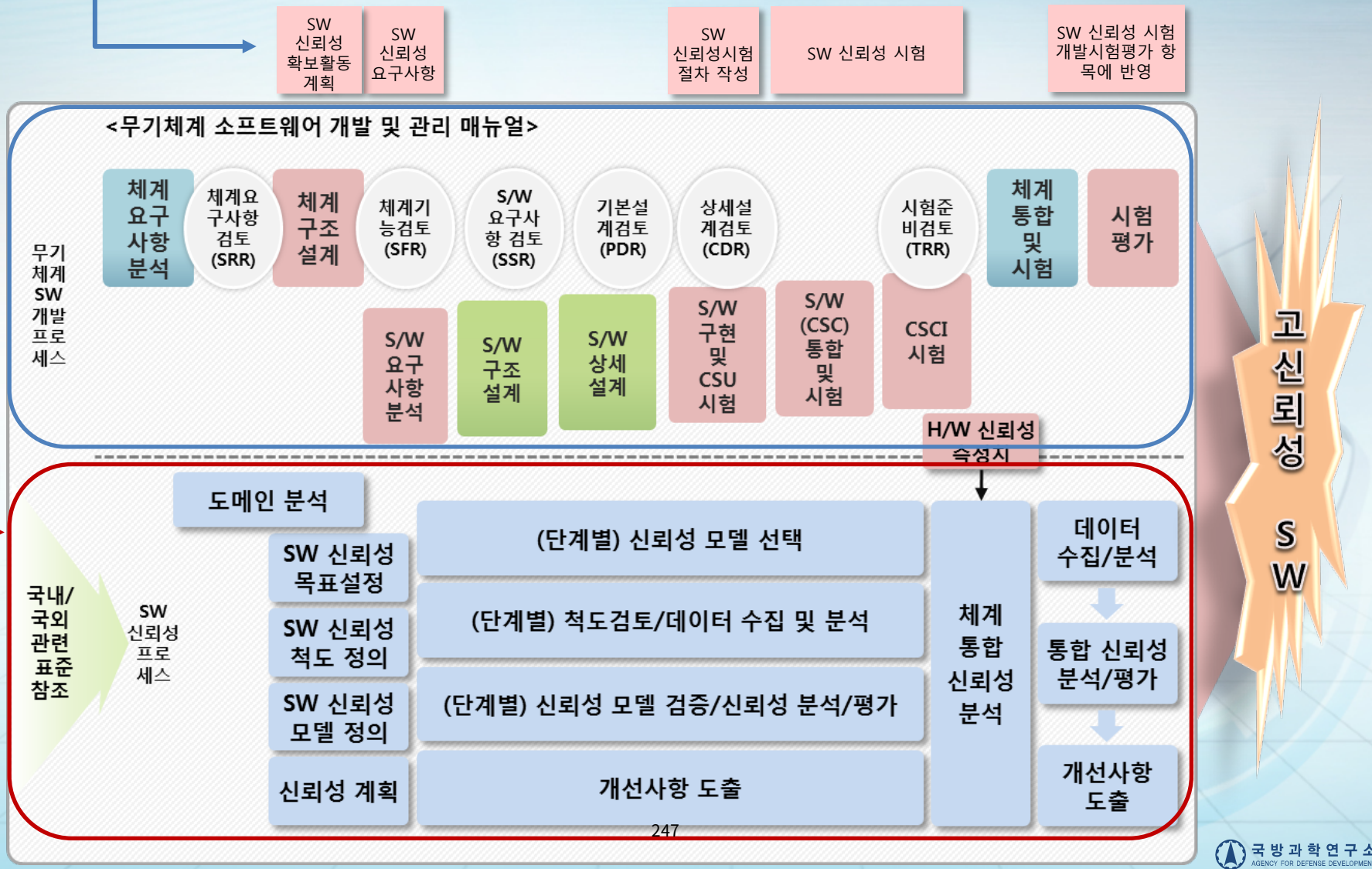
- 무기체계의 대형화/복잡화 및 수출증가로 고수준의 SW 신뢰성 프로세스 정립 요구
- 소스코드 시험 기반의 SW 신뢰성 평가 한계 극복을 위한 미래 지향적 개선 대책 필요
  - 잔존 결함 수, 시험 종료 시점 판단 및 SW 품질수준 파악 곤란
  - 한정된 기간/시험예산 문제로 무한정의 SW 테스팅 노력 투입 한계
- 의사결정 지원을 위해 개발 전(全)단계에서 SW 신뢰성의 정량적 평가 필요



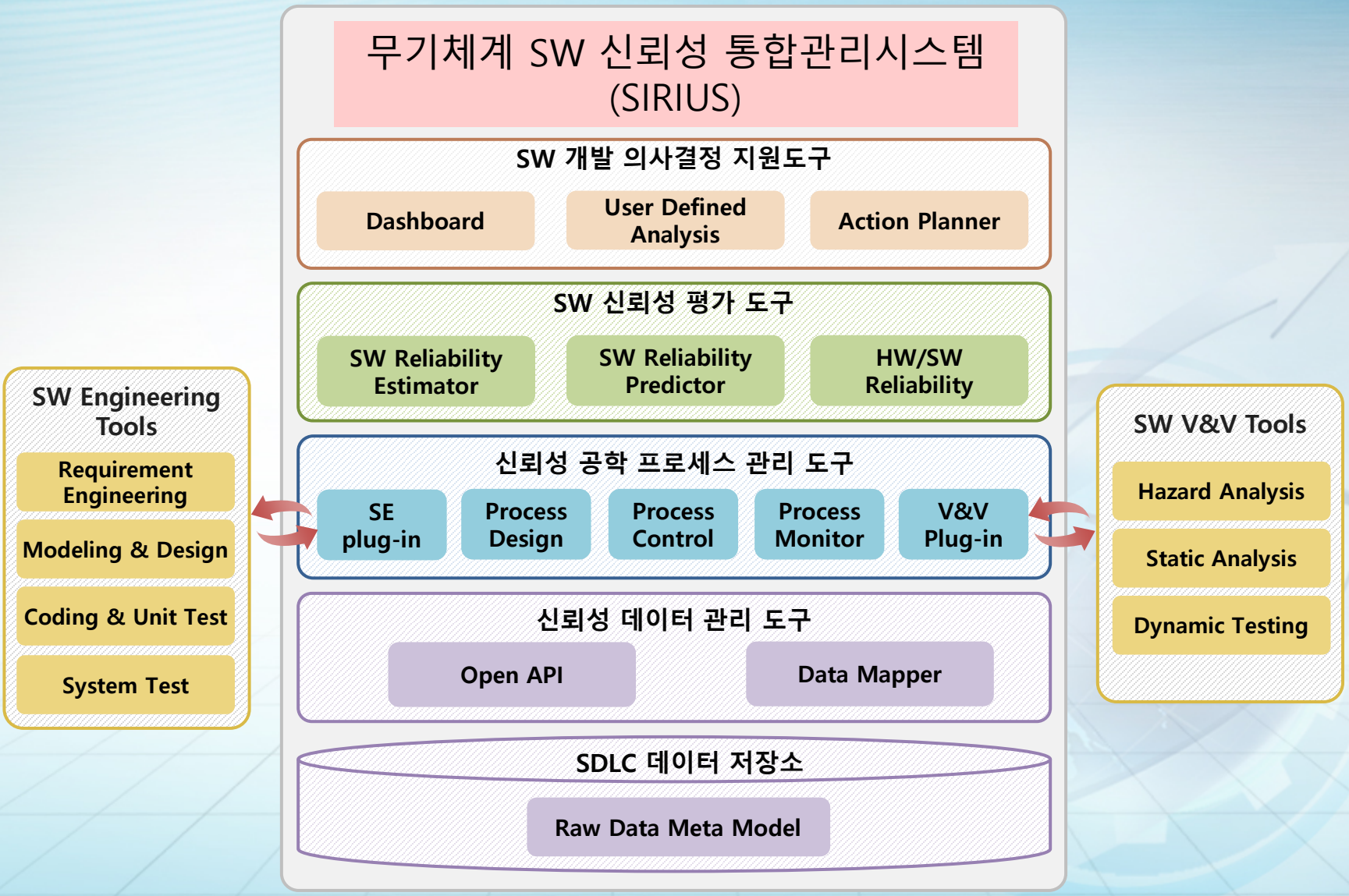
- SW 테스팅(정적/동적시험)으로 충분한가?
- SW 잔존 결함수는 어느 정도인가?
- 지금 개발중인 SW의 신뢰도는 어느 정도일까?
- SW 테스팅은 언제 끝내지?
- 기존에 쓰고 있는 SW 공학 도구와의 관계는?
- SW 신뢰성 통합관리 기능을 제공하는 도구는 없나?
- SW 신뢰성 관리 및 평가는 구체적으로 어떻게 하나?
- SW 신뢰성을 인증해줄 수 있는 방법은 없을까?

사업 개념

- **SW개발 전(全)단계에서 신뢰성 척도 및 평가모델을 활용한 SW 신뢰성 분석/관리 프레임워크 개발**



개발 시스템 구성



개발 시스템 목표

신뢰성 공학 프로세스 관리 도구

전수명주기 SW 프로세스 기반 신뢰성 공학 프로세스 관리 시스템 구축

- 사업별 참조 가능한 표준 기반 SW 신뢰성 참조/표준 프로세스 자원저장소 구축 및 활용
- 사업 특성에 적합한 프로세스 설계, 실행 및 모니터링 기능 제공
- 개발단계별 CASE 도구 1종 이상 및 V&V 도구 3종 이상 연동 기능 제공

신뢰성 데이터 관리 도구

SW 신뢰성 관련 데이터 관리 시스템 구축

- 신뢰성 Raw 데이터 Meta Model 구축
- 신뢰성 데이터 수집/관리 도구
- 원시 데이터 관리 주요 API 10종 이상

SW 신뢰성 평가 도구

SW 신뢰성 평가 시스템 구축

- SW 신뢰도 예측모델 3종 이상 구축
- SW 신뢰도 추정모델 5종 이상 구축
- HW/SW 통합 시스템 신뢰도 모델 1종 이상 구축

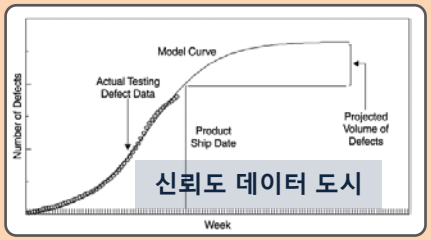
SW 개발 의사결정 지원 도구

SW 신뢰성 정보에 의한 SW 개발 의사결정 지원 시스템

- 신뢰성 척도 10종 이상 정량화 및 가시화
- SW 신뢰성 대시보드 구축
- SW 신뢰성 활동 플래너 구축

시스템 운영개념

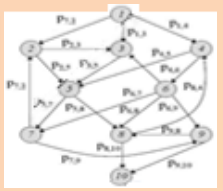
무기체계 SW 신뢰성 통합관리 시스템(SIRIUS)



SW 개발 의사결정 지원

RL-TR-92-52 Model

IEEE Std 982.1-2005



SW 신뢰도 평가

데이터 관리 APIs

SDLC 데이터 저장소

SE Plug-in

V&V Plug-in

요구분석 | 구조설계 | 상세설계 | 구현 | 단위시험 | 통합시험 | 시스템 시험 | 인수시험

도메인 분석 → 목표 설정 → 척도 정의 → 모델 정의 → 모델 검증 → 신뢰성 검증 → 신뢰성 평가

SW 신뢰성 프로세스

250

무기체계 SW 신뢰성 확보 및 검증 기술 로드맵

SW 신뢰성 통합관리시스템 개발

’15-’18

SW 신뢰성 통합관리 시스템 구축

- 신뢰성 공학 프로세스 관리 시스템
- 신뢰성 데이터 관리 시스템
- 신뢰성 평가 시스템
- 신뢰성 의사결정 시스템

SW 신뢰성 통합관리시스템 적용 활성화

’19-’21

시범 적용 및 DB 구축

- 무기체계 개발사업 시범적용
- 신뢰성 데이터베이스 구축
- 고유의 SW 신뢰성 평가모델 개발

SW 신뢰성 통합관리시스템 고도화

’22-

SW 신뢰성 확보 및 검증 기술 제도화

- 무기체계 개발사업 적용 확대
- 무기체계 분야별 고유의 SW 신뢰성 평가모델 개발
- 신뢰성 통합관리시스템 고도화
- SW 신뢰성 확보 및 검증 기술 제도화

무기체계 SW 신뢰성 통합관리 프로세스

관련 연구

IEEE Recommended Practice on Software Reliability (1633-2008)

순서	Practice	Description
1	Identify the Application	▪ 소프트웨어 신뢰성 측정 대상 소프트웨어 식별
2	Specify the Reliability Requirement	▪ 소프트웨어 신뢰성에 대한 명세서/목표를 기술
3	Allocate the Reliability Requirement	▪ 소프트웨어에 대한 신뢰성 요구사항 할당
4	Make a Reliability Risk Assessment	▪ 요구사항과 요구사항 변경에 따른 SW 결함이나 에러에 대한 위험 기반으로 수행 ▪ 요구사항의 수 혹은 변경에 의한 영향도를 기반의 위험 규정 방법이 정확하지 않으나, 큰 규모 시스템의 설계 분석시 반드시 필요 ▪ 위험 분석 방법은 남아있는 결함의 위험과 다음 결함이 일어날 시간에 대한 위험을 계산하는 방식으로 수행
5	Define Errors, Faults, and Failures	▪ 에러(Error), 결함(Fault), 실패(Failure)에 대한 정의를 프로젝트의 특성에 맞게 정의 ▪ 프로젝트 전반에 걸쳐 일관성 있게 정의
6	Characterize the Operational Environment	▪ 3가지 측면에 기반한 운영환경 특성 기술(시스템 형상, 시스템 업그레이드, 시스템 운영 프로파일)

관련 연구

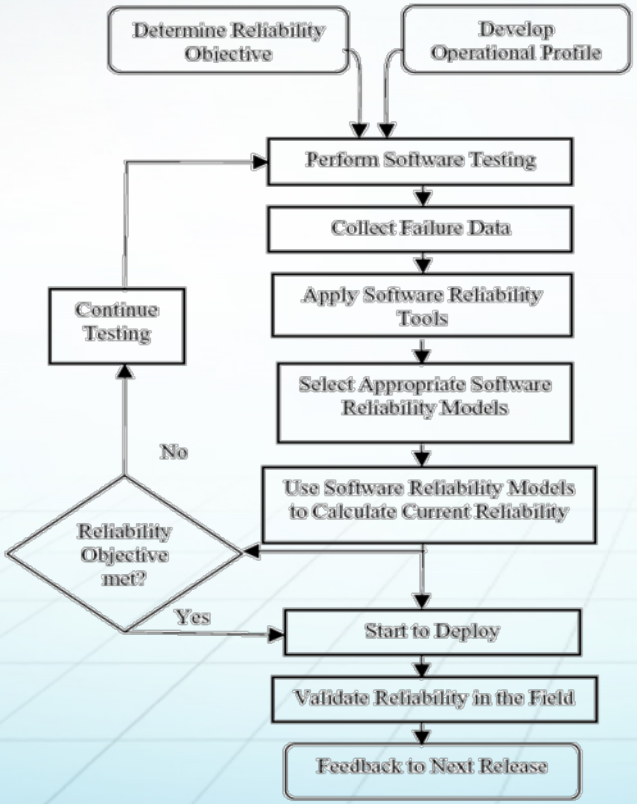
IEEE Recommended Practice on Software Reliability (1633-2008)

순서	Practice	Description
7	Select Tests	▪ 실 시스템에 적용할 테스트와 test case를 선택
8	Select Models	▪ 한 개 이상의 신뢰성 모델 선택 및 적용
9	Collect Data	▪ 데이터 수집 목적을 명확하게 정의 ▪ 적합한 데이터 수집
10	Estimate Parameters	▪ 데이터 평가/분석 방법 선택 ▪ Maximum likelihood estimation (MLE), least squares estimation 등 방법 사용
11	Validate the Model	▪ 선택된 신뢰성 모델의 유효성 검증, 수집 데이터와 파라미터 평가 방법에 대한 검증
12	Perform Assessment and Prediction Analysis	▪ 소프트웨어 신뢰성, 남아 있는 코드 결함 수 등 분석 수행
13	Forecast Additional Test Duration	▪ 목표된 초기 결함과 모델의 모수가 정해지면 추가적인 테스트 기간 예측

- 테스팅에 의한 신뢰성 평가에 중점을 두고 있음
- 각 단계별 프랙티스, 산출물 등 미제시
- 본 사업에서 구축하고자 하는 예측, 추정 전 단계에 대한 프로세스로 적용하는데 한계점 존재

관련 연구

- **SRE Process by Michael R. Lyu**
  - 4개의 주요 컴포넌트(Reliability objective, Operational profile, Reliability Modeling and Measurement, Reliability Validation) 및 12개 Practice로 구성



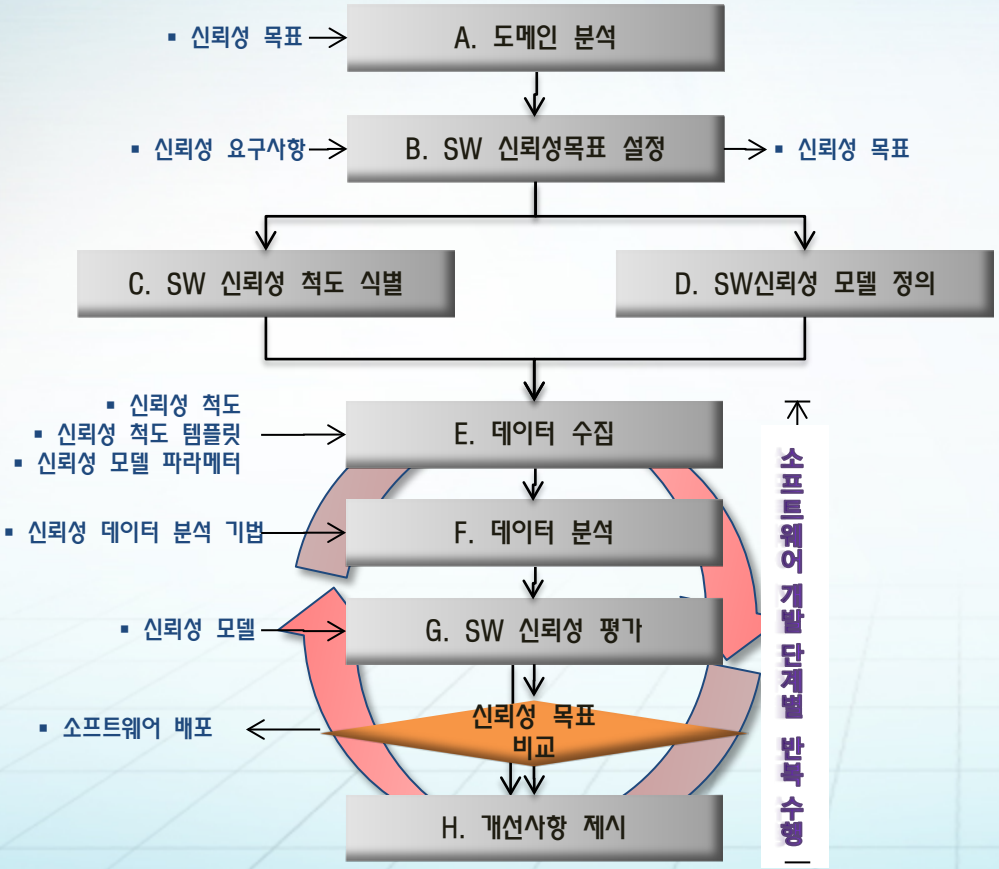
Michael R. Lyu
Software Reliability Engineering: Roadmap,ICSE2007

- SW 테스팅에 의한 신뢰성 평가에 중점을 두고 있음
- 산출물 등에 대한 구체적 제시 미흡
- 예측, 추정 전 단계에 대한 프로세스로 적용 한계점 존재

관련 연구

SPIRAL Reliability Assessment Process v2.0

– 8개 단계, 29개 세부활동



– 예측, 추정 전 단계에 대한 프로세스 정의
– 무기체계 SW 개발 특성을 반영한 프로세스로 변형 필요

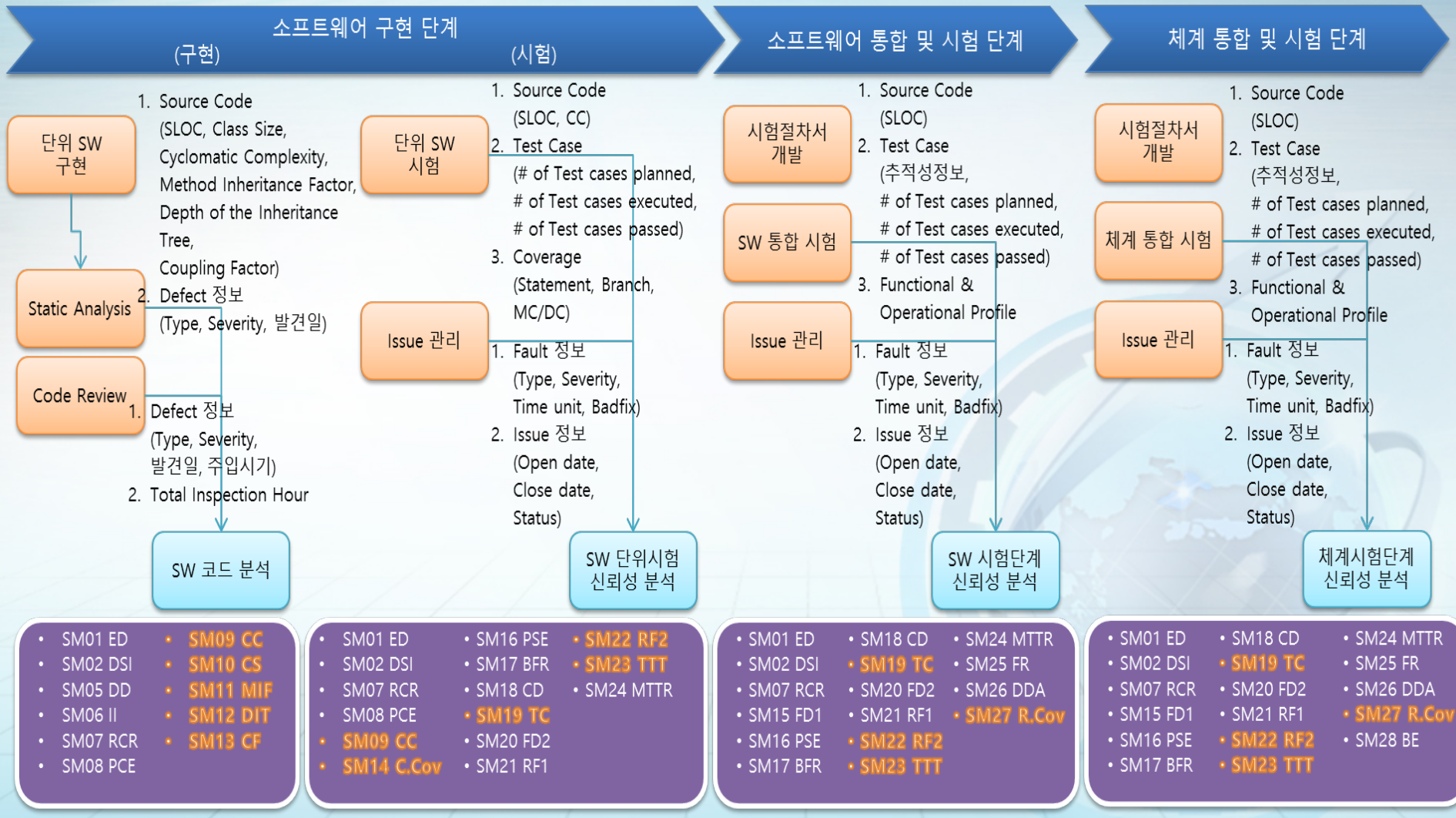
"Software Reliability Assurance Using a Framework in Weapon System Development: A Case Study", 8th IEEE/ACIS International Conference on Computer and Information Science, Shanghai, China, June 1-3, 2009.

SIRIUS 프로세스

CDR 이전 프로세스

SIRIUS 프로세스

CDR 이후 프로세스



결 론

결론

- **SW 개발 단계별 신뢰성 분석/평가 프레임워크 개발 필요**
  - SW 신뢰성 목표 대비 평가 가능, SW 배포 시점 결정 기준 제공
  - 다음 차례 발생하는 소프트웨어 결함 시점 예측 가능
  - 무기체계 SW 개발 초기부터 SW 신뢰성 분석을 통한 체계적인 개발 관리로 무기체계 품질향상 기여
  - 무기체계 SW 신뢰성 프로세스 구축으로 선진국 요구수준의 방산수출 무기체계에 대한 SW 신뢰성 확보

- **무기체계 SW 신뢰성 통합관리 프로세스 개발**
  - SW 신뢰성 예측, 추정 등 무기체계 SW 개발 특성에 적합한 전순기 지속적인 신뢰성 관리/평가 가능한 프로세스 제시

소프트웨어 R&D 프로젝트 검증을 위한 산출물과 활동

송상민 [1], Amarmend Dashbalbar [1], 이병정 [1], 이정원 [2]

[1] 서울시립대학교 컴퓨터과학과
서울 동대문구 전농동 163(전농동 90)
{maro0419, amaraa2848, bjlee}@uos.ac.kr
[2] 아주대학교 전자공학과
경기도 수원시 영통구 월드컵로 206
jungwony@ajou.ac.kr

요약: R&D 프로젝트의 테스트 프로세스와 소프트웨어 개발에서의 테스트 프로세스에 대한 방법이 여러 연구를 통해 다양하게 제안되고 있다. 그러한 테스트 프로세스들을 기반으로 소프트웨어 분야 R&D 프로젝트의 테스트 프로세스에 대해서 정의를 하고, 프로세스를 구성하는 상위/상세/단위명세의 테스트 계획 단계(test planning phase) 및 테스트 단계(testing phase)로 나누어 정의한다. 그리고 각 단계에 속하는 활동(activity)과 구체적인 작업(task)을 정의하고, 각 작업에서 만들어지는 산출물(work product) 및 최종 산출물인 SW Life-cycle Test Plan(SLTP), SW Life-cycle Test Specification(SLTS), SW Life-cycle Test Report (SLTR)에 대한 정의를 하고 모든 산출물 간의 관계를 정의 한다. 또한 본 논문에서 제안하는 테스트 프로세스를 SPEM 2.0 기반의 오픈 소스 도구를 사용하여 모델링하고, 이를 활용한 사례 연구를 통해서 본 연구의 효과성을 보인다.

핵심어: 소프트웨어 R&D 프로젝트, 테스트 프로세스, 산출물 검증, 프로세스 모델링, 소프트웨어 테스팅

1. 서 론

소프트웨어 개발에 있어서 검증 및 확인 단계가 설계 및 개발 단계보다 더 많은 노력이 든다. 일반적인 테스트 프로세스에서는 문서 산출물을 배제한 채 소프트웨어에 초점을 맞춰서 진행한다. 하지만, 만약 산출물(소프트웨어 요구사항 명세서, 연구개발 과제 계획서 등)에서 사전 결함이 발생한다면, 이후 소프트웨어 개발 단계에서 큰 영향을 미치게 될 것이다. 따라서 본 논문은 소프트웨어 분야 R&D 프로젝트의 검증 과정에서 소프트웨어와 함께 문서 산출물의 검증까지 포함하는 테스트 프로세스를 정의한다.

본 연구에서는 효과적인 소프트웨어 R&D 프로젝트의 테스트 프로세스를 제안하기 위해 소프트웨어 개발 프로세스 모델을 분석하였다. 가장 널리 알려진 폭포수 모델의 경우에는 이해하기 쉽고, 생명주기 전반에 걸쳐 진화가 예상되지 않거나 변경이 적어 비교적 위험이 낮은 프로젝트에 적합하다는 장점이 있으나, 특정 단계에서의 결함이 이후의 단계까지 계속 연결이 되어 발견 시기도 늦고 높은 비용이 발생하는 문제가 있다.

나선형 모델은 시스템을 계획 수립과 위험 분석, 개발, 평가 단계로 나누어 여러 번의 개발 주기를 거치면서 시스템을 완성한다. 프로젝트 초기부터 실패 요인과 위험 요소들을 찾아내어 결함이 발생할 확률을 낮추며 초기에 프로토타입이 생성되지만 많은 비용이 발생하며 작은 프로젝트에서 효율적이지 못할 수 있다는 단점이 있다.

V 모델은 생명주기에 반복이 없어서 유연하지 못하고, 테스트 단계에서 발견 된 문제에 대한 명확한 경로를 제공하지 않는 문제가 있는 반면 각 단계에 구체적인 산출물이 있으며 모든 단계에 검증과 확인 과정이 있어서 오류를 줄일 수 있다는 장점이 있다.

본 연구에서는 소프트웨어 개발 프로세스 모델들이 가지는 문제점을 보완하고 장점을 활용하여 소프트웨어 R&D 프로젝트의 테스트 프로세스를 제안한다. 여러 번의 반복을 통하여 위험 요소를 줄이는 나선형 모델과, 단계마다 산출물을 생성하고 검증 과정을 통해 오류를 줄이는 V 모델과 검증하고 확인하는 V&V(Verification and Validation) 프로세스를[1] 기반으로 하는 소프트웨어 R&D 프로젝트의 테스트 프로세스를 제안한다. 또한 프로세스 활동(activity)과 작업(task)의 상세한 내용과 작업에 대한 역할(role)까지 정의한다. 상위/상세/단위명세에서 활용하는 각각의 산출물을 정의하고, 적용한 연구 사례를 보여주어 소프트웨어 R&D 프로젝트 테스트에 유용한 모델이 될 수 있음을 보인다.

본 논문의 기여도는 다음과 같다.

- 소프트웨어 요구사항 명세서, 소프트웨어 설계 명세서, 모듈명세서 같은 개발 산출물을 포함하여 연구개발 과제 계획서, 연구성과 요약보고서,

연차실적 계획서, 연구노트 같은 연구 산출물까지 함께 테스트하여 초기에 결함을 발견하거나 관리의 효율성 높이는 테스트 프로세스 모델을 제시한다.

- 제시한 프로세스의 각 작업에 따른 산출물을 명시적으로 정의하고 활용한 사례를 보여줌으로써 효과성을 보인다.

본 논문의 구성은 다음과 같다. 2 장에서 테스트를 위한 연구들과 테스트 프로세스 표준에 대해 분석을 하고, 3 장에서는 소프트웨어 R&D 프로젝트의 테스트 프로세스와 그에 따른 산출물을 정의하고, 각 산출물간의 관계에 대한 정의를 한다. 4 장에서는 제안한 방법론을 따라 SPEM2.0 기반의 EPF Composer 를 사용해 모델링 하고, 모델링 된 모델을 업무프로세스 관리 도구(uEngine BPMS)에 적용한 사용 사례를 소개한다. 5 장에서는 제안한 테스트 프로세스와 사례연구에 대한 토의를 하고, 6 장에서는 본 연구의 결론과 향후 연구방향을 제시한다.

2. 관련연구

본 논문은 소프트웨어 R&D 분야 테스트 프로세스를 정의하는 연구로써 테스트 프로세스의 국제 표준과 프로세스 실행하는 기술에 중점을 두고 있다.

T. Abdou [2]은 오픈 소스 프로젝트를 위한 기존 테스트 프로세스들이 새로운 프로세스 표준과 얼마나 적합한지를 조사하여 비교한 연구다. ISO 29119-2 의 활동과 세분화된 작업을 조사하여 현재 오픈 소스 프로젝트(Apache HTTP server, Mozilla Web browser, NetBeans 등)에서 사용되는 테스트 프로세스의 활동을 비교하였다. 분석 결과 오픈 소스 프로젝트는 테스트 계획을 작성하지 않아서 테스트 환경과 실행에 대한 정확한 정보가 없기 때문에 테스트의 효율성이 떨어진다.

S. Imoto [3]는 연구개발 프로젝트의 목표와 연구내용, 기대 효과, 특허를 취득 할 가능성, 프로젝트의 일정, 개발에 드는 비용 등 정보를 기반으로 R&D 프로젝트 평가에 대한 모델을 제안했다. 그러나 연구노트, 회의록 같은 비주기적 산출물에 대한 내용을 포함하는 본 논문이 제안한 프로세스와 다르게 이 모델은 비주기적 산출물은 다루지 않기 때문에 평가가 정확하지 않을 수 있다.

정학선 [4]은 JIRA 도구를 ISO 29119-2 에서 제시하는 테스트 프로세스로 구성하여 프로젝트 환경에 따라 테스트 프로세스 및 테스트 관리 항목 변경에서 신속한 테스트 변화 관리를 할 수 있다. 하지만 연구개발 산출물에 대한 테스트는 포함하지 않아서 소프트웨어 R&D 프로젝트의 테스트 프로세스로 적용하기에 적절하지 않다.

ISO 29119-2 의 상위 레벨인 조직 테스트 프로세스(organizational test process)에서는 회사 내부에서 전체적으로 다루는 일반적인 정책과 전략을 세운다. 그것들을 기반으로 테스트하는 프로젝트 별로 테스트 계획이 만들어지고 계획서대로 진행되는지를 모니터링 하는 작업과 테스트 작업이 테스트 관리 프로세스(test management process) 레벨에서 이뤄진다.

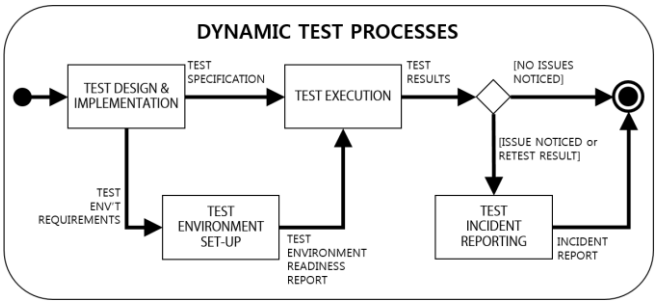


그림 1 ISO 29119-2 테스트 프로세스 모델

동적 테스트 프로세스(dynamic test process) 레벨에서는 그림 1 과 같이 테스트 케이스 생성하고, 테스트 환경을 설정하여, 테스트를 실행하고, 결함 발생했을 경우에는 결함 보고서를 작성하는 총 4 가지 활동이 있다[5-6]. 이러한 각 활동들은 세분화되어 한 단계 더 작은 단위인 작업으로 나눠진다.

3. 소프트웨어 R&D 프로젝트의 테스트 프로세스와 산출물 정의

본 절에서는 소프트웨어 R&D 프로세스의 테스트 프로세스를 정의하고, 이러한 테스트 프로세스에서 이루어지는 각 활동과 작업에 해당하는 소프트웨어 R&D 산출물을 제안한다.

3.1 소프트웨어 R&D 프로젝트의 테스트 프로세스 정의

3.1.1 R&D 프로젝트의 테스트 프로세스

R&D 프로젝트의 테스트 프로세스는 계획 단계 산출물, 연차 별 산출물, 최종 단계 산출물을 대상으로 순차적으로 진행되며, R&D 테스트 프로세스 상의 산출물들은 주기적 산출물과 비주기적 산출물로 나눈다. R&D 테스트 프로세스에서 주기적 산출물과는 다르게 검증에 포함하지 않는 회의록과 연구노트 같은 비주기적 산출물은 상시적으로 만들어지기 때문에 R&D 프로젝트의 진행 상황 검증이나 각 검증 활동 단계에 추가 입력 산출물로 효과적인 프로젝트 검증에 도움이 될 수 있다. [7]

그러므로 본 연구에서 정의하는 소프트웨어 분야 R&D 테스트 프로세스는 회의록, 연구노트, 월간/주간 보고서 등과 같은 상시적 산출물 입력에 대한 검증 활동까지 포함한다. 상시적 산출물 검증을 통해서

R&D 프로젝트의 진행 상황의 효과적인 모니터링을 가능하게 하고, 검증 활동 단계별로 정확한 검증을 위한 추가 자료로 활용된다. 또한 최종 단계의 연구 개발 성과와 연구 성과 보고서를 통해 결과만을 확인하는 것이 아니라 연구개발 과정에 대한 검증도 가능하게 한다.

3.1.2 SW 개발의 테스트 프로세스

소프트웨어 개발은 소프트웨어의 비가시적인 특성 때문에 소프트웨어 개발 단계별로 지속적이고 면밀하게 검증해야만 적절하게 검증될 수 있다. 그래서 소프트웨어 개발에서의 테스트 프로세스는 검증하고 확인하는 V&V 프로세스가 일반적으로 활용된다. V&V 소프트웨어 테스트 프로세스에서 시스템 테스트 계획은 시스템 테스트와, 통합 테스트 계획은 통합 테스트와 같이 각 검증 단계의 계획 활동들은 확인 단계의 테스트 활동들과 연관된다. 이렇게 연관된 테스트 계획과 테스트 활동은 비가시적이고 복잡한 특성의 소프트웨어 산출물 검증을 단계별로 반복적으로 상세하게 검증한다.

본 연구에서도 소프트웨어 R&D 프로젝트를 대상으로 하기 때문에 V&V 프로세스의 테스트 계획 및 테스트 활동을 포함하는 테스트 프로세스를 정의한다. 소프트웨어 분야의 테스트 프로세스를 포함으로써 SW 개발 산출물인 소프트웨어 요구사항 명세서, 소프트웨어 설계 명세서를 검증 및 활용한다. SW 개발 산출물 검증 및 활용을 통해 비가시적이고 복잡한 특성을 지닌 소프트웨어 산출물의 품질에 대한 효과적인 검증을 달성한다.

3.1.3 소프트웨어 R&D 프로젝트의 테스트 프로세스

본 논문은 R&D 프로젝트의 테스트 프로세스 분석과 소프트웨어 개발에서의 테스트 프로세스 분석을 통해서 연구 산출물뿐 아니라 소프트웨어 개발 산출물 검증도 포함하는 SW 분야 R&D 테스트 프로세스를 정의한다. 제안하는 테스트 프로세스의 각 단계별 활동들은 테스트 프로세스 표준인 ISO 29119-2 를 기준으로 정의한다.

R&D 프로젝트의 테스트 프로세스와 소프트웨어 개발에서의 테스트 프로세스의 각 활동들로부터 SW 분야 R&D 테스트 프로세스의 각 단계별 활동을 정의한다. 테스트 계획 단계로 상위명세 테스트 계획 단계(high-level test planning phase), 상세명세 테스트 계획 단계(detail-level test planning phase), 단위명세 테스트 계획 단계(unit-level test planning phase)를 정의한다. 그리고 테스트 단계로 상위명세 테스트 단계(high-level testing phase), 상세명세 테스트 단계(detail-level testing phase), 단위명세 테스트 단계(unit-level testing phase)를 정의한다.

앞서 정의한 테스트 프로세스의 단계를 아래 그림 2 와 같은 프로세스로 구성한다.

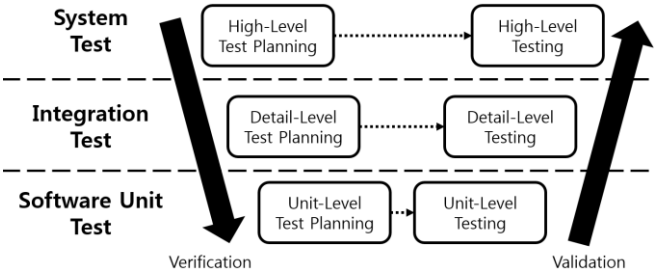


그림 2 각 단계와 테스트 종류의 관계

연구개발 과제 계획서 검증과 시스템 테스트 계획은 상위명세 테스트 계획 단계로 정의한다. 연차실적 계획서 검증과 통합 테스트 계획은 상세명세 테스트 계획 단계로, 연구노트 검증과 단위 테스트 계획은 단위명세 테스트 계획 단계로, 단위 테스트는 단위명세 테스트 단계로, 통합 테스트는 상세명세 테스트 단계로 정의한다. 마지막으로 연구 성과 보고서 검증과 시스템 테스트를 포함하여 상위명세 테스트 단계로 정의한다.

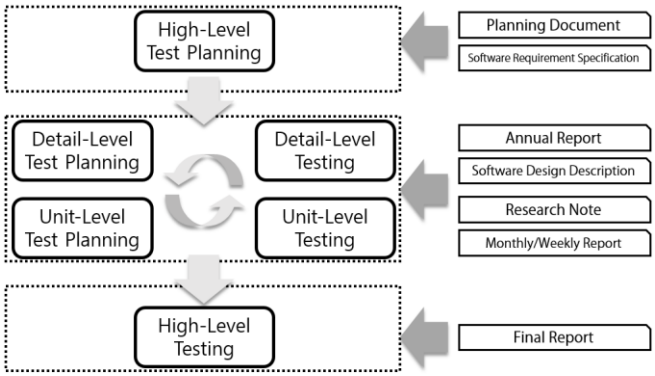


그림 3 테스트 프로세스의 순서와 활용하는 산출물

그림 3 과 같이 연구과제 산출물과 SW 및 관련 산출물들을 대상으로 프로젝트 계획 초기에 상위명세 테스트 계획 단계를 한다. 상위명세 테스트 계획 단계가 끝나면 상세명세 테스트 계획 단계, 단위명세 테스트 계획 단계, 단위명세 테스트 단계, 상세명세 테스트 단계가 주기적으로 반복되며 프로젝트를 지속적으로 검증한다. 마지막으로 프로젝트 최종 단계에서 상위명세 테스트 단계를 통해 종합적인 검증을 진행한다.

상위명세 테스트 계획 단계에서 사용되는 연구개발 과제 계획서와 달리 소프트웨어 요구사항 명세서(SRS)는 변경 요청이 발생한다. 그런 경우에는 상세명세 테스트 단계 혹은 단위명세 테스트 단계 등 진행 중인 단계에서 다시 상위명세 테스트 계획 단계로 돌아가서 진행을 하면 변경 사항에 따라 테스트

를 수행할 수 있다.

본 논문은 SW 분야 R&D 테스트 프로세스의 각 활동 별 입력 산출물들을 정의한다. 상위명세 테스트 계획 단계, 상세명세 테스트 계획 단계, 단위명세 테스트 계획 단계와 같은 테스트 계획 단계들에서는 공통적으로 연구자가 입력하는 테스트 계획인 SLTP(Software Life-cycle Test Plan)를 작성한다. 그리고 SLTP 로 상위/상세/단위명세 테스트 계획 단계에 따라 검증에 필요한 테스트 명세인 SLTS(Software Life-cycle Test Specification)를 생성하여 진행한다. 테스트 명세에 따라 테스트 단계가 완료되면 결과가 SLTR(Software Life-cycle Test Report)를 통해 출력된다. SLTP, SLTS, SLTR 은 뒤에 3.3 장에서 자세하게 설명된다.

3.2 활동과 작업, 역할 정의

소프트웨어 R&D 프로젝트의 테스트 프로세스는 크게 테스트 계획 단계(test planning phase)와 테스트 단계(testing phase)로 나눈다.

상위명세 테스트 계획 단계는 요구사항 및 개발 산출물에 대한 이해를 통해 테스트 해야 할 범위를 이해는 배경 이해(understand context)와 요구사항에서 얻어낸 정보로부터 프로젝트의 위험 요소를 추정하고 식별하는 위험 요소 추정 및 식별(identify & estimate risk), 앞서 찾아낸 위험 요소에 대한 해결 방법을 식별하는 위험 요소 해결 방법 식별(identify risk treatment approaches), 테스트에 사용할 도구 선택이나 방법에 대한 테스트 전략을 설계하는 테스트 전략 설계(design test strategy), 테스트 전략에 따른 적당한 인력을 구성하고 구성원들에 대한 일정을 결정하는 팀원 및 일정에 대한 결정(determine staffing and scheduling)의 작업들로 구성된 계획 활동(planning activity)이 있다.

표 1 테스트 계획 단계

Activity	Task	Work Product (Outcome)
Planning	Understand Context	Scope
	Identify & Estimate Risk	Analyzed Risks
	Identify Risk Treatment approaches	Treatment Approaches
	Design Test Strategy	Test Strategy
	Determine Staffing and Scheduling	Schedule and Staffing Profile

표 1 에서 보여주는 테스트 계획 단계의 경우 상위명세 테스트 계획 단계가 상세명세 테스트 계획 단계, 단위명세 테스트 계획 단계와 다르게 계획 활동의 작업이 세 가지 더 많다. 그 이유는 IEEE 829[8]이 Master Test Plan(MTP)과 Level Test Plan(LTP)으로 구성된 것과 같이 상위명세 테스트 계획 단계에서는 프로젝트 전체에 대한 계획을 포함하기 때문이다. 따라서 상세명세 테스트 계획 단계와 단위명세 테스트 계획 단계에서는 배경 이해 작업과 위험 요소 추정 및 식별 작업과 위험 요소 해결 방법 식별 작업을 수행하지 않는다.

표 2 처럼 테스트 단계는 테스트 설계 및 구현(test design and implementation)과 테스트 환경 설정 및 유지(test environment set-up and maintenance), 테스트 실행(test execution), 테스트 결함 보고서 작성(test incident reporting)의 활동으로 이루어진다.

테스트 설계 및 구현 활동은 개발 산출물과 연구 산출물의 작업의 해당하는 내용이 다소 차이가 있다.

개발 산출물의 경우에는 테스트 설계 및 구현 활동에서 개발 산출물의 특징을 식별하는 특징 식별(identify feature set)과 테스트 케이스로 검증 해야 할 시스템 또는 컴포넌트의 항목이나 이벤트를 도출하는 테스트 조건 생성(derive test conditions) 작업을 통해서 테스트 항목(test item)을 만든다. 그리고 얼마나 테스트가 필요할지에 대한 범위와 어떠한 방법으로 수행할 것인지를 결정하는 테스트 범위 생성(derive test coverage) 작업을 통해 테스트 설계 명세서(test design specification)와 테스트 환경 요구사항(test environment requirement)을 생성하여, 테스트 설계 명세서를 기반으로 테스트 케이스를 생성하는 테스트 케이스 생성(derive test cases) 작업을 수행한다. 그 후에 테스트 케이스들의 실행 순서를 포함하는 테스트 프로시저를 생성하는 테스트 프로시저 생성(derive test procedures) 작업에서 테스트 프로시저 명세서(test procedure specification)가 나온다.

표 2 테스트 단계

Activity	Task	Work Product (Outcome)
Test Design and Implementation	Identify Feature Set	Test Item
	Derive Test Conditions	
	Derive Test Coverage	Test Design Specification
		Test Environment Requirement
	Derive Test Cases	Test Case Specification
	Derive Test Procedures	Test Procedure Specification
Test Environment Set-Up and Maintenance	Establish Test Environment	Test Result
	Maintain Test Environment	
Test Execution	Execute Test Procedures	
	Compare Test Results	
Test Incident Reporting	Analyze Test Result	Incident Report
	Create Incident Report	

연구 산출물은 특징 식별과 테스트 조건 생성 작업에서 문서 산출물의 구성 항목(configuration item)을 식별하여 항목 간 연관 관계를 명시하는 연관성 링크에 대한 테스트 항목을 생성한다. 생성된 테스트 항목에 대해 테스트 범위 생성 작업에서 연관성 링크마다 어떠한 규칙을 가지고 있어야 하는지에 대한 연관성 규칙을 생성하여 테스트 설계 명세서에서 포함한다. 테스트 케이스 생성 작업에서는 앞서 만든 연관성 링크가 연관성 규칙을 위배하는지 판단하는 테스트 케이스 명세서를 생성하고 이후의 작업은 개발 산출물과 동일한 과정으로 진행한다. [9]

테스트 환경 설정 및 유지 활동에서는 앞서 만든 테스트 환경 요구사항을 기반으로 테스트 환경을 수립하는 테스트 환경 수립(establish test environment) 작업 후에 수립한 테스트 환경을 유지하는 테스트 환경 유지(maintain test environment) 작업을 수행한다. 마찬가지로 테스트 실행 활동은 테스트 프로시저 명세서에서 정의 한 테스트 프로시저를 실행하는 테스트 프로시저 실행(execute test procedures) 작업과 테스트 프로시저를 수행하며 나온 테스트 결과를 테스트 케이스에 있는 예상 결과와 비교하는 테스트 결과 비교(compare test results) 작업을 거쳐 테스트 결과(test result)가 만들어진다.

테스트 단계의 마지막으로 테스트 결과를 분석하는 테스트 결과 분석(analyze test result) 작업과 테스트 결과 분석에 이상이 발견된다면 결함 보고서를 생성하는 결함 보고서 생성(create incident report)에 대한 테스트 결함 보고서 작성 활동에서 결함 보고서(incident report)를 생성하여 해당 명세의 테스트 프로세스를 마무리한다.

표 3 각 작업에 해당하는 역할

Task	Primary performer	Additional performer
Understand Context	PM	Tester
Identify & Estimate Risk		
Identify Risk Treatment approaches		
Design Test Strategy		
Determine Staffing and Scheduling		
Identify Feature Set	Tester	PM
Derive Test Conditions		
Derive Test Coverage		
Derive Test Cases		
Derive Test Procedures		
Establish Test Environment	Tester	PM
Maintain Test Environment		
Execute Test Procedures		
Compare Test Results		
Analyze Test Result	PM	Tester
Create Incident Report	Tester	PM

각 작업을 수행하는 역할에 대한 표 3 에서는 필수 역할(primary performer)과 추가 역할(additional performer)을 PM(Project Manager)과 테스터(tester)의 역할로 구성한다.

3.3 산출물 정의

앞에서 정의한 작업들의 산출물은 테스트의 진행에 따라 상위/상세/단위명세 테스트 계획으로 SLTP 를 나타나고, 테스트 명세인 SLTS 로 실질적인 테스트에 대한 설계 및 구현을 하여 테스트의 결과가 SLTR 로 나타난다.

산출물의 종류는 크게 개발 산출물과 연구 산출물, 소프트웨어와 문서 산출물로 구분하여 설명한다. 문서 산출물이란 소프트웨어 제품(software product)을 제외한 모든 문서를 말하며, 소프트웨어와 구분되는 산출물을 뜻한다. 개발 산출물은 소프트웨어를 포함하여 소프트웨어 요구사항 명세서, 소프트웨어 설계 명세서 같은 소프트웨어 개발에 직접적인 영향을 주는 산출물이며, 연구 산출물은 연구개발 과제 계획서, 연차실적 계획서와 같은 R&D 프로젝트에서 사용하는 문서 산출물이다.

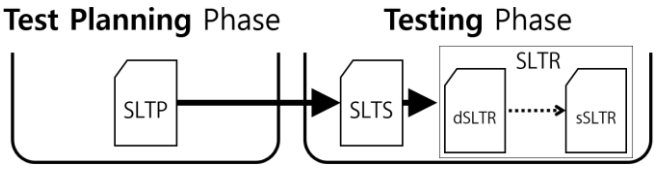


그림 4 산출물이 생성되는 각 단계

그림 4 에서와 같이 순차적으로 SLTP 는 테스트 계획 단계에서 생성되며, SLTS 와 SLTR 은 테스트 단계에서 순차적으로 생성되는 산출물이다. 테스트 계획 단계의 계획 활동에서 산출물들이 SLTP 로 생성되고, 테스트 설계 및 구현 활동에서 SLTS 가 생성된다. SLTS 에 따라 나머지 활동인 테스트 환경 설정 및 유지 활동과 테스트 실행, 테스트 결함 보고서 생성 활동에서는 SLTR 이 생성된다. 소프트웨어 요구사항 명세서 같은 개발 산출물은 연구개발 과제 계획서 같은 연구산출물에 의존적이며, 그림 4 에서 SLTR 의 dSLTR 과 sSLTR 가 의존 관계인 것을 보여준다.

3.3.1 SLTP (SW Life-cycle Test Plan)

소프트웨어 생명 주기의 단계 별 테스트 계획인 SW Life-cycle Test Plan(SLTP)는 연구 산출물과 소프트웨어 관련 산출물을 대상으로 하는 테스트 계획으로 정의한다. 이러한 테스트 계획은 소프트웨어 R&D 프로젝트의 테스트 프로세스 산출물에 대한 검증을 위해 작성된다. [10]

SLTP 의 구성은 상위명세에 해당하는 소프트웨어 요구사항 명세서, 연구개발 과제 계획서, 연구성과

요약보고서와 상세명세에 해당하는 소프트웨어 설계 명세서, 연차실적 계획서 그리고 단위명세에 해당하는 모듈명세서, 연구노트로 구성된다. 표 4 에서는 상위명세, 상세명세, 단위명세에 해당하는 각 구분들에 속한 항목들을 보여준다.

표 4 SLTP 의 구성

구분	항목	내용
소프트웨어 요구사항 명세서	요구사항	
연구개발 과제 계획서	연구의 필요성	• Policy
	연구목표 및 내용	• Scope
	연구의 추진전략 및 방법	
	연구결과의 활용방안 및 기대성과	• Analyzed Risks
연구성과 요약보고서	연구내용 및 결과	• Treatment Approaches
	연구성과	
소프트웨어 설계 명세서	사용자 인터페이스 설계	• Test Strategy
	아키텍처 설계	
연차실적 계획서	기 수행 연구실적	• Schedule and Staffing Profile
	차년도 연구계획	
모듈명세서	컴포넌트/클래스/메소드	
연구노트	세부연구목표	

각 항목은 표 1 에서 정의한 테스트 프로세스의 테스트 계획 단계의 산출물인 범위(scope)와 분석된 위험 요소(analyzed risks), 해결 방법(treatment approaches), 테스트 전략(test strategy), 팀원 및 일정(schedule and staffing profile)에 대해 각 상위/상세/단위명세에 맞는 내용으로 구성하며, 팀이나 회사 같은 조직의 정책을 포함하는 해당 R&D 프로젝트의 정책(policy)도 포함한다.

상위명세 계획 단계의 테스트 전략 설계 작업은 다음 명세인 상세명세 테스트의 반복 조건에 대해 정하여 반복에 대한 기준을 작성하며, 마찬가지로 상세명세 계획 단계에서 테스트 전략 설계 작업은 단위명세 테스트의 반복 조건에 대한 기준을 포함한다.

3.3.2 SLTS (SW Life-cycle Test Specification)

표 5 SLTS 의 구성

구분	항목	내용
소프트웨어 요구사항 명세서	요구사항	• Evaluation Method
연구개발 과제 계획서	연구의 필요성	• Test Item
	연구목표 및 내용	
	연구의 추진전략 및 방법	• Test Design Specification
	연구결과의 활용방안 및 기대성과	
연구성과 요약보고서	연구내용 및 결과	• Test Environment Requirement
	연구성과	
소프트웨어 설계 명세서	사용자 인터페이스 설계	
	아키텍처 설계	• Test Case Specification
연차실적 계획서	기 수행 연구실적	
	차년도 연구계획	
모듈명세서	컴포넌트/클래스/메소드	• Test Procedure Specification
연구노트	세부연구목표	

테스트 계획인 SLTP 생성 후에는 소프트웨어 생명 주기의 테스트 명세인 SW Life-cycle Test Specification(SLTS)를 작성한다. SLTS 는 각 테스트의 실질적인 명세로 이루어져 있으며, 표 5 와 같이 SLTP 의 각 항목에 대응하는 평가 방법(evaluation method)과 테스트 항목, 테스트 설계 명세서, 테스트 환경 요구사항, 테스트 케이스 명세서, 테스트 프로시저 명세서로 구성되어 있다.

개발 산출물의 상위명세에서는 ISO 9126 의 소프트웨어 품질 평가 척도(metrics)인 기능성, 신뢰성, 사용성, 효율성, 유지보수성, 이식성을 기준으로 소프트웨어에 대한 평가 방법을 작성하고, 기능성에 대한 평가는 개발 산출물에 해당하는 소프트웨어 요구사항 명세서와 소프트웨어 설계 명세서까지 포함하여 평가한다. 상세명세와 단위명세에서는 항목의 입력 유무나 동작의 확인에 대한 평가 방법을 명시한다.

연구 산출물은 전체 항목에서 차지하는 비중, 성능 수준, 목표치 도달 수준을 기준으로 테스트하여 추적성, 완전성, 일관성에 대해 평가 방법을 포함한다.

3.3.3 SLTR (SW Life-cycle Test Report)

소프트웨어 분야 R&D 프로젝트의 테스트 보고서인 SW Life-cycle Test Report (SLTR)은 SLTP 에 작성된 테스트 계획을 바탕으로 작성한 테스트 명세인 SLTS 에 따라 수행된 결과이다. SLTR 은 표 6, 표 7 같이 연구 산출물에 대한 테스트 보고서인 dSLTR (document SLTR)과 개발 산출물에 대한 테스트 보고서 sSLTR(software SLTR)로 나뉘어진다.

표 6 dSLTR 의 구성

분류	구분	항목	내용
상위명세	연구개발 과제 계획서	연구의 필요성	• 추적성
		연구목표 및 내용	
		연구의 추진전략 및 방법	• 완전성
		연구결과의 활용방안 및 기대성과	
	연구성과 요약보고서	연구내용 및 결과	• 일관성
		연구성과	
상세명세	연차실적 계획서	기 수행 연구실적	• 수행결과
		차년도 연구계획	• ID
단위명세	연구노트	세부연구목표	

표 7 sSLTR 의 구성

분류	구분	항목	내용
상위명세	소프트웨어 요구사항 명세서	요구사항	• 기능성 • 신뢰성 • 사용성 • 효율성 • 유지보수성 • 이식성 • 입력유무 • 수행결과 • ID
상세명세	소프트웨어 설계 명세서	사용자 인터페이스 설계	
		아키텍처 설계	
단위명세	모듈명세서	컴포넌트/클래스/메소드	

SLTS 에서 작성된 평가 방법에 맞도록 dSLTR 은 추적성, 완전성, 일관성에 대한 평가 결과를 포함하며, sSLTR 은 기능성, 신뢰성, 사용성, 효율성, 유지보수성, 이식성의 평가 결과와 입력유무의 결과로 구성된다. 또한 공통적으로 가지고 있는 수행결과에서는 결함의 여부 및 테스트 결과에 대한 정보가 있고, 문제가 있을 시 해당 결과에서 발생한 결함의 고유한 ID 를 부여하여 포함하고 있다.

표 8 결함 보고서의 구성

ID	Status	Type	Description	Assigned To	Modified Date	Reporter	Reported Date
626	NEW	Document	Needs to Recover Traceability Links between Code and SDD	F. Collins	-	V. Ferreira	2015-10-23
419	FIXED	Software	Defect Created date is taking Future Date	M. Kuhrmann	2015-12-02	R. Bendraou	2015-11-27

결함에 대한 정보는 표 8 와 같이 ID 를 기준으로 별도의 결함 보고서(incident report)에 자세한 정보가 기술된다. 고유한 ID 로 구분되는 결함은 각 해결 또는 수정 여부를 나타내는 상태(status)와 연구 산출물과 개발 산출물에 대한 종류(type), 해당 결함에 대한 내용 설명(description), 결함 수정을 할당 받은 담당자(assigned to), 최종 수정된 날짜(modified date), 최초 발견한 사람(reporter), 최초 작성된 날짜(reported date)로 구성된다.

4. 사례연구

4.1 SPEM 2.0 모델링

Eclipse Process Framework(EPF) Composer[11]는 SPEM 2.0[12-13] 프로세스 메타-모델링 언어를 지원하는 Eclipse 기반 오픈 소스 도구이며 본 연구에서 제안하는 테스트 프로세스를 모델링 할 때 사용되었다. 3 장에서 언급하였던 상위명세 테스트 계획 단계, 상세명세 테스트 계획 단계, 단위명세 테스트 계획 단계, 단위명세 테스트 단계, 상세명세 테스트 단계, 상위명세 테스트 단계를 EPF 의 단계(phase)로 정의하여 모델링 했다.

그림 5 는 3.1 장에서 제시한 모델의 시스템 테스트와 통합 테스트, 단위 테스트를 수행하는 상위/상세/단위명세에 해당하는 테스트 계획 단계와 테스트 단계에 대한 모델이다. 3 장의 정의와 같이 소프트웨어 요구사항 명세서에 수정이 생기면 다시 상위명세 테스트 계획 단계부터 수행한다.

제안하는 테스트 프로세스에서는 상세명세 테스트 계획, 단위명세 테스트 계획, 단위명세 테스트, 상세명세 테스트 4 개의 단계에 대한 반복을 표현하기 위해 결정 노드(decision node)를 사용하여 표현한다. 해당 결정 노드는 테스트 완료 기준을 만족하는지에 대한 여부에 따라 테스트 진행을 결정한다. 따라서 문서 산출물에 대한 테스트 작업을 포함하는 것 외에도 반복 작업을 통해 기존의 V 모델과의 차이점을 볼 수 있다.

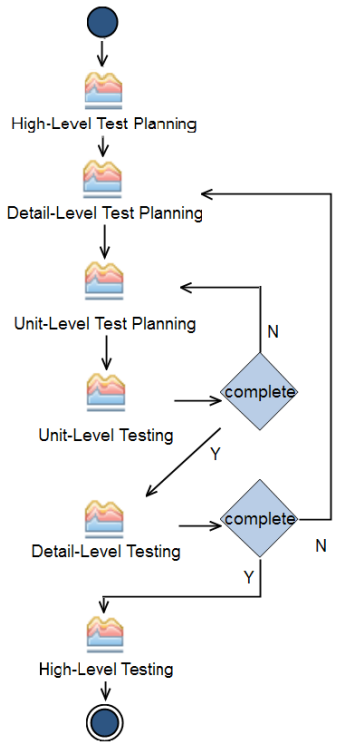


그림 5 테스트 프로세스의 단계 모델링

그림 5 에서 상위/상세/단위명세 테스트 계획 단계에서는 표 1 의 테스트 계획 단계에 속한 각각의 작업을 수행하고, 생성된 산출물은 표 4 에 해당하는 SLTP 로 나온다. 상위/상세/단위명세 테스트 단계는 테스트 단계를 나타내는 표 2 의 작업을 수행하며 테스트 설계 및 구현 활동에서 표 5 에서 나타내는 SLTS 가 생성되며, 나머지 활동의 최종 산출물은 개발 산출물과 연구 산출물에 따라 표 6, 표 7 에 해당하는 SLTR 이 생성된다.

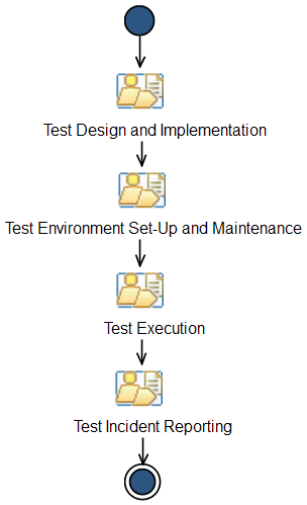


그림 6 테스트 단계를 구성하는 활동의 순서

단계들의 내부 구성을 보여 주기 위한 한 예제로 테스트 단계의 활동을 순차적으로 모델링 한 그림 6 은 테스트 단계를 보여주는 표 2 와 동일하게 구성된 것을 보여준다.

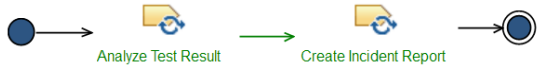


그림 7 테스트 결함 보고서 작성 활동의 작업

활동 안에 포함되는 작업의 순서는 그림 7 과 같이 모델링 된다. 이 그림은 테스트 결함 보고서 작성 활동의 테스트 결과 분석 작업과 테스트 결과 보고서 생성 작업이 순차적으로 진행되는 것을 모델링 한 것이다.

각 작업을 진행하는 역할과 산출물을 설정하면 그림 8 에서 보이는 바와 같고, 역할은 필수(primary)와 추가(additional) 2 가지로 분류한다. 산출물은 각 작업의 입력(input)과 출력(output) 산출물이 있으며 입력 산출물은 필수 입력 산출물(mandatory input)과 추가 입력 산출물(optional input)로 구분한다. [14-15]

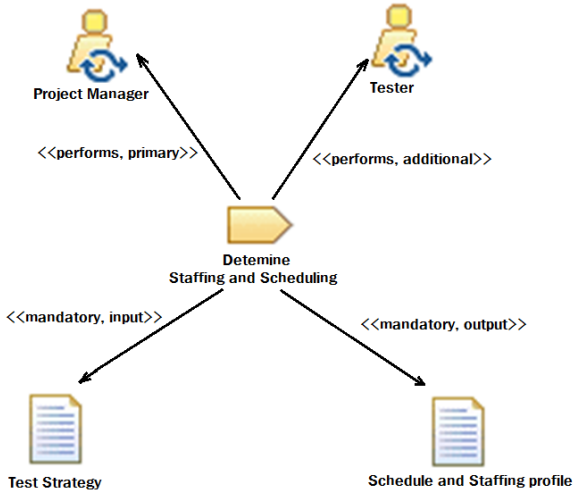


그림 8 작업의 역할과 산출물 정의

4.2 프로세스 실행 : 활동과 산출물 예시

본 절에서는 제안한 소프트웨어 R&D 프로젝트의 테스트 프로세스를 EPF 에서 SPEM2.0 기반으로 모델링 한 후에 uEngine BPMS 에 적용한 사례를 보여준다. uEngine 은 업무 프로세스를 수행하고 관리 할 수 있는 오픈 소스 도구이다.

표 9 SPEM 과 uEngine 의 대응 항목

SPEM 2.0	Phase	Task	Work Product	Role
uEngine BPMS	Sub Process	Form Activity, Human Activity	Variable	Role

표 9 에서 SPEM 과 uEngine 의 대응되는 항목에 대해 보여주고 있다. 서브 프로세스(sub process)는 단계에 대응하고, SPEM 에서 작업과 유사한 기능인 서식 활동(form activity)이나 사람 활동(human activity) 같은 활동들로 구성한다. SPEM 의 산출물은 uEngine 에서 변수(variable)로 정의한다. 변수에는 서식을 비롯한 산출물과 각종 데이터들이 존재한다. 마지막으로 역할은 uEngine 과 SPEM 이 동일한 내용으로 대응된다.

그림 9 는 uEngine 의 서브 프로세스로 적용한 사례를 보여준다. 상위/상세/단위명세 테스트 계획을 순차적으로 수행한 뒤 단위명세 테스트의 완료 기준을 만족한다면 상세명세 테스트를 수행한다. 마찬가지로 상세명세 테스트의 완료 기준이 충족되면 마지막으로 상위명세 테스트를 수행한다.

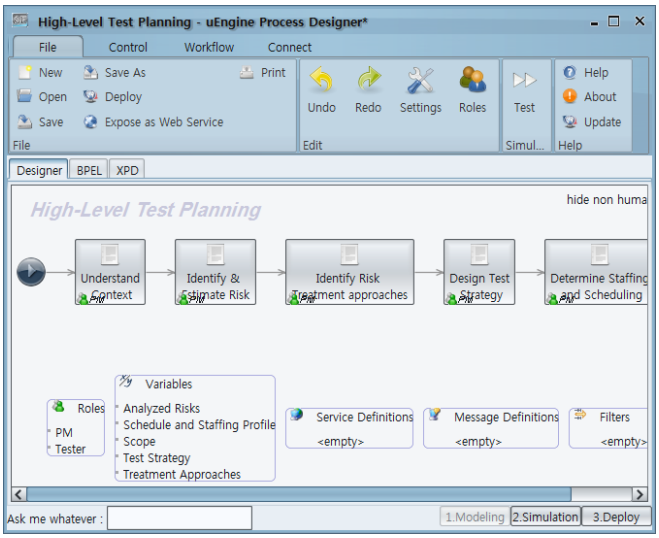


그림 10 uEngine 프로세스 작성 과정

상위명세 테스트 계획 단계의 작업을 보여주는 그림 10 은 3 장에서 정의한 내용과 일치하는 계획 활동의 작업인 배경 이해와 위험 요소 추정 및 식별, 위험 요소 해결 방법 식별, 테스트 전략 설계, 팀원 및 일정에 대한 결정으로 정의됨을 보인다.

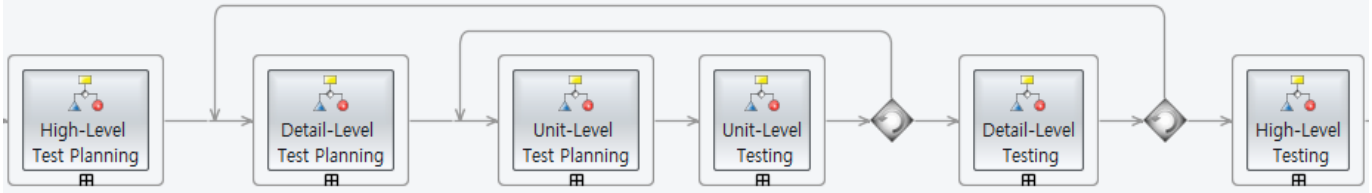


그림 9 uEngine 에서 테스트 프로세스

PM 과 테스터에 대한 역할을 등록하고, 미리 uEngine 에서 서식으로 만들어둔 범위, 분석된 위험 요소, 해결 방법, 테스트 전략, 팀원 및 일정 같은 산출물이나 필요한 데이터를 변수에 등록한다. 그 후에 정의한 프로세스에 맞는 사람 활동이나 서식 활동 같은 활동을 등록하고 각 작업들의 속성을 설정하여 모델링 한다.

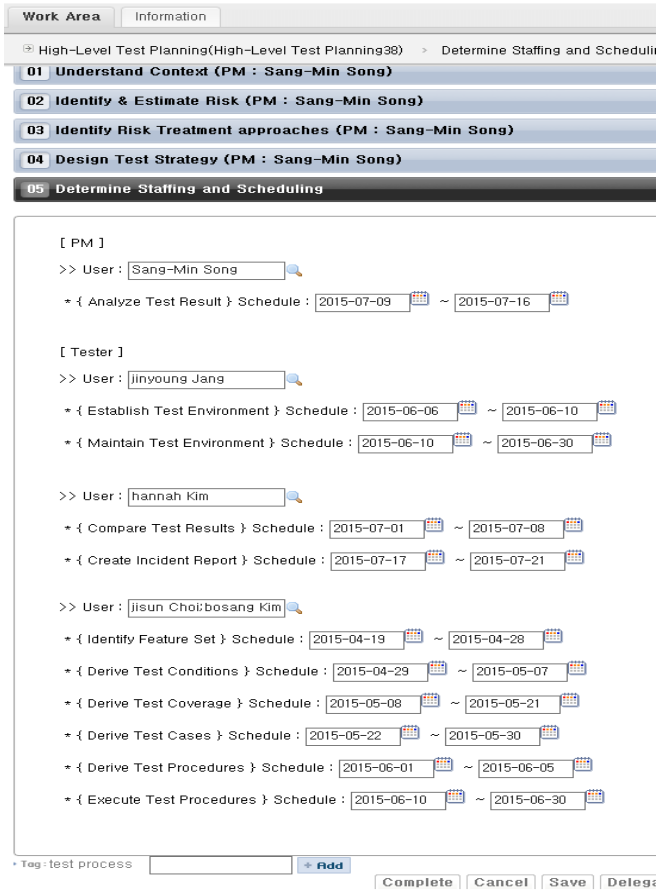


그림 11 프로세스 수행 과정

프로세스의 수행 과정을 보여주는 그림 11 에서는 그림 10 과 같은 계획 활동의 팀원 및 일정에 대한 결정 작업이 수행되며 팀원 및 일정 산출물에 대한 내용을 작성하는 과정이다. 해당 작업에서는 각 역할별로 담당자를 배정하고, 담당자 마다 작업과 기간에 대해 할당하는 작업이다.

5. 토 의

본 연구에서는 제안한 테스트 프로세스를 정의하며 제안한 테스트 프로세스 모델을 활용한 적용 사례를 보여주었다. 하지만 SPEM 2.0 기반의 오픈 소스 모델링 도구를 사용하고 프로세스를 실행할 때는 모델링한 도구가 아닌 다른 프로세스 엔진을 사용했다. 추후에 SPEM 기반의 도구에서 직접 모델링도 하고 바로 프로세스에 적용할 수 있는 엔진을 구현하거나, EPF 에서 uEngine 으로 모델링 데이터를 자동으로 변환할 수 있는 도구를 개발하면 프로세스 자동화에 도움이 될 것이다.

3.3.3 장에서 제시한 소프트웨어 분야 R&D 프로젝트의 테스트 보고 기술인 SLTR 에서는 발생한 결함에 대한 정보를 포함하는 결함 보고서가 생성이 된다. 하지만 각 테스트 결과에 대한 결함을 보여줄 뿐 체계적으로 결함에 대한 데이터 관리를 통해 결함에 대한 수정 시간을 예측하거나 발생 위치를 추적하고 적절한 담당자를 배정하는 결함 관리 기술을 도입한다면 프로세스에서 반복을 진행 할수록 더 빠르고 효율적인 테스트 작업이 가능할 것이다.

상세/단위명세의 테스트 계획 단계와 테스트 단계는 주기적으로 반복하여 프로젝트를 지속적으로 검증한다. 하지만 상위명세 테스트 계획 단계와 상위명세 테스트 단계는 프로젝트에 대해 한 번 밖에 수행하지 않는다. 만약 소프트웨어 요구사항 명세서 같은 상위명세 단계에서 활용하는 문서에 수정이 발생한다면 다시 상세명세 테스트 계획 단계부터 실행하여 변경사항에 유연하게 대처할 수 있다.

6. 결 론

본 논문에서는 R&D 프로젝트의 테스트 프로세스와 소프트웨어 개발 프로세스 및 검증 모델을 분석하여 소프트웨어 R&D 프로젝트에 맞는 테스트 프로세스에 대해 정의하였다. 상위명세, 상세명세, 단위명세 단계로 구성된 테스트 프로세스의 활동과 작업에 대해 자세한 내용을 기술했으며, 각 작업에 맞는 산출물과 역할을 정의하였다. 그리고 각 단계별 생성되는 최종 산출물에 대한 정의와 산출물간의 관계에 대해 정의했다.

또한 본 연구에서 제안하는 테스트 프로세스에 따라 오픈 소스 도구를 활용하여 모델링과 업무프로세스 관리 도구를 활용하면 연구자들이 연구에 적용하거나 프로젝트의 특성에 맞는 프로세스로 개선하여 활용이 가능하다.

향후 SLTP 와 SLTS, SLTR 을 체계적인 시스템으로 구현하여 소프트웨어 분야 R&D 프로젝트의 산출물 테스트 계획 및 테스트 명세와 보고에 적용함으로써 본 연구에서 제안한 방법의 완성도를 높이고자 한다. 그리고 EPF 에서 모델링한 데이터를 업무프로세스 관리 도구로 바로 적용할 수 있는 변환 도구를 개발하여 테스트 프로세스의 자동화를 지원할 계획이다.

Acknowledgement

이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업(NRF-2014M3C4A7030504)과 서울시의 재원으로 수행한 서울시 창조전문인력 양성사업(CAC15106)의 지원을 받아 수행된 연구임.

참고문헌

[1] J. Schumann and W. Visser, "Autonomy software: V&V challenges and characteristics," In Proc. of 2006 IEEE Aerospace Conference, pp. 1233-1249, 2006.

[2] T. Abdou, P. Grogono and P. Kamthan, "A conceptual framework for open source software test process," In Proc. of Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual, IEEE, pp. 458-463, 2012.

[3] S. Imoto, Y. Yabuuchi and J. Watada, "Fuzzy regression model of R&D project evaluation," In Applied Soft Computing, vol. 8 no. 3, pp. 1266-1273, 2008.

[4] 정학선, 김희천, "JIRA 도구를 이용한 테스트 프로세스 관리," 한국정보과학회 2015 한국컴퓨터종합학술대회 논문집, pp. 600-602, 2015.

[5] S. Matalonga, F. Rodrigues and G. H. Travassos, "Matching Context Aware Software Testing Design Techniques to ISO/IEC/IEEE 29119," In Proc. of Software Process Improvement and Capability Determination, Springer International Publishing, pp. 33-44, 2015.

[6] J. Kasurinen, P. Runeson, L. Riungu and K. Smolander, "A self-assessment framework for finding improvement objectives with ISO/IEC 29119 test standard," In Proc. of Systems, Software and Service Process Improvement, Springer Berlin Heidelberg, pp. 25-36, 2011.

[7] A. Dashbalbar, E. Lee, J. Lee and B. Lee, "A Test Process for Real Monitoring in Software R&D Project," In Proc. of 10th Asia Pacific International Conference on Information Science and Technology (APIC-IST), 2015.

[8] IEEE Standard for Software and System Test Documentation, the Institute of Electrical and Electronics Engineers, 2008.

[9] 백두산, 이정원, "SW 연구 개발 문서 산출물 간의 대응 항목 추적을 위한 연관성 분석," 한국소프트웨어공학 학술대회 논문집, vol. 17, no. 1, pp. 13-20, 2015.

[10] 진광희, 송상민, 이정원, 이병정, "소프트웨어 R&D 프로젝트의 상시 모니터링을 위한 테스트 계획 및 보고," 한국정보과학회 2015 한국컴퓨터종합학술대회 논문집, pp. 597-599, 2015.

[11] Y. K. Chiam, M. Staples and L. Zhu, "Representation of Quality Attribute Techniques Using SPEM and EPF Composer," In Proc. of European Software Process Improvement, EuroSPI, 2009.

[12] T. Mäkilä and A. Järvi, "Spemmet–A Tool for Modeling Software Processes with SPEM," In Proc. of the 9th International Conference on Information Systems Implementation and Modelling, 2006.

[13] V. Seidita, M. Cossentino and S. Gaglio, "Using and extending the SPEM specifications to represent agent oriented methodologies," In Proc. of Agent-Oriented Software Engineering IX, Springer Berlin Heidelberg, pp. 46-59, 2009.

[14] R. Bendraou, B. Combemale, X. Crégut and M. P. Gervais, "Definition of an Executable SPEM 2.0," In Proc. of Software Engineering Conference, APSEC 14th Asia-Pacific. IEEE, pp. 390-397, 2007.

[15] A. Rochd, M. Zrikem, A. Ayadi, T. Millan, C. Percebois and C. Baron, "SynchSPEM: A synchronization metamodel between activities and products within a SPEM-based Software Development Process," In Proc. of Computer Applications and Industrial Electronics (ICCAIE), 2011 IEEE International Conference, pp. 471-476, 2011.

국가 사이버안보 역량 강화를 위한 사이버보안 성숙도 모델 설계

이민재

(주)티큐엠에스
서울 서초구 바우뫼로 135-4
mjl22@tqms.co.kr

손영동

고려대학교 정보보호대학원
서울 성북구 안암로 145
viking@paran.com

요약: 세계 각국은 사이버안보를 국가안보의 핵심과제로 간주하고 사이버대응 역량 강화에 속도를 내고 있다. 한국 또한 사이버안보의 중요성을 인식하고, 2013 년 7 월에 '국가 사이버안보 종합대책'을 발표했다. 사이버안보 강국 실현을 위해서는 사이버테러 대응능력을 강화하고 사이버전 억지력을 확보해야 하는데, 이를 위해서는 사람과 프로세스와 기술이 상호 유기적으로 작용하는 사이버보안 체계의 구축이 필요하다. 그러나 종합대책을 보면, 사이버보안을 위한 전문가를 양성하고 기술을 개발하는 데에만 초점을 맞추고 있다. 사이버안보 역량을 강화시켜 나가는 데 필요한 프로세스는 언급하고 있지 않다. 사이버보안 사고 대부분이 프로세스 부재와 인력관리 미흡으로 발생하고 있는 현실을 감안할 때, 사이버보안을 실행하기 위한 프로세스는 반드시 필요하다. 금번 연구에서는 그 동안 국내외에서 이뤄진 사이버역량 평가 및 성숙도 모델 관련 문헌연구를 통해 각 각의 장단점을 분석하고 한국형 사이버보안 성숙도 모델(Cyber Security Maturity Model, CSMM)의 구조와 구성을 설계했다. CSMM 은 사이버 역량의 핵심인 공격, 방어, 기반시설을 포함하여 전체 4 개 성숙도 단계, 4 개 프로세스 범주에 18 개 프로세스 영역으로 정의했다.

핵심어: 사이버공간, 사이버보안, 사이버역량, 사이버전(戰), 성숙도 모델, 프로세스

1. 연구 배경

미래학자 앨빈 토플러는 그의 저서 「전쟁과 반전쟁」에서 "인간은 일하는 방식대로 전쟁을 수행하고 인류의 역사는 전쟁의 역사다."라고 언급했다[1]. 시스코(Cisco)는 네트워크에 연결된 사물의 수가 2008 년에는 65 억 개에 불과했지만, 2015 년에는 250 억 개, 2020 년에는 500 억 개에 달할 것이라 예측했다[2]. 우리는 네트워크상에서 일하고 있고 전쟁 또한 네트워크상에서 이뤄지는 사이버전(戰)이 된다는 의미다. 국제연합(UN) 또한 "제 3 차 세계대전이 일어난다면 그것은 사이버전이 될 것이며, 그 어떤 국가도 성역으로 남을 수 없다."라고 경고했다.
세계 각국은 사이버안보를 국가안보의 핵심과제로 간주하고 사이버대응 역량강화에 속도를 내고 있다. 한국은 지난 2013 년 발생한 3 · 20 과 6 · 25 사이버테러로 인해 약 4,400 억~8,000 억 원에 이르는 직간접적인 피해를 입었다. 이후에야 비로소 범국가차원의 역량을 결집하여 체계적으로 대응하고자 같은 해 7 월에 '국가 사이버안보 종합대책'을 발표했다. 종합대책은 "사이버안보 강국 실현"이라는 목표 아래 첫째, 사이버위협 대응체계 즉응성 강화, 둘째, 유관기관 스마트 협력체계 구축, 셋째, 사이버공간 보호대책 견고성 보강, 넷째, 사이버안보 창조적 기반 조성이라는 네 가지 주요 골자로 이뤄져 있다[3].
사이버안보 강국 실현을 위해서는 사이버테러 대응능력을 강화하고 사이버전 억지력을 확보해야 하는데, 이를 위해서는 사람과 프로세스와 기술이 상호 유기적으로 작용하는 사이버보안 체계의 구축이 필요하다. 그러나 종합대책을 보면, 사이버보안을 위한 전문가를 양성하고 기술을 개발하는 데에만 초점을 맞추고 있다. 사이버안보 역량을 강화시켜 나가는 데 필요한 프로세스는 언급하고 있지 않다. 사이버보안 사고 대부분이 프로세스 부재와 인력관리 미흡으로 발생하고 있는 현실을 감안할 때, 사이버보안을 실행하기 위한 프로세스는 반드시 필요하다.
미국은 지난 1988 년에 국방부 산하 소프트웨어공학연구소 내에 국가침해대응센터를 설치하고, 국토안보부와 연계하여 사이버보안 및 위험 등과 관련한 다양한 프로세스 모델을 연구하여 활용하고 있다.
우리도 국가 사이버안보를 위한 사이버보안 프로세스 모델 개발 및 활용이 필요한 시점이다.
금번 연구에서는 그 동안 국내외에서 이뤄진 사이버역량 평가 및 성숙도 모델 관련 문헌연구를 통해 각 각의 장단점을 분석하고 한국형 사이버보안 성숙도 모델(Cyber Security Maturity Model, CSMM)의 구조와 구성을 설계했다.

2. 관련 연구

2.1 사이버역량 평가 연구

미국의 테크놀리틱스社는 지난 2009 년에 160 개 국가의 군 사이버역량을 다음과 같은 3 가지 기준으로 평가했다[4].

•사이버역량 목적(cyber capabilities intent): 목적달성을 위한 목표와 심리상태
•사이버공격 역량(offensive cyber capabilities): 전시 특수목적을 달성하기 위한 능력(군 조직・기술우위・준비도・지속성)
•사이버 정보수집 등급(cyber intelligence rating): 새로운 사이버 영역에서의 정보수집 적응력

미 군사전문가 리차드 클라크는 2010 년에 북한, 미국, 중국, 러시아, 이란의 사이버역량을 다음과 같은 3 가지 유형으로 구분하여 평가했다[5].
•공격(offense): 타 국가를 공격할 수 있는 능력
•방어(defense): 공격에 대한 저지 및 완화 능력
•의존(dependence): 국가 기반시설이 네트워크에 연결된 정도

한국 국가보안기술연구소는 2012 년에 한국, 미국, 중국, 일본, 러시아의 사이버역량을 평가했다[6]. 공격역량・방어역량・기반역량의 3 개 대 분류로 구분하고 각 역량별 평가 항목 점수를 합산하여 각 국의 역량 점수를 계산했다.
그러나 위의 3 가지 평가 방법은 평가 항목만 정의했을 뿐, 프로세스는 정의하지 않았다.

2.2 사이버보안 역량 성숙도 모델 연구

미국은 지난 2014 년 2 월에 사이버보안 역량 성숙도 모델(Cybersecurity Capability Maturity Model, C2M2) 버전 1.1 을 발표했다[7]. 백악관의 지시 하에 국토안보부와 에너지국이 조직의 사이버보안 역량을 강화시키는데 필요한 지침을 제공하기 위해 개발한 것으로 4 개 성숙도 단계(Maturity Indicator Level, MIL), 10 개 도메인으로 구성돼 있다. 각 도메인은 하나 이상의 접근목표(approach objectives)와 하나의 관리목표(management objective)를 보유한다. 각 도메인 별로 성숙도 단계를 달성하는 구조이나 성숙도 단계별로 프로세스가 정의되어 있지는 않다.
영국 옥스포드대학 글로벌사이버보안역량센터는 정부와 유관기관의 사이버역량을 강화시키는데 필요한 지침을 제공하기 위해 사이버보안 역량 성숙도 모델(Cyber Security Capability Maturity Model, CMM)을 발표했다[8]. 2014 년 12 월 15 일 파일럿 버전을 발표했는데, 5 개 성숙도 단계(Maturity Level, ML)에 5 개 차원으로 구성돼 있다. 각 차원은 3~6 개 하위차원으로 구분되며, 각 하위차원은 1~5 개의 범주로 분류된다. 아직까지는 파일럿 버전이라 5 개 차원별, 성숙도 단계별 개념만을 제시하는 수준이다.

2.3 기타 관련 모델 연구

미 국방부 산하 소프트웨어공학연구소의 침해대응센터는 지난 2011 년 10 월, 써트 레질리언스 관리 모델(CERT Resilience Management Model, CERT-RMM) 버전 1.1 을 발표했다[9]. 레질리언스(복원력) 활동의 이행 및 관리에 필요한 지침을 제공하기 위해 개발한 것으로 보안, 사업 지속성, 재해복구 및 IT 운용의 컨버전스를 위한 프랙티스를 제공한다. 4 개의 프로세스 범주와 26 개의 프로세스 영역으로 구성되며, 성숙도 단계가 아닌 4 개의 역량수준으로 표현된다. 보안, 사업 지속성, 재해복구 및 IT 운용의 레질리언스 전반에 대한 포괄적인 내용으로 정의되어 있으나, 사이버보안에 초점을 맞춘 개념과 용어는 사용되지 않았다.
정보보호관리체계(Information Security Management Systems, ISMS)는 조직의 정보보호관리체계 수립・구현・유지 및 개선을 위한 규정인 ISO/IEC27001 에 기반한 지침이다[10]. 한국은 「정보통신망이용촉진 및 정보보호 등에 관한 법률」 제 47 조에 근거하여 정보보호관리체계 인증제도를 운영하고 있다. 정보보호관리체계는 5 단계 12 개 통제항목으로 구성된 '정보보호 관리과정'과 13 개 분야 92 개 통제항목으로 구성된 '정보보호대책'으로 구분되며 성숙도 단계나 역량수준의 구분은 없다. 전반적인 정보보호 관리 활동 통제에 초점을 맞추고 있으나, 사이버역량의 핵심인 공격 및 방어기술 개발 영역은 명시적이지 않다.

2.4 연구결과 분석

그 동안 국내외에서 이뤄진 사이버역량 평가 및 성숙도 모델 관련 문헌을 연구하고 표 1 과 같이 각 각의 장단점을 분석하여 정리했다.

참조모델	특장점
사이버역량 평가	• 사이버역량을 공격, 방어, 기반영역으로 구분 • 영역별 평가항목만 정의하고 프로세스는 정의되지 않음 • 모델 개발 시, 사이버역량의 핵심 영역을 공격, 방어, 기반시설로 정의 • 제시된 평가항목은 모델 내, 관련 기법의 예시로써 활용 • 프로세스 정의를 위한 참고용으로는 활용 안 함
C2M2	• 4 개 성숙도 단계, 10 개 도메인으로 구성 • 사이버보안에 초점을 맞춘 개념과 용어 사용 • 각 도메인 별로 성숙도 단계를 달성하는 구조 • 성숙도 단계별로 프로세스를 구분하지 않음 • 모델 개발 시, 사이버보안 개념과 용어 활용 • 제시된 도메인을 프로세스 정의에 활용 • 성숙도 단계 개념 반영

CMM	• 파일럿 버전으로 5 개 성숙도 단계, 5 개 차원으로 구성 • 각 차원별, 성숙도 단계별 개념제시 수준 • 성숙도 단계별로 프로세스를 구분하지 않음 • 범주별 수행을 위한 프랙티스를 정의하지 않음 • 모델 개발에 활용하지 않음
CERT-RMM	• 4 개 프로세스 범주, 26 개 프로세스 영역으로 구성 • 성숙도 단계가 아닌 4 개 역량단계로 구성 • 보안, 사업 지속성, 재해복구 및 IT 운용의 레질리언스 전반에 대한 포괄적인 내용으로 정의 • 사이버보안에 초점을 맞춘 개념과 용어가 사용되지 않음 • 모델 개발의 기본문서로 프로세스 정의에 활용 • 역량단계 개념 반영 • 사이버보안에 초점을 맞춘 개념과 용어로의 변경 필요
ISMS	• 정보보호 관리과정과 정보보호대책으로 구성 • 사이버역량의 핵심인 공격과 방어 영역은 명시적이지 않으나 기반시설 영역은 폭넓게 다루고 있음 • 성숙도나 역량단계 구분 없음 • 모델 개발 시, 기반시설 내용을 프로세스 정의에 활용

표 1 참조모델간 특장점 비교

분석 결과로써, 한국형 사이버보안 성숙도 모델 설계는 CERT-RMM 을 기본 참조모델로 하고 사이버보안 개념과 용어는 C2M2 를 기반시설 관련 사항은 ISMS 를 활용하기로 했다.
참조모델간 특장점 분석을 통해 그림 1 과 같이 프로세스, 조직, 전략 및 운용이 융합된 사이버보안 성숙도 모델 개발 방향을 설정했다.

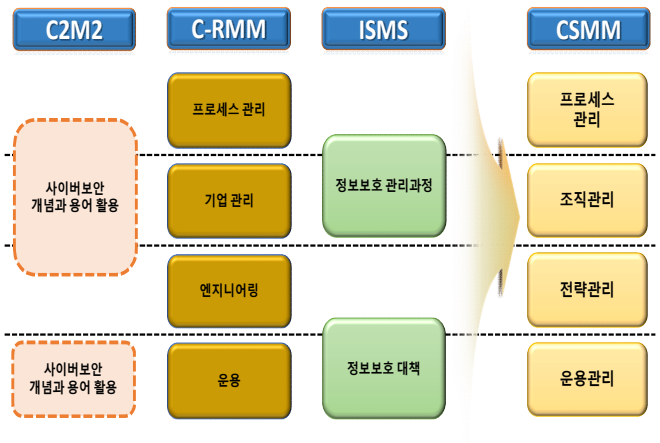


그림 1 한국형 사이버보안 성숙도 모델 개발 방향

3. CSMM 설계

3.1 CSMM 프로세스 구조

각 참조모델의 도메인, 프로세스 및 통제항목 내용을 검토하여 중복되거나 유사한 내용은 통합하고 누락된 내용은 추가하여 전체 18 개 프로세스 영역으로 조정 및 선정하였다. 선정된 18 개 프로세스 영역에 대한 상세 내용은 표 2 와 같다.

C2M2	ISMS	CERT-RMM	CSMM	
		측정 및 분석	사이버보안 측정 및 분석	프로세스 관리
상황인지		모니터링	사이버보안 정보관리	
사이버보안 프로그램 관리	정보보호정책 수립 및 범위설정 정보보호정책	조직 프로세스 정의	사이버보안 프로세스 정의	
		조직 프로세스 개선		
정보공유 및 의사소통		의사소통	사이버보안 프로세스 관리	
	사후관리	제 규정 준수		
사이버보안 프로그램 관리	경영진책임 및 조직구성 정보보호 조직	기업 중점관리	사이버보안 조정관리	조직관리
작업인력 관리		인력관리	사이버보안 인력관리	
	인적 보안	인적자원 관리	사이버보안 교육훈련	
	정보보호 교육	조직 훈련 및 인식		
위험관리	위험관리	위험관리	사이버보안 위험관리	
자산, 변경 및 형상관리	정보자산 분류	자산 정의 및 관리	사이버보안 자산관리	전략관리
		금융자원 관리		
		통제관리	사이버보안 전략관리	
		복원력 요구사항 개발	사이버보안 요구사항 개발	
		복원력 요구사항 관리		
	시스템 개발 보안	복원력 기술 솔루션 엔지니어링	사이버보안 기술관리	
		기술관리		
운용 지속성	IT재해복구	서비스 지속성	사이버보안 서비스 지속성	운용관리
	접근통제 운용보안	접근권한 관리	사이버보안 운용통제	
	물리적 보안 운용보안	환경통제	사이버보안 공급자 협약	
공급망 및 외부 의존성 관리	외부자 보안	대외 의존성 관리		
신원 및 접근관리	암호통제	신원관리	사이버보안 침해사고 관리	
침해사고 및 대응	침해사고 관리	침해사고 관리 및 통제		
	운용보안	지식 및 정보관리	사이버보안 지식 및 정보	
위협 및 취약점 관리	정보보호대책 구현	취약점 분석 및 해결	사이버보안 취약점 분석	

표 2 CSMM 프로세스 선정

선정된 프로세스 영역은 프로세스의 활용을 높이기 위해 그림 2 와 같이 4 개의 프로세스 범주로 그룹화 하였다.

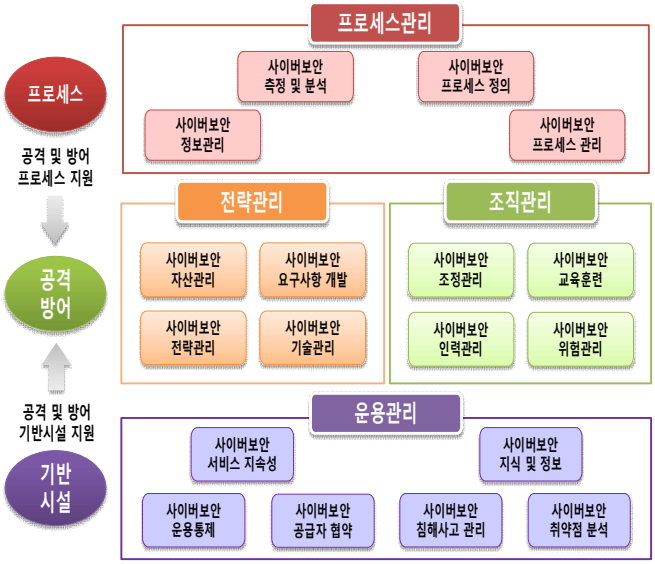


그림 2 CSMM 프로세스 범주

CSMM 은 프로세스, 조직, 전략 및 운용관리 활동을 지원하는 프로세스 영역들로 분류되며 각 프로세스 범주의 특성은 다음과 같다.

•프로세스 관리: 조직의 사이버보안 프로세스 관리를 위한 프로세스 영역으로 구성

•조직 관리: 조직차원의 사이버보안 활동 수행을 위한 프로세스 영역으로 구성

•전략 관리: 사이버보안 전략 및 전술을 수행하기 위한 프로세스 영역으로 구성

•운용 관리: 사이버보안 기반시설 운용을 위한 프로세스 영역으로 구성

각 프로세스 영역에 대한 상세 내용은 표 3, 4, 5 및 6 과 같다.

범주	프로세스 영역	프로세스 목적	참조모델
프로세스 관리	**사이버보안 측정 및 분석** (CSMA, Cyber Security Measurement and Analysis)	• 사이버보안 프로세스를 관리하기 위해 경영진이 필요로 하는 정보 요구를 지원하기 위한 측정 역량을 개발하고 활용하는 것	C-RMM
	사이버보안 정보관리 (CSIM, Cyber Security Information Management)	• 사이버공격・방어, 기반시설과 관련한 대외 사이버보안 정보를 정기적으로 수집・분석・경고・제공 및 활용하기 위한 활동과 기술을 개발하는 것	C2M2 C-RMM
	사이버보안 프로세스 정의 (CSPD, Cyber Security Process Definition)	• 조직의 사이버보안 활동을 위해 내부 통제체제와 조직 및 경영진 참여에 대한 기업 사이버보안 프로그램과 사용 가능한 프로세스 자산을 정립하고 활용하는 것 • 이 프로그램은 조직의 전략목표 및 주요 기반시설에 대한 위험과 사이버보안 목표간 일관성을 유지하는 활동을 포함	C2M2 ISMS C-RMM
	사이버보안 프로세스 관리 (CSPM, Cyber Security Process Management)	• 사이버보안 관리를 위해 내외부적으로 정립된 관련 지침・표준・규정・규제・법규 및 계약과 같은 의무를 준수하는 것을 보장하기 위해 정기적으로 내부심사를 수행하고 심사결과에 따라 개선을 계획 및 이행하며 확산적용하는 활동을 수행하는 것	C2M2 ISMS C-RMM

표 3 프로세스 관리 프로세스 영역

범주	프로세스 영역	프로세스 목적	참조모델
조직관리	**사이버보안 조정관리** (CSCM, Cyber Security Coordination Management)	• 사이버보안 활동과 프로세스를 지원하기 위한 내/외부 관련 조직간 관계를 정립하고 유지하는 것 • 이 관계는 주요 기반시설과 조직목표에 대한 위험을 감소시키고 운용 복원력을 증가시키는 위협과 취약점을 포함	C2M2 C-RMM
	사이버보안 인력관리 (CSWM, Cyber Security Workforce Management)	• 사이버보안 관리 활동에 기여할 수 있는 인력의 채용과 그들의 성과를 관리하는 것 • 이 활동은 사이버공격・방어, 기반시설과 조직목표에 대한 위험에 상응하는 인력의 지속적인 적합성과 능력을 보장하는 것을 포함	C2M2 ISMS C-RMM
	사이버보안 교육훈련 (CSET, Cyber Security Education and Training)	• 사이버보안에 대한 조직전반에 걸친 인식을 고취하고 구성원 각자의 역할 및 책임에 부합하는 스킬과 지식을 개발하는 것	ISMS C-RMM
	사이버보안 위험관리 (CSRM, Cyber Security Risk Management)	• 조직의 사이버보안 위험을 식별하고 분석하며 완화하기 위한 사이버보안 위험관리 프로그램을 정립 및 운용하고 활용하는 것 • 이 프로그램은 단위조직, 자회사, 상호 관련된 기반시설과 이해관계자를 포함	C2M2 ISMS C-RMM

표 4 조직 관리 프로세스 영역

범주	프로세스 영역	프로세스 목적	참조모델
전략관리	**사이버보안 자산관리** (CSAM, Cyber Security Asset Management)	• 조직의 IT와 운용기술 및 금융자산을 식별하고 분류하며 관리하고 적용하는 것 • 이 활동은 주요 기반시설과 조직목표에 대한 위험에 상응하는 하드웨어와 소프트웨어 자산을 식별 및 문서화하고 관리하는 것을 포함	C2M2 ISMS C-RMM
	사이버보안 전략관리 (CSSM, Cyber Security Strategy Management)	• 사이버공격・방어, 기반시설과 관련한 사이버보안 요구사항에 기반한 전략 계획을 수립하고 계획에 따른 이행을 모니터링 및 통제하는 것 • 이 활동은 고부가가치 서비스와 이를 지원하는 자산의 확실한 임무 성공을 통해 운용의 효과성과 효율성을 보장하는 것을 포함	C-RMM
	사이버보안 요구사항 개발 (CSRD, Cyber Security Requirements Development)	• 사이버 공격 및 방어, 기반시설과 관련한 요구사항을 식별하고 분석하며 명세화하는 것 • 이 활동은 요구사항과 요구사항에 부합하기 위해 조직에서 수행하는 활동 간의 불일치 식별을 포함	C-RMM
	사이버보안 기술관리 (CSTM, Cyber Security Technology Management)	• 사이버보안 요구사항에 부합하는 소프트웨어와 시스템이 개발됨을 보장하는 것 • 이 활동은 사이버보안을 지원하는 기술 자산의 무결성과 유용성에 대한 적절한 수준의 통제를 정립하고 관리하는 것을 포함	ISMS C-RMM

표 5 전략 관리 프로세스 영역

범주	프로세스 영역	프로세스 목적	참조모델
운용관리	**사이버보안 서비스 지속성** (CSSC, Cyber Security Service Continuity)	• 주요 기반시설과 조직목표에 대한 위험에 상응하는 사건이나 재해 또는 여하한 업무중단 사태에도 필수적인 서비스 및 이와 관련된 자산의 지속적인 운용을 보장하는 것	C2M2 ISMS C-RMM
	사이버보안 운용통제 (CSOC, Cyber Security Operations Control)	• 물리적이고 환경적이며 지역적인 조직 기반시설에 대한 적절한 수준의 접근 통제를 정립하고 관리하는 것 • 이 통제는 조직의 자산에 접근하거나 접근을 관리하는 적절한 수준의 신원을 만들고 유지하며 정지시키는 활동을 포함	C2M2 ISMS C-RMM
	사이버보안 공급자 협약 (CSSA, Cyber Security Supplier Agreement)	• 외부 공급자의 활동에 의존적인 사이버보안 서비스와 자산 관리를 위한 적절한 수준의 통제를 정립하고 관리하는 것 • 이 활동에는 외부 공급자에 대한 보안 요구사항을 정의하고 이행하는 것을 포함	C2M2 ISMS C-RMM
	사이버보안 침해사고 관리 (CSIM, Cyber Security Incident Management)	• 사이버보안 침해사고를 식별, 분석 및 탐지하고 이에 대한 조직의 적절한 대응을 결정하는 프로세스를 정립하고 활용하는 것	C2M2 ISMS C-RMM
	사이버보안 지식 및 정보 (CSKI, Cyber Security Knowledge and Information)	• 조직의 사이버보안 정보와 필수 문서 및 지적 자산에 대한 비밀유지와 무결성 및 유용성을 지원하는 적절한 수준의 통제를 정립하고 관리하는 것 • 이 활동에는 각종 저장매체에 의한 관리와 이에 대한 백업관리를 포함	ISMS C-RMM
	사이버보안 취약점 분석 (CSVA, Cyber Security Vulnerability Analysis)	• 조직의 주요 IT 및 운용관련 기반시설과 목표에 대한 위협에 상응하는 사이버보안 위협 및 취약점을 탐지, 식별, 분석 및 관리하고 대응하는 것 • 이 활동에는 관련 계획 및 절차와 기술을 개발하고 활용하는 것을 포함	C2M2 ISMS C-RMM

표 6 운용 관리 프로세스 영역

표 6 운용 관리 프로세스 영역

3.2 CSMM 프로세스 구성

CSMM 의 각 프로세스 영역은 그림 3 과 같이 구성돼 있으며, 구성요소 별 상세 내용은 다음과 같다.

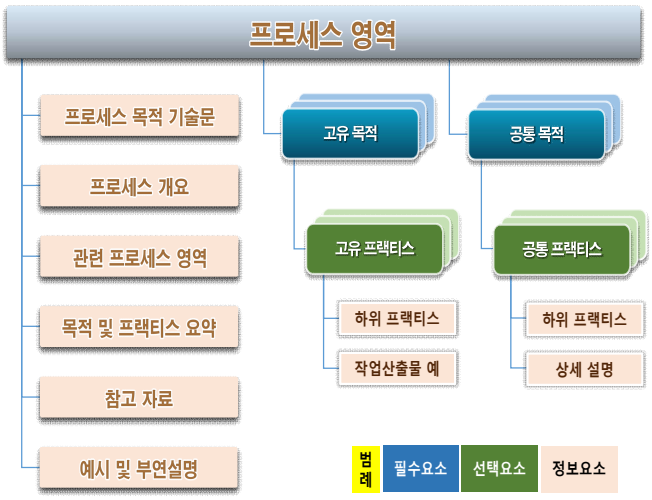


그림 3 CSMM 프로세스 구성 요소

•프로세스 영역: 관련된 프랙티스들의 모음으로 이들이 함께 이행될 때 해당 영역의 개선을 이루는 데, 중요하다고 여겨지는 목적들을 만족
•프로세스 목적 기술문: 해당 프로세스 영역의 목적을 기술
•프로세스 개요: 해당 프로세스 영역에서 다뤄지는 주요 개념을 기술
•관련 프로세스 영역: 관련된 프로세스 영역에 대한 참조가 열거되고 프로세스 영역들 간의 높은 수준의 관계가 반영
•고유 목적: 해당 프로세스 영역을 만족하기 위해 반드시 존재해야 하는 고유한 특성을 기술
•공통 목적: 프로세스 영역을 구현하는 프로세스들을 내재화하기 위해 반드시 존재해야 하는 특성을 기술
•목적 및 프랙티스 요약: 해당 고유 목적과 고유 프랙티스에 대한 간단한 요약을 제공
•고유 프랙티스: 관련 고유 목적 달성에 중요하다고 여겨지는 활동에 대한 기술로 프로세스 영역의 고유 목적 달성으로 귀결될 것으로 기대되는 여러 활동들을 기술
•작업산출물 예: 고유 프랙티스의 산출물 예시를 제공
•하위 프랙티스: 고유 또는 공통 프랙티스를 해석 및 이행하는 데 필요한 지침을 제공
•공통 프랙티스: 공통 목적을 달성하는 데 있어 중요한 것으로 여겨지고 프로세스 영역과 관련된 프로세스의 내재화에 기여하는 활동들을 기술
•상세 설명: 공통 프랙티스를 각 프로세스 영역에 어떻게 개별적으로 적용할 수 있는지에 대한 가이드 제공
•참고 자료: 세부 정보, 배경 설명 또는 이론적 근거를 기술
•예시 및 부연설명: 개념 또는 기술된 활동을 명확히 하기 위해 한 개 이상의 예나 설명을 제공

3.3 역량수준 및 성숙도 단계

CSMM 은 지난 20 여 년간 전 세계 산업군에서 사용하여 효과가 입증된 CMMI 평가방법을 채택했다[11]. CMMI 는 단계적 표현(staged representation)과 연속적 표현(continuous representation)의 두 가지 평가방법을 사용하고 있다. 단계적 표현방법은 기존에 정의된 프로세스 집합의 평가를 통해 조직 전체의 능력을 파악할 수 있다. 이에 반해, 연속적 표현방법은 개별 프로세스 영역별 평가를 통해 개별 프로세스 영역 각각에 대한 능력을 파악할 수 있다는 점이 두 가지 표현방법에 있어 가장 큰 차이다.
사이버보안 분야는 매우 빠르게 변화하며, 이로 인해 적용되는 프로세스 영역이 변경될 수 있기에 CSMM 은 단계적 표현방법이 아닌 연속적 표현방법을 사용했다.
CMMI 의 연속적 표현방법에서는 공통 목적 달성도에 따라 0 부터 3 수준까지 4 개의 역량수준으로 구분하는데, CSMM 도 동일 방법을 채택했다.
각 역량수준별 특성은 표 7 과 같다.

역량수준	특성	설 명
0	불완전한 (Incomplete)	• "불완전한 프로세스"는 아직 수행되지 않았거나 부분적으로만 수행된 프로세스를 의미 • 해당 프로세스 영역의 고유 목적 중, 하나 이상의 목적이 충족되지 않은 상태 • 부분적으로 수행되는 프로세스는 내재화될 수 없기에 공통 목적이 없음
1	수행되는 (Performed)	• "수행되는 프로세스"는 작업산출물을 생산하는 데 필요한 작업을 수행한 프로세스를 의미 • 해당 프로세스 영역의 고유 목적을 충족한 상태
2	관리되는 (Managed)	• "관리되는 프로세스"는 정책에 따라 계획 및 실행되고, 통제된 결과물을 생산하기 위해 적절한 자원을 갖춘 숙련된 인력을 고용하며, 관련 이해관계자들을 참여시키고, 모니터링, 통제 및 검토 대상이 되며, 해당 프로세스 기술 준수 여부의 평가 대상이 되는 수행된 프로세스를 의미
3	정의된 (Defined)	• "정의된 프로세스"는 조직의 조정지침에 따라 조직의 표준 프로세스로부터 조정되고, 유지관리되는 프로세스 기술서가 있으며, 조직의 프로세스 자산에 프로세스 관련 경험을 축적하여 관리하는 프로세스를 의미
☞ 역량수준 2는 단위 사업(과제)에 적용하는 프로세스 간 차이가 존재할 수 있음 ☞ 역량수준 3은 단위 사업(과제)에 적용하는 프로세스 간 일부 조정된 차이를 제외하고는 높은 일관성을 유지하고 역량수준 2보다 프로세스를 엄격하게 기술하고 관리		

표 7 역량수준별 특성

CSMM 의 모든 프로세스 영역은 해당 프로세스의 특징을 표현하는 고유 목적과 고유 프랙티스 그리고 프로세스 영역과는 상관없이 공통의 특징을 표현하는 공통 목적과 공통 프랙티스로 구분된다.
고유 목적과 고유 프랙티스는 특정한 프로세스 영역과 연관된 목적과 프랙티스이기 때문에 각 프로세스 영역별로 그 내용이 다르다. 반면 공통 목적과 공통 프랙티스는 CSMM 의 18 개 프로세스 영역 모두에서 공통적으로 사용되는 목적과 프랙티스이기 때문에

내용이 유사하다는 특징을 갖는다.
CMMI 에서는 각 프로세스 영역별로 3 개의 공통 목적과 13 개의 공통 프랙티스를 제시하고 있는데, CSMM 도 동일하며 상세 내용은 그림 4 와 같다.

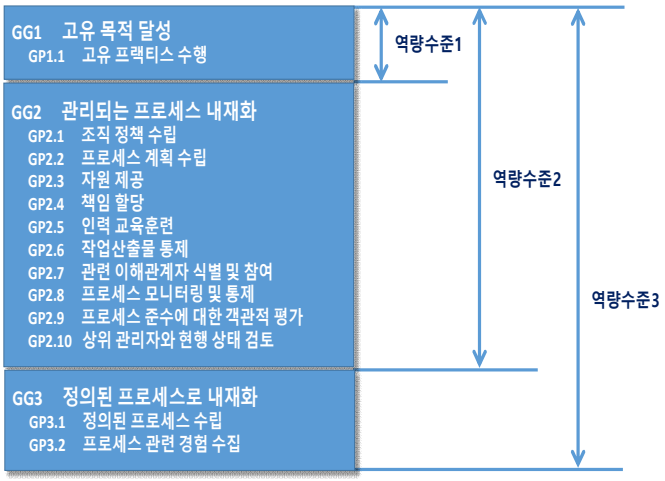


그림 4 공통 목적과 공통 프랙티스

CSMM 은 개별 프로세스가 달성한 역량수준에 따라 성숙도 단계를 결정하기에 CMMI 가 5 개의 성숙도 단계로 구조화 되어 있는 것과는 달리 4 개의 성숙도 단계로 구조화 되어 있으며, 각 성숙도 단계별 특성은 표 8 과 같다.

단 계	명 칭	특성(Characteristics)	
3	선도 (Pioneering)	• 모든 사이버보안 위협요소를 공격 및 대응 범위 내에서 관리 가능한 체계 구축 • 현재 요구되는 사이버보안 체계를 독자적(일부 모방 설계) 기술로 구현 가능 • 모든 프로세스가 역량수준 3에 도달	사이버보안 역량
2	확립 (Established)	• 전방위적 사이버침해에 대한 대응체계 구축 • 현재 요구되는 사이버보안 체계 중, 일부를 제외하고는 독자기술로 구현 가능 • 모든 프로세스가 역량수준 2이상에 도달	
1	수행 (Enabled)	• 국지적인 보안 침해에 대한 대응체계 구축 • 현재 요구되는 사이버보안 체계 중, 일부만 독자기술로 구현 가능 • 모든 프로세스가 역량수준 1이상에 도달	
0	초기 (Initial)	• 현재 요구되는 사이버보안 체계 대부분을 도입에 의존 • 하나 이상의 프로세스가 역량수준 0인 경우	사이버보안 위험

표 8 성숙도 단계별 특성

CSMM 의 성숙도 단계 결정은 개별 프로세스가 달성한 역량수준에 따른다. 예로써, 그림 5 와 같이 모든 프로세스가 역량수준 3 을 달성하면 성숙도 단계 3 으로 판정한다.

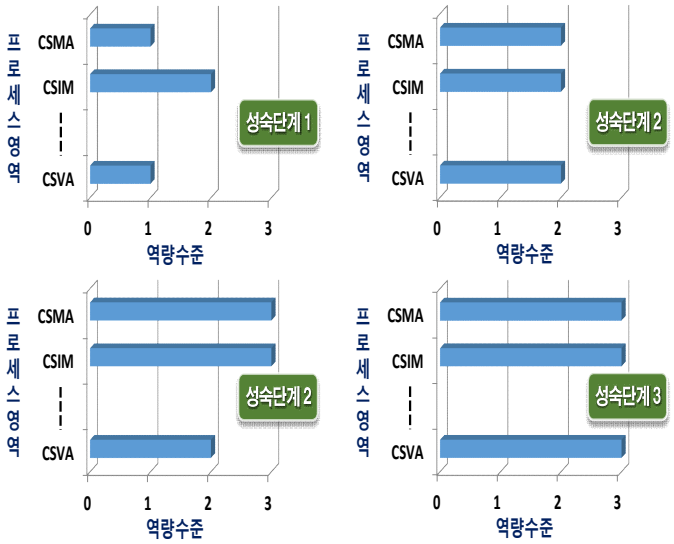


그림 5 성숙도 단계 결정 예

4. 결론

CSMM 은 4 개의 성숙도 단계와 4 개의 프로세스 범주로 구조화 되며, 사이버역량의 핵심인 공격, 방어, 기반시설을 아우르는 18 개의 프로세스 영역으로 구성된다. CSMM 은 사이버보안 활동을 이행, 관리 및 유지하기 위한 지침과 프랙티스를 제공하며, 사이버보안 프로세스의 측정 및 내재화(institutionalization)를 위해 활용할 수 있다. 조직은 조직의 사이버보안 역량을 강화시키기 위해 그림 6 과 같이 두 가지 접근 방법을 결합하여 사용할 수 있으며, 이를 통한 이점은 다음과 같다.

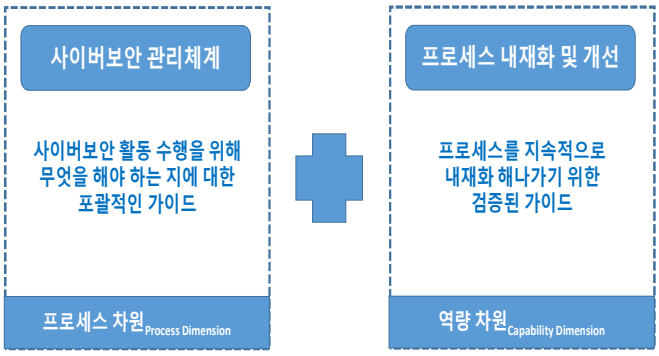


그림 6 CSMM 활용 방법

•사이버보안 역량의 효과적이고 일관적인 평가 및 벤치마크 제공
•현행 사이버보안 역량 파악 및 목표 역량 확보를 위한 방향 제시
•사이버보안 역량 향상을 위한 활동 및 작업 우선순위화

•사이버보안 역량 향상을 위해 조직 전반에 걸쳐 관련 지식, 모범사례, 참고자료 공유
•사이버보안 목표 및 임무 수행을 위한 활동간 조정 및 프로세스 수행 촉진
•사이버보안 역량 향상을 위해 유사한 활동 통합 및 중복된 작업 제거

금번 연구에서는 그 동안 국내외에서 이뤄진 사이버 역량 평가 및 성숙도 모델 관련 문헌연구를 통해 각각의 장단점을 분석하고 한국형 사이버보안 성숙도 모델인 CSMM 의 구조와 구성을 설계했다. 설계한 CSMM 의 구조와 구성 요소에 대해서는 추가 검토가 필요하며, 각 프로세스 영역별 프랙티스를 정의하기 위한 추가 연구가 필요하다.

참고문헌

[1] Son, Y. D., "Special Article: A Study on the New Order and Cyber-psychological Warfare," *Journal of KIISE,* pp. 22-29, January 2012.

[2] Son, Y. D. and Ko, S. H., "Cyber Warfare Extension Tendency and Response Capability," *Journal of Military Studies,* March 2015.

[3] Park, S. D. and Kim, I. J., "A Study on Tasks for the Legal Improvement for the Governance System in Cybersecurity," *Journal of Information and Security*, Vol. 13, No. 4, pp. 3-10, 2013.

[4] Technolytics, *Cyber Commander's eHandbook (Version 2.0)*, The Technolytics Institute, 2011.

[5] Clarke, R. A., *Cyber War: The Next Threat to National Security and What to Do About It*, HarperCollins Publishers, 2010.

[6] Kang, J. M., Hwang, H. U., Lee, J. M., Yun, Y. T., Bae, B. C., and Jung, S. Y., "A Study on National Cyber Capability Assessment Methodology," *Journal of KIISC*, Vol. 22, No. 5, 2012.

[7] DOE Program, *Cybersecurity Capability Maturity Model (Version 1.1)*, CMU-SEI, 2014.

[8] Global Cyber Security Capacity Centre, *Cyber Security Capability Maturity Model (Pilot Version)*, University of Oxford, 2014.

[9] CERT Program, *CERT Resilience Management Model (Version 1.1)*, CMU-SEI, 2011.

[10] KISA, *Information Security Management Systems Accreditation Guidelines*, Ministry of Science, ICT and Future Planning, 2013.

[11] Lee, M. J., *CMMI for Development Manual*, Hantee Media, 2013.

KCSE 2016 논문집 18권 1호

SPL 기반 무기체계 SW 통합개발환경 운용 개념 연구

Operational Concept of an Integrated Development Environment for SPL-based Weapon System Software Development

백옥현, 이태호, 박삼준

국방과학연구소 SW신뢰성기술실

순 서

연구 개요

연구 배경

공통 플랫폼을 적용해서 개발하면 어떨까?

기존에 적용하던 개발 방법으로 충분한가?

좀더 효과적이고 체계적인 재사용 방법은 없을까?

무기체계 SW 기능 요구가 점점 많아지고 복잡해지고 있다.

SW의 품질을 높이면서 단기간에 개발할 수 있는 방법은 없을까?

기존에 사용했던 아키텍처나 코드를 재사용할 수 있는 방법이 없을까?

무기체계 SW의 개발 특성에 맞도록 체계적으로 잘 정의된 플랫폼 개발 방법론 필요

방법론만으로는 그대로 적용하기 어렵다. 방법론에 따라 개발할 수 있도록 하는 자동화된 도구 필요

연구 개념

- **SPL(Software Product Line)**
  - 무기체계군(예 : 무인기, 유도무기, 전차, 전투기 등)이 가지는 공통의 기능들을 표준화된 플랫폼으로 확보하여,
  - 이를 기반으로 가변 기능과 추가적인 기능들을 구현 및 조립하여 무기체계 별 SW를 완성할 수 있도록 함으로써 전략적인 재사용을 지원하는 개발 패러다임

연구 범위

SPL 프레임워크
- SPL 표준 프로세스 연구
- 각 단계별 활동 정의
- 분석 활동별 기술적 접근방법
- 산출물 및 통합관리 절차

요소 기술
- 도메인분석 기법
- 플랫폼 아키텍처 설계 기술
- 플랫폼 구현 및 검증 기술
- 플랫폼 기반 SW 개발 기술

통합개발 환경
- 도메인분석 기능 설계
- 아키텍처 설계 기능 분석
- 컴포넌트 정의/구현 기능 분석
- 검증 지원 기능 분석
- 무기체계SW구현지원기능분석

과제 산출물

SPL 기반 무기체계 SW 개발 방법론
- 적용 가이드북
- 단계별 산출물 정의서

SPL 요소기술 확보

도메인 분석 | 아키텍처 설계 | 플랫폼 구현검증 | 플랫폼기반 SW 개발

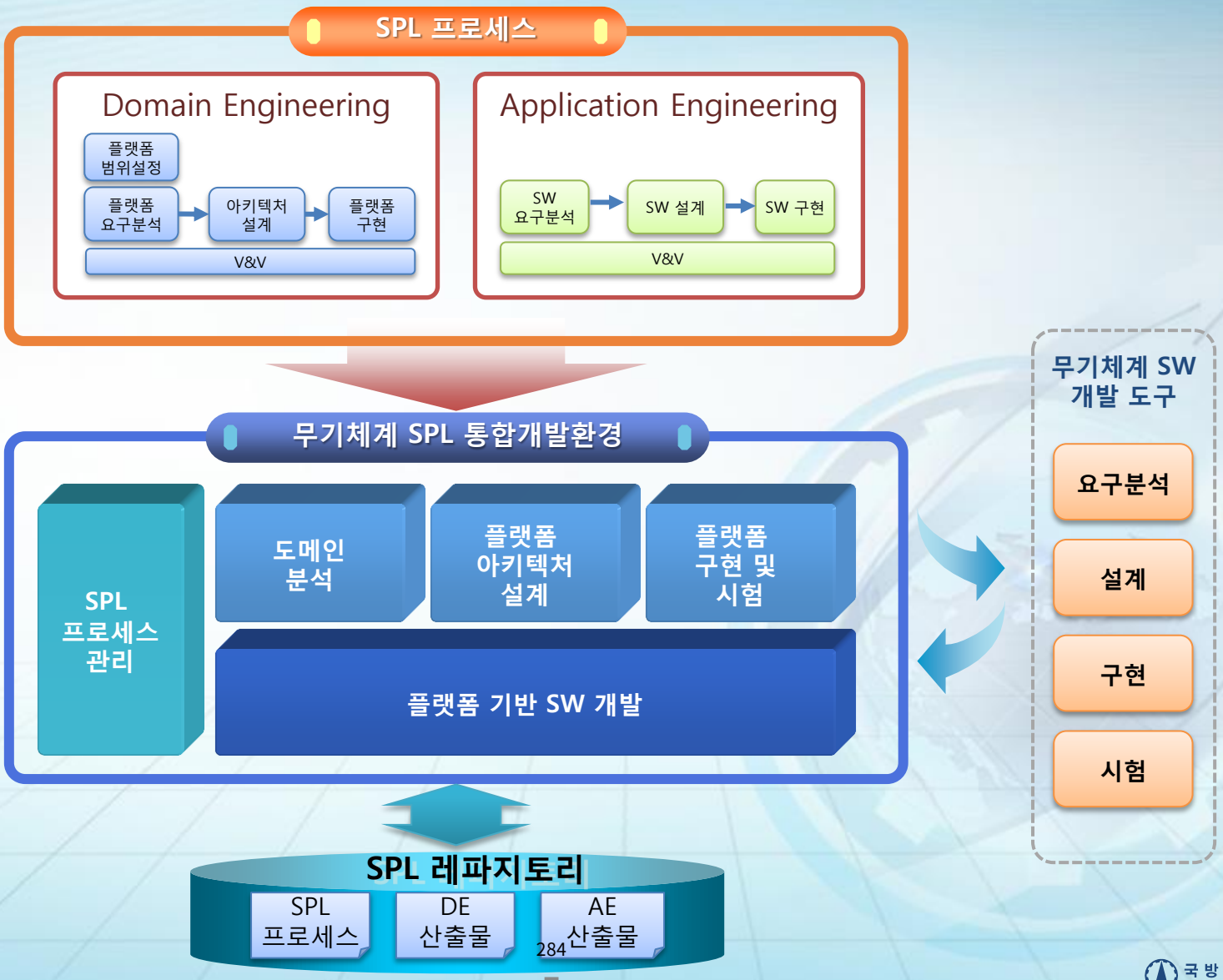
SPL 기반 무기체계 SW 통합개발환경

적용성 시험

SPL 시범구현

플랫폼 참조모델 | 플랫폼 참조구현

연구 범위



현 체계 분석

국방 SW 분류

구분	분류	소프트웨어
무기체계	지휘통제•통신무기체계	지휘통제체계 - 지상/해군/공군 전술 C4I - 합동지휘통제체계 - 군사정보통합처리체계
	감시•정찰무기체계	내장형 소프트웨어 - 감시•정찰 - 기동 - 함정 - 항공 - 화력 - 방호
	기동무기체계	
	함정무기체계	
	항공무기체계	
	화력무기체계	
	방호무기체계	
	기타 무기체계	국방M&S - 워게임모델(훈련용, 분석용, 획득용) - 전술훈련모의장비
전력지원체계	일반군수품	
	국방정보시스템	자원관리정보체계 M&S체계(무기체계로 기 분류된 것 제외) 기반체계
	교육훈련용 장비/물자	훈련용 장비 내장형 SW
	기타 일반시설	

전장관리 정보체계

286 : 국방정보시스템

9

무기체계 SW 분류 및 구성

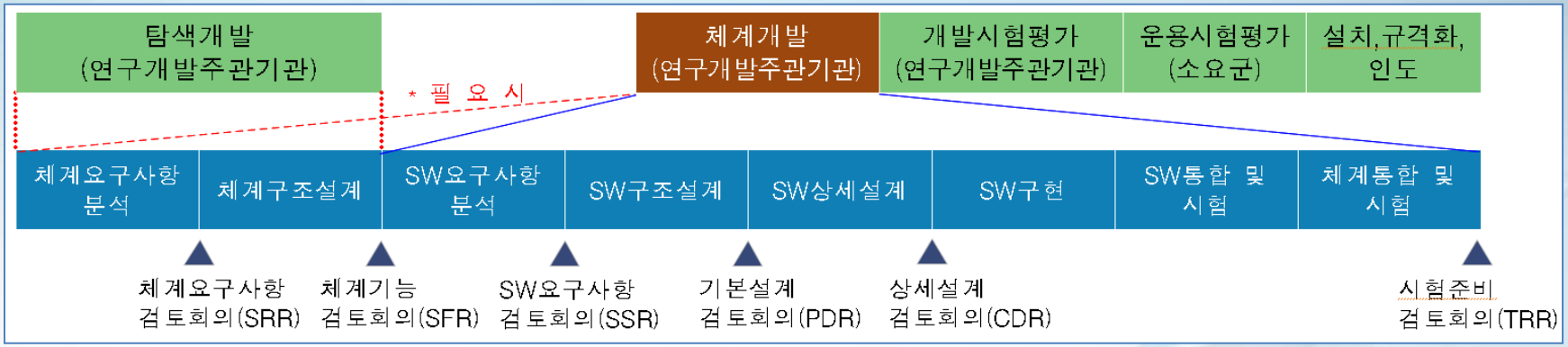
무기체계 SW 분류

무기체계 SW 구성

구분	설 명
응용SW	특정한 기능을 수행하도록 만들어진 응용 프로그램 (예) 자동제어, 지휘통제, 비행제어, 사격통제 등)
미들웨어	응용SW와 운용체계(OS) 사이에서 매개역할을 하는 중간 프로그램(예) DBMS, 통신미들웨어(RTI) 등)
운영체계(OS)	컴퓨터를 작동시키고 응용SW가 효율적으로 실행될 수 있는 환경을 제공하는 기본 프로그램 (예) Windows, Linux, VxWorks 등)

무기체계 SW 개발 프로세스

단일 SW 개발 프로세스



무기체계 SW 개발 조직 및 개발 형태

- **무기체계 SW 개발 조직**
  - 체계단과 기술부서로 구분
  - 체계단은 무기체계 개발, 체계 종합
  - 기술 분야별로 여러 체계의 구성품 개발

- **개발 형태**
  - 자체 개발
  - 외주 개발

무기체계 SW 재사용 사례

- **일부 기술분야 공통 프레임워크 운용 사용**
  - 공통기능과 대상 체계 고유 기능 식별
  - 재사용성, 변경용이성 증대 목표로 기존 소프트웨어 아키텍처 개선
  - 팀 내에서 SW 공용 프레임워크 구축
  - Asset 관리 : 아키텍처 전담 인원

- 일부 원시적 재사용(Copy & Paste)을 넘어선 주로 팀 내 필요에 의한 자발적 재사용
- 라이브러리, 시스템 내 공통 모듈 사용의 형태

- 프로세스 기반 체계적 재사용을 위한 프레임워크 필요

Cf. 전투체계 개방형 아키텍처(Open Architecture), AUTOSAR

SPL 기반 개발 도구

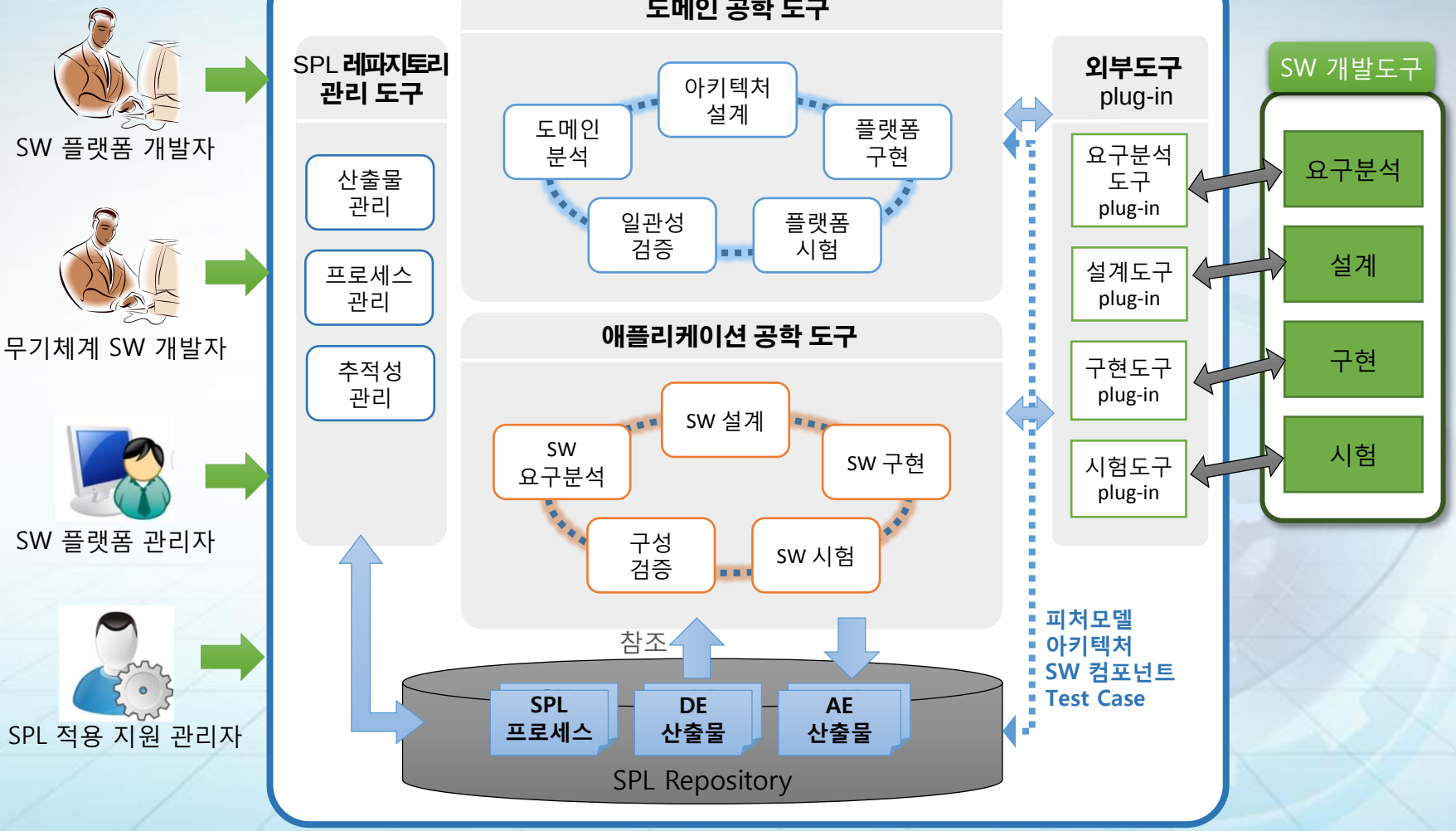
- **상용 도구**
  - Gears (BigLever Software, Inc(USA))
    - Feature profil과 shared asset을 이용한 자동화된 제품생산 지향
    - 외부 도구와 bridge를 통해 연동
  - Pure::variants(Pure System, Germany)
    - Feature 기반 제품 생성 지원
    - Doors, Rhapsody, Simulink 등의 도구와 연동
  - PLTP(Product Line Technical Probe), PLQL(Product Line Quick Lokk), SIMPLE(Structured Intuitive Model for Product Line Economics), OAR(Options Analysis for Reengineering) (CMU SEI)
    - SEI에서 개발한 PLP(Product Line Platform) 프레임워크 지원
  - VULCAN Workbench(한국)
    - FORM(Feature-Oriented Reuse Method) 지원

- **공개 도구**
  - FeatureIDE
    - Feature-Oriented Software Development (FOSD)방법 지원

- **Academic prototype**
  - Feature와 code의 mapping : FEAT, ConcernMapper, CIDE 등
  - Feature 와 구현물 visualization : FeatureCommander, FeatureMapper 등

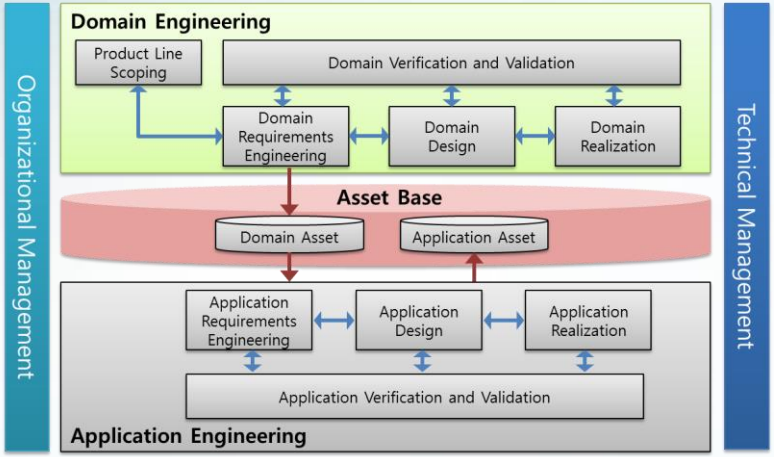
SPL 기반 무기체계 SW 통합개발환경 운용 개념

목표 체계 개념

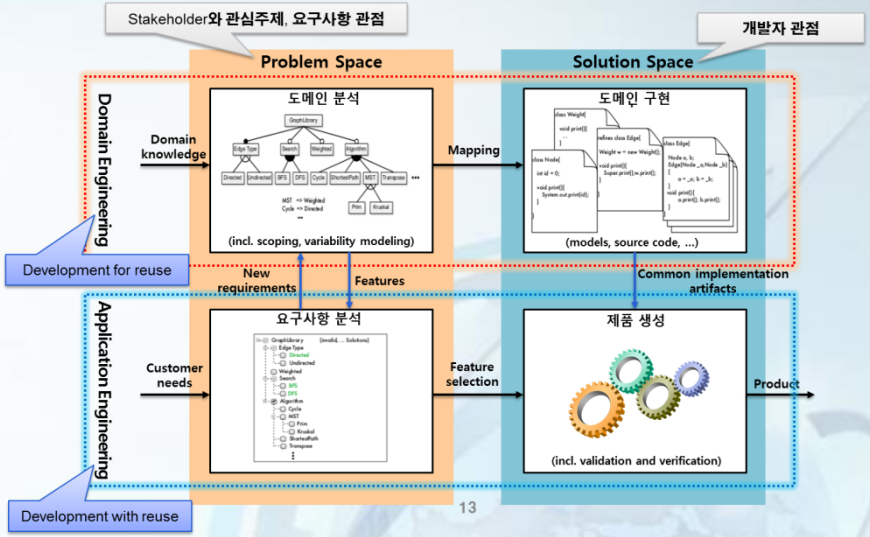


목표 체계 범위

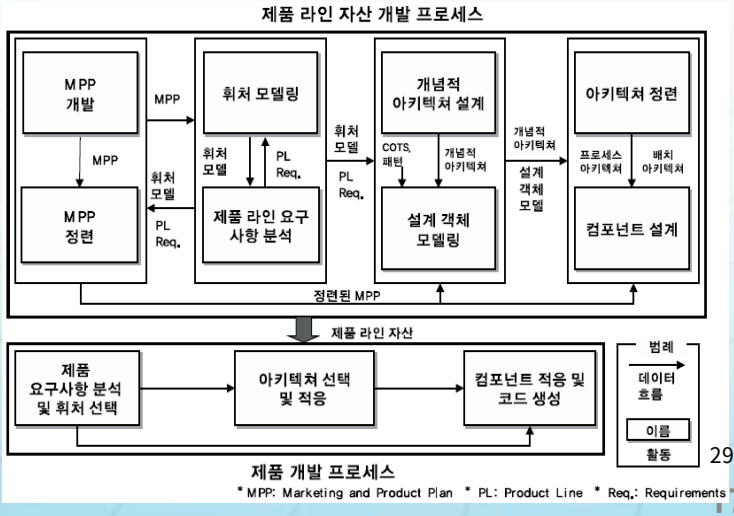
- **SPL 프로세스를 지원하는 SPL 도구 개발**
  - ISO/IEC 표준 및 민간 분야 프로세스를 참고하여 무기체계 SPL 프로세스 개발



ISO/IEC 26550



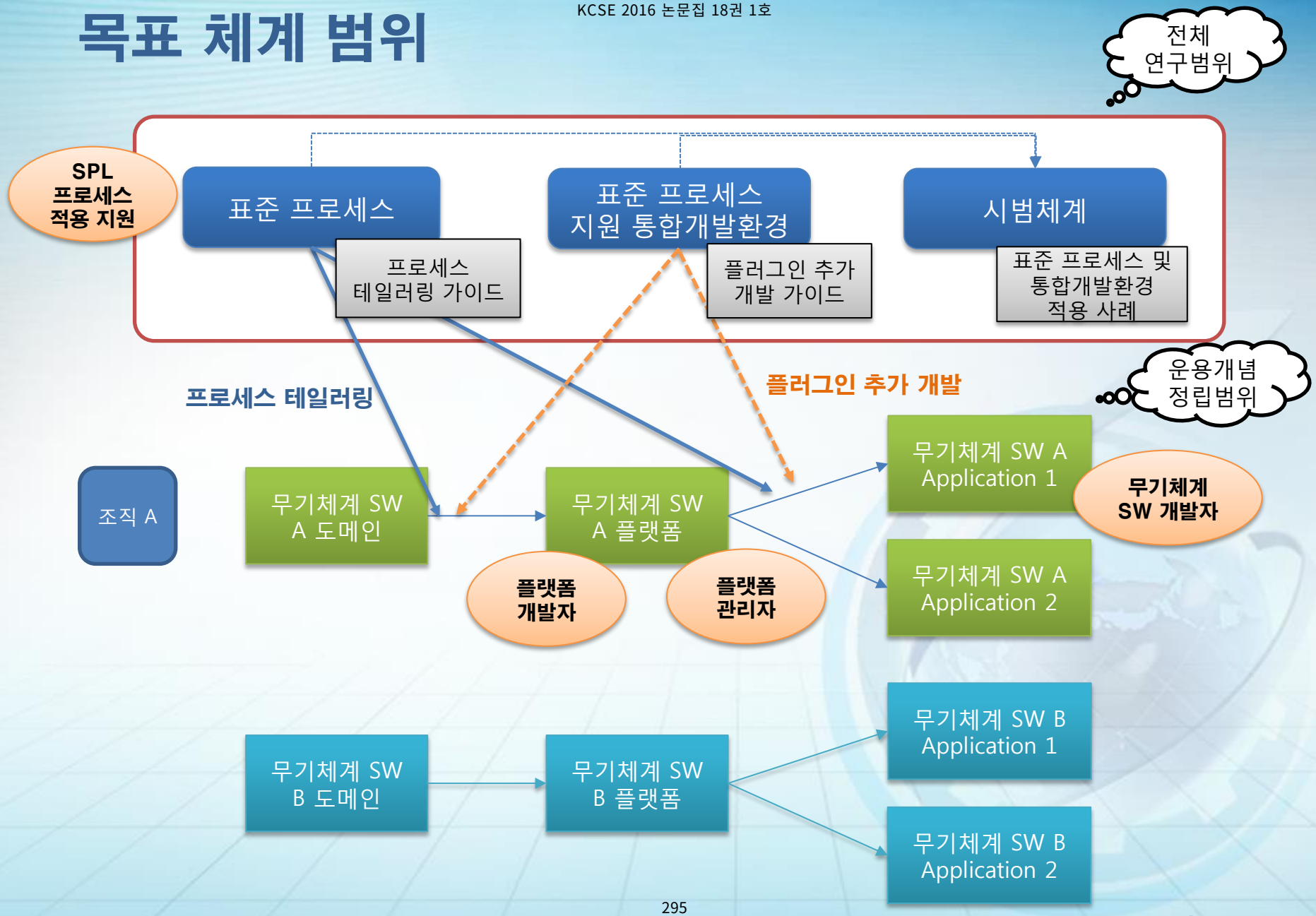
Feature-oriented product line process



FORM 제품 라인 개발 프로세스

294

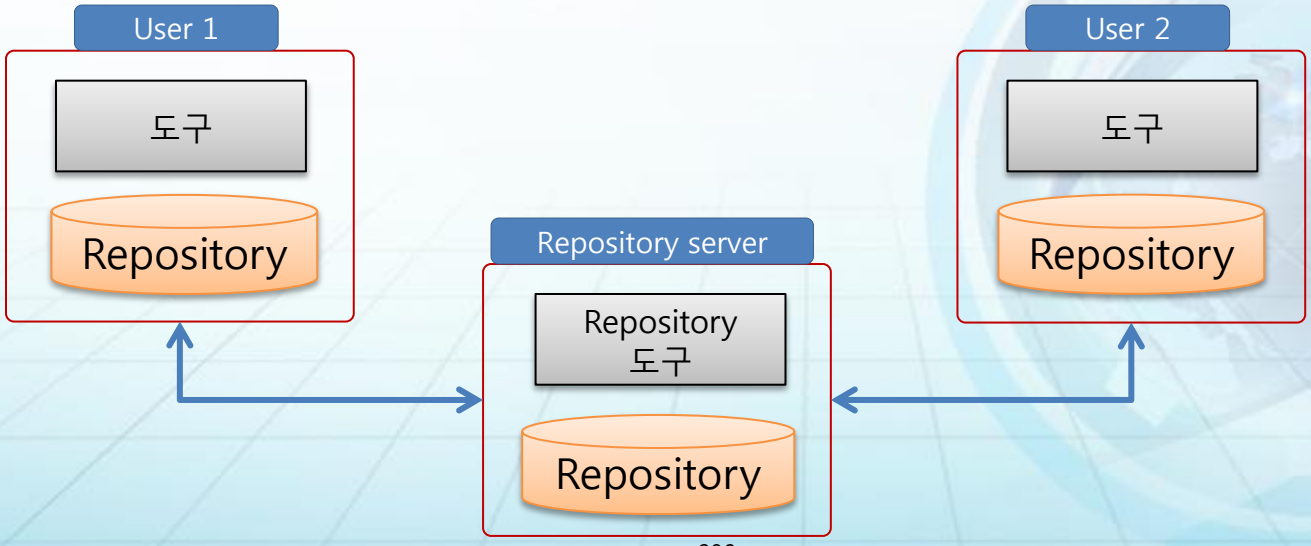
목표 체계 범위



운영방식, 제약사항 및 사용 조직

- **운영방식 및 제약사항**
  - 보안정책상 연구소 내 망 연결 제한
  - 공통 repository 사용 가능한 구조 필요
  - Repository 운용 측면
    - Repository server와 연결은 사용자 환경에 따라 online 또는 offline

- **사용 및 운용 조직**
  - 국과연 내 체계/기술개발 조직
  - 무기체계 SPL 구축 단위
    - 무기체계, 부체계, 구성품 등
    - Scoping 에 관한 가이드라인 개발 예정



사용자 분류 및 임무

- **SW 플랫폼 개발자(domain engineering 영역)**
  - 특정 도메인 내에서 재사용 가능한 플랫폼을 개발

  ※ 플랫폼 : 재사용 가능한 산출물의 모음(요구사항 모델, 아키텍처 모델, 소프트웨어 컴포넌트, 시험 계획, 시험 설계 등)

- **무기체계 SW 개발자(application engineering 영역)**
  - 플랫폼을 이용하여 무기체계 SW 애플리케이션 개발

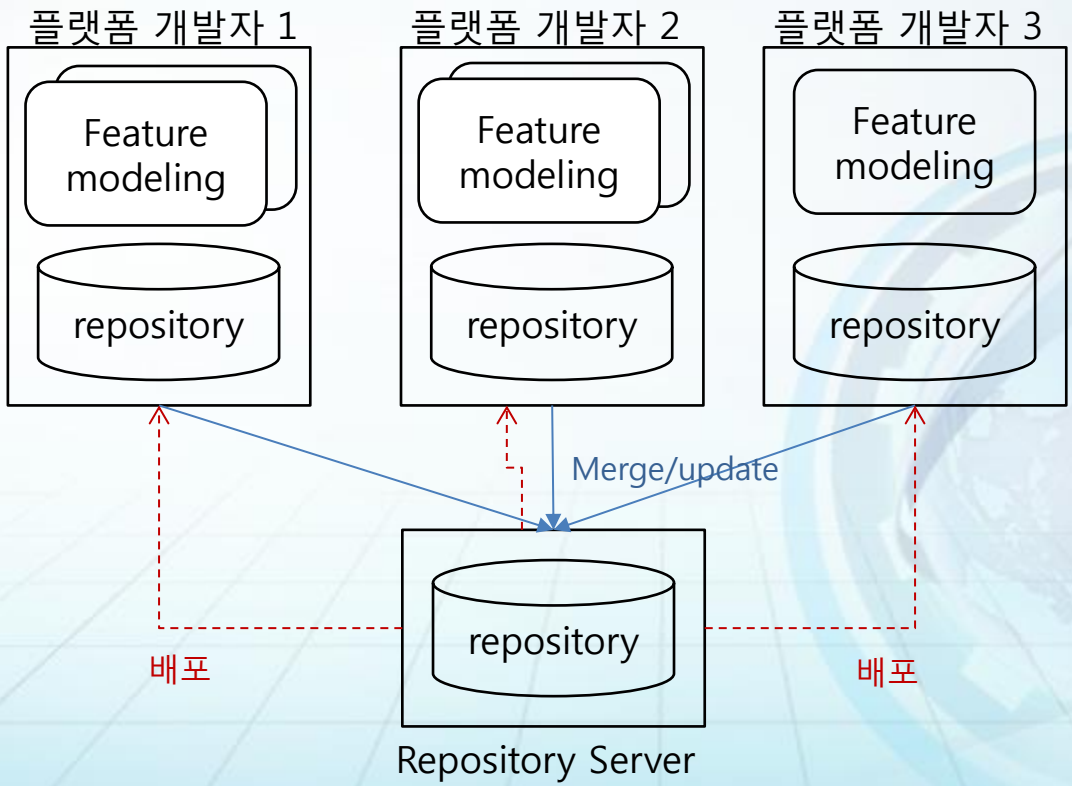
- **SW 플랫폼 관리자(개별 SPL 관리자)**
  - 피처 변경관리
  - Asset 변경관리
  - 플랫폼 기반 재사용 모니터링
  - 플랫폼 evolution 관리

- **SPL 적용 지원 관리자**
  - SPL 통합개발환경 운용시의 관리자 권한을 소유하여, 통합개발환경에 대한 설정을 변경할 수 있는 권한을 가진 사람
  - 연계 도구 설정
  - 자료의 삭제 혹은 복구
  - 사용자 권한 관리

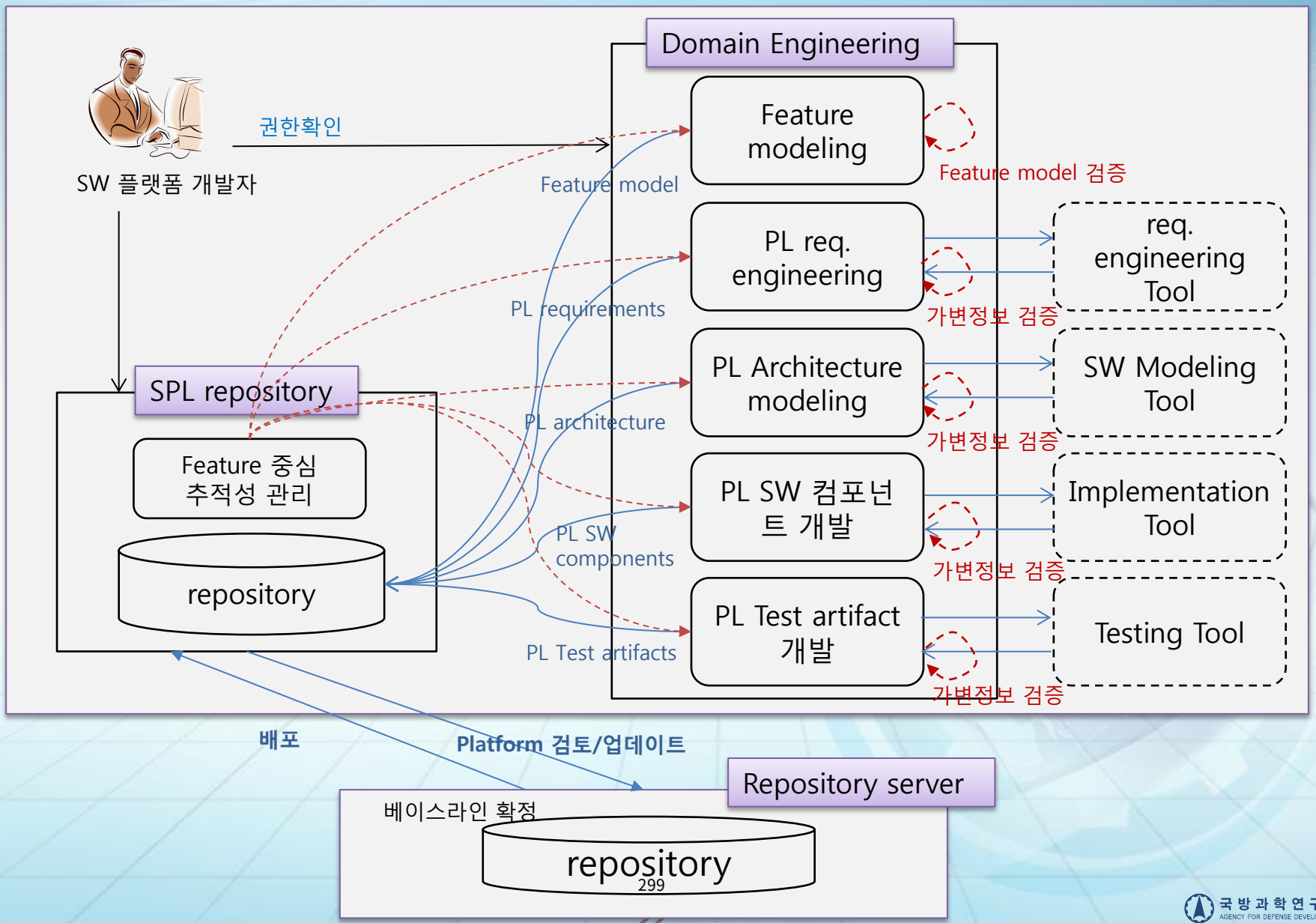
목표 체계 운영 시나리오

- 개별 플랫폼 개발자들이 작업
- 개별 작업한 내용을 검토한 후 (주기적) 업데이트
- 베이스라인 확정 후 배포

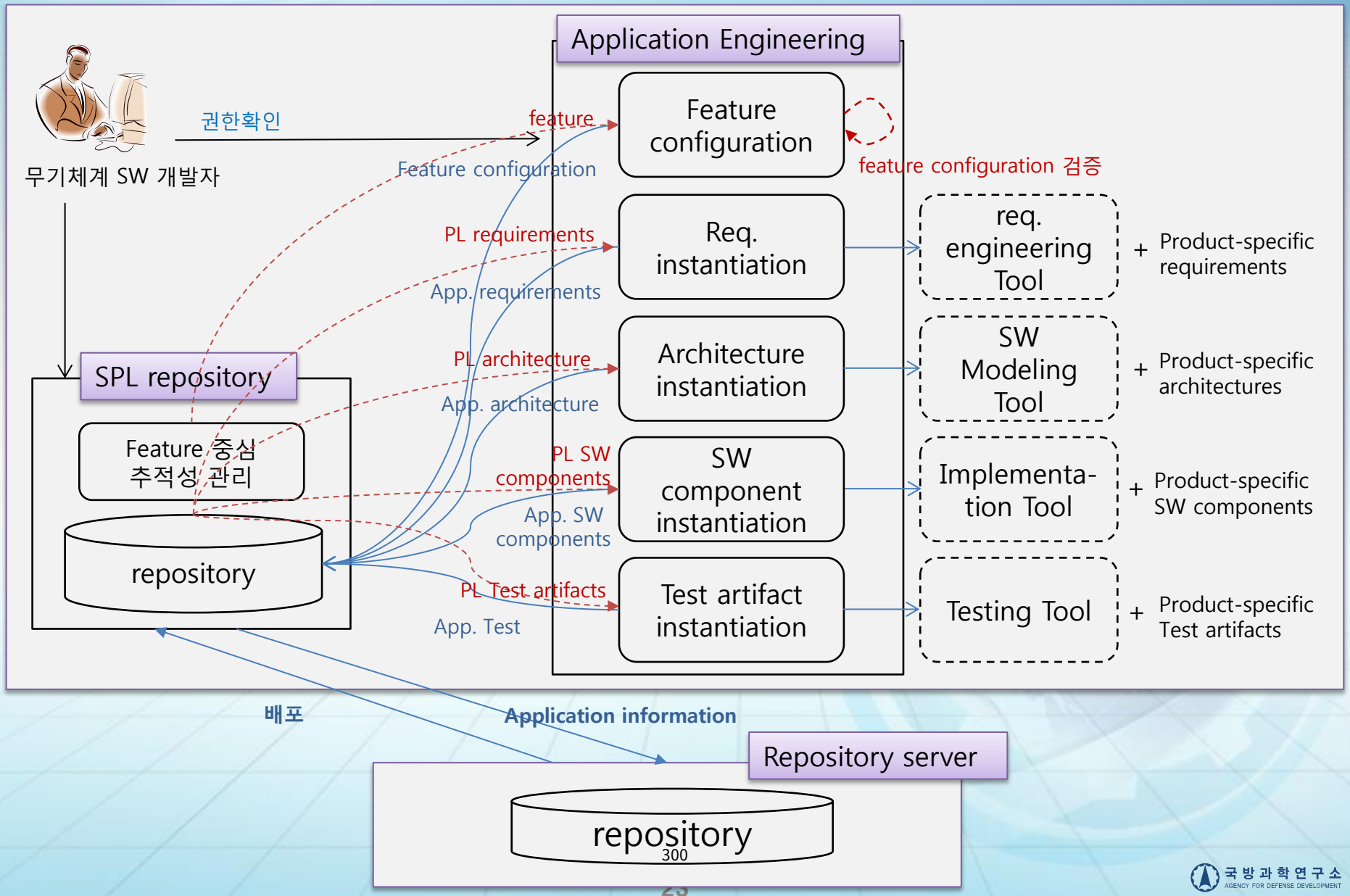
Iteration
SPL 프로세스와 유기적 관계



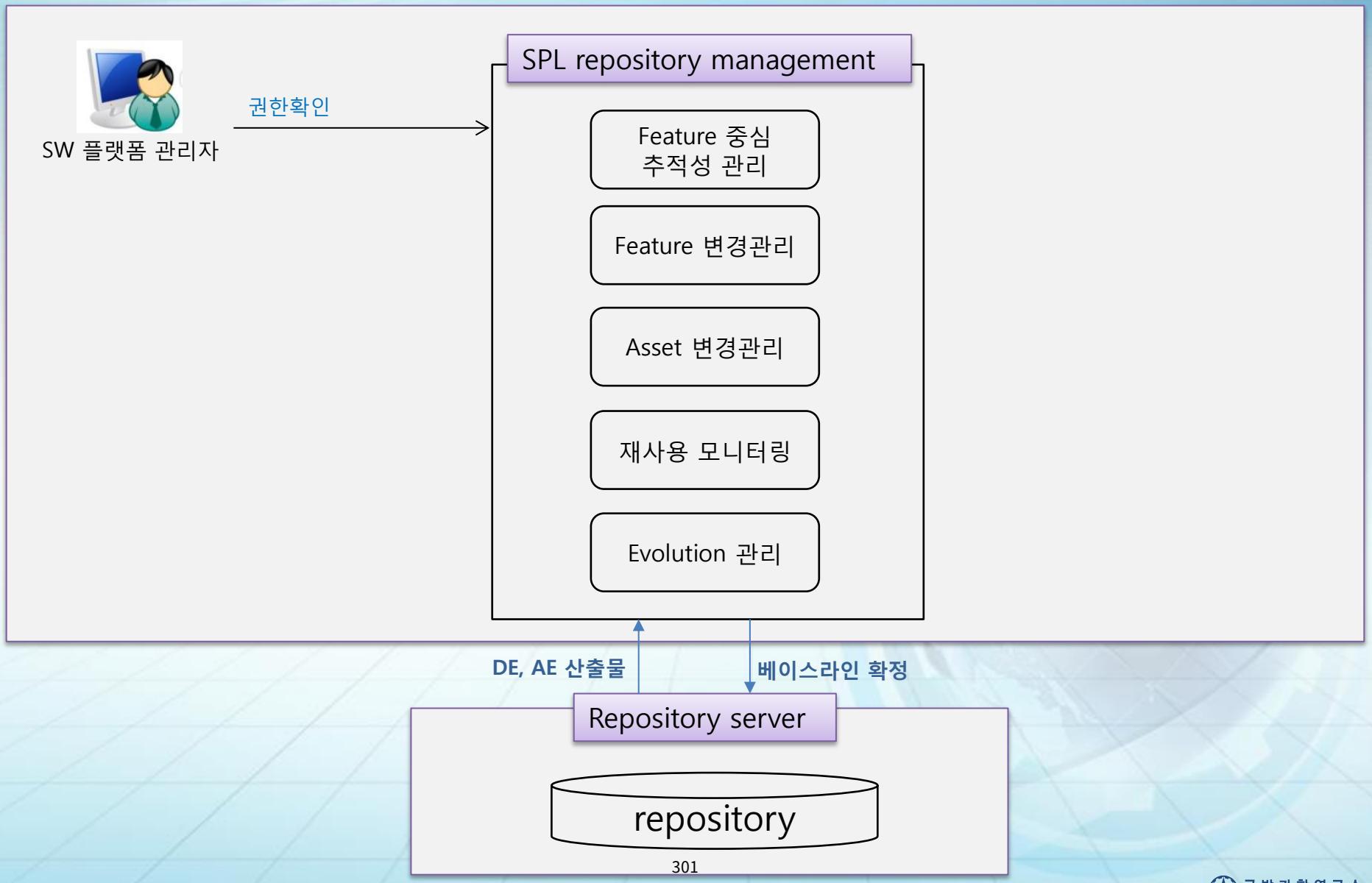
사용자 별 운영 시나리오 : SW 플랫폼 개발자



사용자 별 운영 시나리오 : 무기체계 SW 개발자



사용자 별 운영 시나리오(SW 플랫폼 관리자)



결론 및 향후 연구

- **무기체계 SW 개발에 Software Product Line Engineering Paradigm 적용**
  - 무기체계의 복잡화, 첨단화, 무인화, 자율화, 네트워크화 되며 무기체계 제반 기능의 상당 부분을 SW로 구현
  - 무기체계 SW 비중 증가 추세 지속
  - 효과적인 SW 개발방법 적용을 통한 품질향상, 기간단축, 비용절감 필요

- **SPL 기반의 무기체계 SW 통합개발환경 운용 개념 연구**
  - 무기체계 SPL 프로세스를 지원하는 도구
  - 현 무기체계 SW 개발 현황 분석하여 개발방안 수립
  - 주 사용자 도출 및 사용자 별 개념 운용 시나리오 도출
  - SPL 고유 기능 직접 개발하되, 일반 SW 개발 기능은 SE 도구 plug-in

- **향후 연구**
  - 무기체계 SPL 프로세스 정립
  - 운용개념 정립에 따른 통합개발환경 개발

References

- ISO/IEC 26550 Software and systems engineering – Reference model for product line engineering and management, 2013.
- Klaus Pohl, Gunter Bockle, Frank van der Linden, Software Product Line Engineering : Foundations, Principles, and Techniques, Springer, 2005.
- Sven Apel, Don Batory, Christian Kastner, Gunter Saake, Feature-Oriented Software Product Lines : concepts and implementation, Springer-Verlag Berlin Heidelberg 2013.
- Kyo C. Kang et al., Feature-Oriented Domain Analysis(FODA) Feasibility Study, CMU/SEI-90-TR-21, 1990.
- 월간 국방과 기술, Vol.441, 2015.11
- 국방과학연구소, 무기체계 SW 개발 관리 아카데미 교육자료, 2015.

분산시스템에서의 공유자원 가용성을 고려한 실시간 제어방법 연구

장부철 정우진 송대기 신진범

국방과학연구소 제 1 기술연구본부
대전 유성우체국 사서함 35-13 호
bcjang@add.re.kr

요약: 본 연구에서는 다수의 무장연동장치와 한 개의 발사대제어장치와의 연동에 있어서 발사대제어장치에 제어명령을 송신하는 오직 한 개의 무장연동장치를 선정하는 마스터 선출 기술에 관한 내용으로써, 기존 방식의 문제점인 프로세스 자체는 살아있지만 공유자원을 더 이상 사용할 수 없는 경우에 대한 정보공유 수단이 없다는 점과 마스터 재선정 과정에 따른 제어명령의 실시간 처리가 이루어지지 않는 문제점을 해결하여 발사통제시스템의 신뢰성을 향상시키는 방법을 제안한다.

핵심어: 마스터 선출기법, 발사통제시스템, 분산시스템, 무장연동장치

1. 연구배경

함정 발사통제시스템의 경우 시스템의 생존성 향상을 위해 분산시스템 형태로 구성하는 방식이 일반적이다. 이때, 여러 개의 분산시스템이 하나의 자원을 공유하는 경우 이는 시스템 전체의 생존성 및 실시간성에 영향을 주는 요소가 된다. 이를 해결하기 위한 기존의 분산시스템에서 사용하던 마스터 선출 방식으로는 Bully 알고리즘[1]과 Ring 알고리즘[2] 등이 있다. 이는 분산된 프로세스 중 마스터 프로세스가 죽은 것을 발견한 프로세스가 자신뿐만 아니라 다른 프로세스들에게 마스터 선출과정을 시작시키는 방식이다. 즉, Bully 알고리즘에선 그림 1 과 같이 기존에 마스터로 선출되어 있던 4 번 프로세스가 죽은 것을 1 번 프로세스가 발견한 경우 1 번 프로세스는 자신보다 번호가 큰(상위 ID 프로세스) 2, 3, 4 번 프로세스에게 마스터 선출명령(Election)을 보내고 이에 대한 답변(Answer)을 받은 경우 자신보다 상위 프로세스가 존재하므로 1 번 프로세스는 역할을 종료한다. 위의 과정을 2, 3, 4 번 프로세스에서 반복적으로 수행하여 자신보다 더 높은 프로세스로부터의 응답이 없는 프로세스가 마스터가 되며 이를 다른 프로세스들에게 알리는 방식이다.

마스터 선출시간을 단축하기 위한 고속 불리 알고리즘[3]의 경우는 마스터 프로세스가 비정상임을 발견한 프로세스가 선출명령에 대한 답변을 받은 프로세스 중 최상위 ID 프로세스에게 새로운 마스터임을 알리는 메시지(Nomination)를 송신하고 마스터 후보 프로세스는 이에 대한 수락(Acceptance) 및 코디네이션(Coordination) 메시지를 송신하는 것으로 선출과정을 종료하게 된다.

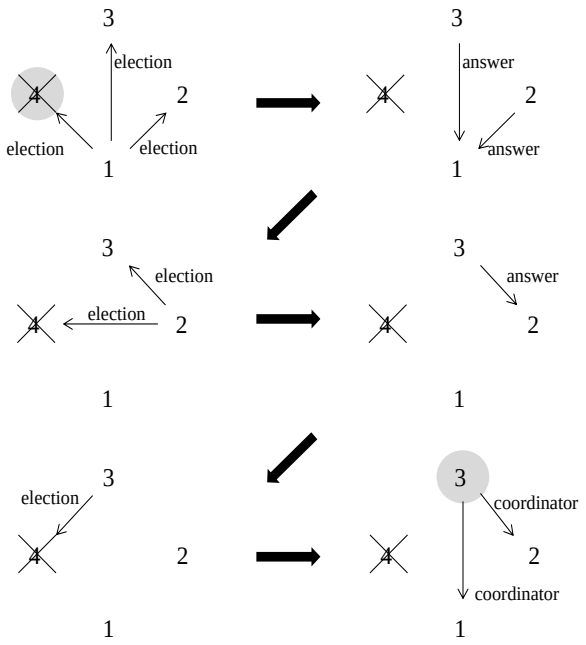


그림 1 불리 알고리즘

Improved 불리 알고리즘[4]은 차기 마스터 프로세스가 누구인지를 모든 프로세스가 공유하도록 하기 위해 주기적으로 Heartbeat(HB) 메시지를 교환하며 마스터 프로세스로가 비정상임을 발견한 프로세스가 차기 마스터에게 마스터가 비정상임을 알리고 차기 마스터는 불리 알고리즘을 수행하는 방식이다. 고속 불리 알고리즘과의 차이점은 차기 마스터가 누구인지를 기존 마스터가 다른 프로세스들에게 미리 알려주기 위해 주기적인 HB 메시지를 사용한다는 점이다. 위의 두가지 방법 모두 일반 불리 알고리즘과 마

찬가지로 공유자원에 대한 제어가 불가능한 경우에 대한 정보공유 수단이 없다는 문제점 및 마스터가 정상인지 여부를 사전에 알 수 없으므로 마스터 비정상 시 제어명령의 실시간 처리가 이루어지지 않는 단점이 있다. 그리고, Nomination 메시지 및 이에 대한 응답 메시지인 Acceptance 메시지가 추가로 필요하여 구현의 복잡성이 증가하게 된다. 즉, 기존 방식을 단일노드 제어 및 실시간성을 필요로하는 발사통제시스템에 적용하는 경우 다음과 같은 문제점이 있다.

첫 번째, 마스터 프로세스가 스스로 재선출 명령을 전송하지 못함으로써 발생되는 문제로 마스터만이 공유자원에 대한 엑세스가 가능하도록 제한된 시스템의 경우 마스터 프로세스 자체는 살아있지만 공유자원을 더 이상 사용할 수 없는 경우에 대한 정보공유 수단이 없다는 문제점이 있다. 즉, 기존 알고리즘의 경우 마스터 프로세스에게 서비스 요청이 있을 때까진 마스터 프로세스의 정상여부를 확인할 수 없다.

두 번째, 마스터 재선출에 따른 실시간 제어의 어려움으로서 어떤 프로세스가 마스터 프로세스에게 보낸 제어명령이 처리되기 전에 마스터 프로세스가 비가용 상태가 된 경우 이를 해결하기 위해서는 제어명령을 보낸 프로세스가 명령을 기억하고 있다가 재선출 과정을 통해 새로운 마스터 프로세스가 선출된 이후 이전 마스터에게 전송한 명령을 새로운 마스터 프로세스에게 다시 전송해야하므로 제어명령의 전송에 시간지연이 발생한다. 즉, 마스터가 살았는지 죽었는지 여부를 사전에 알 수 없으므로 마스터 재선출 과정에 따른 제어명령의 실시간 처리가 이루어지지 않는 단점이 있다.

2. 분산시스템 구성

본 알고리즘을 적용한 발사통제시스템의 구성은 그림 2 와 같다. 본 시스템 구성은 본 논문에서 제안한 마스터 선출 기법을 도출하게 된 원인으로 발사통제콘솔의 제어 명령이 발사대 제어장치로 실시간성과 신뢰성을 만족하며 전송되도록 하고자 하였다.

공유자원에 해당하는 발사대 해치를 제어하기 위한 명령의 흐름은 다음과 같다.

① 기존 불리 알고리즘를 이용하여 4 대의 무장 연동장치(Weapon Interface Unit)중 하나가 마스터로 선출되고 무장 연동장치간의 통신에 의해 필요시 마스터의 가용여부에 따른 재선출 과정을 수행한다.
② 발사통제콘솔(Fire Control Console)은 마스터 무장 연동장치로 발사대 제어명령(Launcher Control Command)을 송신한다.
③ 마스터 무장 연동장치는 발사대 제어장치(Launcher Control Box)로 제어명령을 전송하고 발사대 제어가 이루어진다.
④ 발사대 제어장치로부터 발사대 제어상태(Launcher Status) 데이터를 수신받는다.
⑤ 발사통제콘솔은 무장 연동장치를 통해 발사대 제어상태 데이터를 수신받는다.

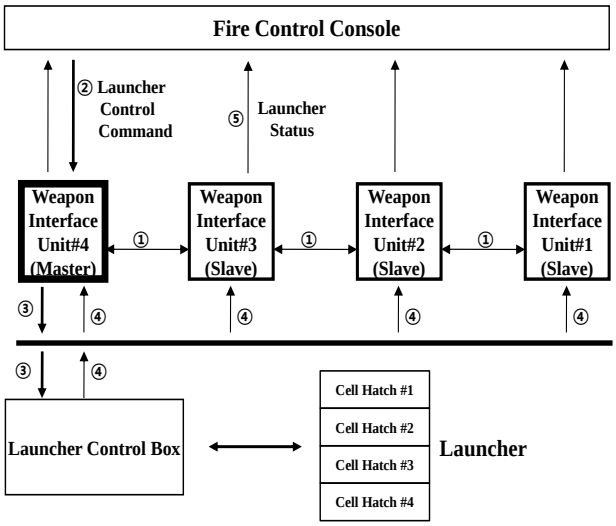


그림 2 발사통제시스템

3. 제안 개념

분산시스템의 공유자원 이용을 위한 마스터 선출 방식에 있어 위에서 언급한 두 가지 문제점에 대한 해결책으로 다음과 같은 기법을 제안한다.

3.1 공유자원 가용성 확인 방법

마스터 프로세스가 공유자원과의 통신 비정상으로 더 이상 공유자원을 사용할 수 없게 된 경우 다른 프로세스에게 마스터 선출 과정을 수행하도록 알려줌으로써 기존 마스터 선출 방식의 문제점인 프로세스 자체는 살아있지만 공유자원을 더 이상 사용할 수 없는 경우에 대한 정보공유 수단 부재를 해결하는 그림 3 와 같은 방식을 제안한다.

각 프로세스는 초기 부팅 시 불리 알고리즘을 수행하여 마스터 프로세스를 선출하고 마스터 프로세스로 선출된 프로세스는 T1 주기로 공유자원 가용성을 확인한다. 마스터 프로세스가 공유자원을 사용하지 못하는 비정상적인 상황이 발생한 경우 다른 프로세스에게 T1 주기로 송신하는 “I’m Master” 메시지를 보내지 않음으로써 다른 프로세스가 마스터 선출과정을 수행하도록 한다. 이때, 마스터 선출을 위한 이벤트성 메시지인 선출 메시지(Election)를 송신하는 방식보다 제안하는 방식과 같이 T1 주기로 송신하는 주기 메시지를 보내지 않는 방식을 사용함

으로써 기존 불리 알고리즘의 기본 내용을 유지하면서 공유자원 비가용 정보를 알려줄 수 있도록 하였다.

즉, 일반 불리 알고리즘의 경우 상위 프로세스들에게만 마스터 선출 메시지(Election)를 보낼 수 있기 때문에 최상위 ID 프로세스가 마스터인 경우 기존 방식으로는 재선출 과정을 수행하지 못한다.

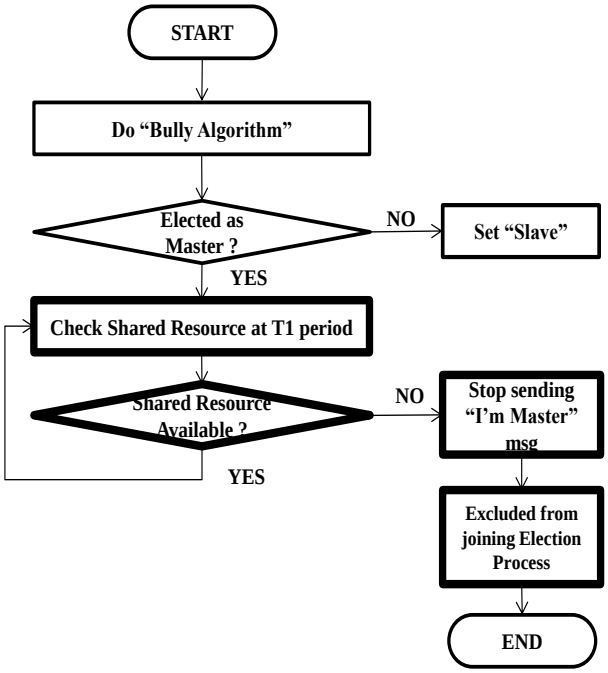


그림 3 공유자원 가용성 검사 기법

3.2 실시간 재선출 방법

제어명령의 실시간 처리를 위해 제어명령 수행 시점이 아닌 사전에 마스터를 재선출함으로써 명령의 실시간 처리 능력을 향상시키는 그림 4 와 같은 방식을 제안한다. 마스터 프로세스는 모든 프로세스들에게 주기적으로 자신이 살아있음을 알리는 메시지를 T1 주기로 전송하여 마스터가 누구이며 정상 여부를 알려준다. 마스터 프로세스가 아닌 다른 프로세스는 “I’m Master” 메시지를 T1*3 시간 동안 수신하지 못하면 마스터 프로세스가 비정상인 것으로 판단하여 마스터 재선출 과정을 수행한다. 제안한 기법을 통해 제어명령 수행 시 마스터 재선출에 따른 시간 지연을 사전에 방지하고, 발사통제콘솔은 송신한 제어 명령을 기억하고 있다가 무장 연동장치의 마스터 재선출 완료 후 명령을 다시 전송할 필요가 없으므로 별도의 명령 기억 및 재전송 메커니즘이 필요 없어 전체 시스템 소프트웨어 구조가 단순해지는 장점을 가진다.

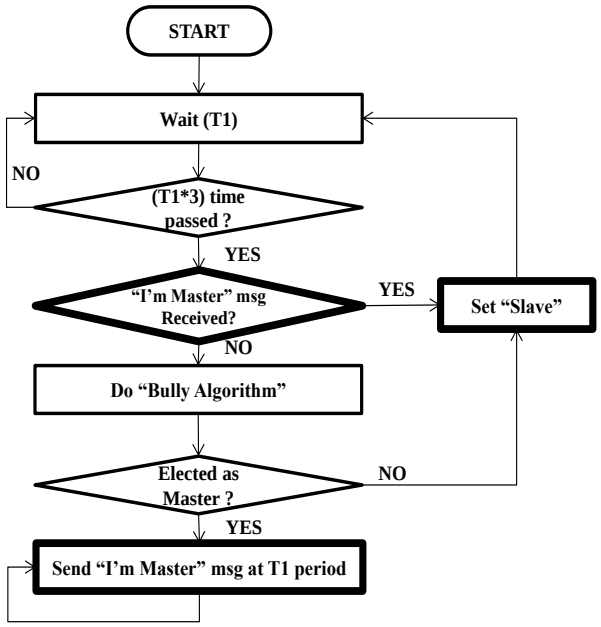


그림 4 실시간 선출 기법

4. 결 론

본 논문은 분산시스템의 마스터 선출 방법에 있어 공유자원 가용여부 및 마스터 정상 여부를 실시간으로 확인함으로써 발사통제장비와 같은 실시간 분산 시스템에 기존 검증된 마스터 선출 알고리즘을 적용할 수 있는 기법을 기술하였다. 본 기법의 적용이 반드시 필요한 실시간 발사통제시스템 구조를 보이고 제안한 기법을 통해 이를 해결할 수 있는 방안을 제시하였다. 본 기법을 적용한 발사통제시스템 소프트웨어의 설계 및 구현 내용은 추후 기술할 예정이다.

참고문헌

[1] H. Garcia-Molina, “Elections in a distributed computing system”, IEEE Trans. On Computers, Vol. 31, pp. 48 ~ 59, Jan 1982

[2] G. Le Lann, “Distributed system – Toward a formal approach” , Proc. Of the IFIP Congress 77, 1977

[3] H. Choi, “An Efficient Algorithm for Electing a Coordinator Process in Distributed Systems”, The Korean Institute of Information Scientists and Engineers, Vol. 25, pp. 926 ~ 936, 1998

[4] A.Arghavani, E.Ahmadi, AT.Haghighat, “Improved Bully Election Algorithm in Distributed Systems” , Proceedings of the 5th International Conference on IT & Multimedia, Malaysia, 2011

정형 기법을 이용한 NFV Policy 들의 일치성 검증

구근회

고려대학교 컴퓨터전파통신공학과
서울 성북구 안암로 145
khkoo@formal.korea.ac.kr

강미영, 최진영

고품질융합소프트웨어센터
서울 성북구 안암로 145
{mykang, choi}@formal.korea.ac.kr

이승익

ETRI
대전광역시 유성구 가정로 218
seungiklee@etri.re.kr

요약: NFV 는 통신사업자들이 사용하고 있는 네트워크 장비 내의 여러 기능들을 분리시켜 소프트웨어적으로 제어 및 관리가 가능하도록 가상화시키는 기술이다. 그리고 네트워크 서비스 장비의 기능을 가상화하여 하드웨어와 소프트웨어를 분리하므로 NFV 에서는 자원관리가 중요하다. NFV 의 자원관리는 Network Service 측면에서 관리하는 policy 와 NFVI policy 가 서로 다른 요구사항에 의해 충돌이 발생할 수 있다. 그러므로 policy 간에 충돌이 일어나기 이전에 이를 확인할 수 있는 연구가 반드시 필요하다.

본 연구에서는 NS policy 와 NFVI policy 를 모델링하고 SMT solver 인 Z3 를 이용하여 policy 간의 충돌(conflict)를 검증하는 방법을 제시한다.

핵심어: NFV, Z3, 정형 모델링, 정형 검증

1. 연구배경

Network Function Virtualization(NFV)[1]는 가상 하드웨어 추상화를 사용하여 하드웨어로부터 네트워크 기능들을 분리시키는 원칙이다. 본 논문에서는 ETSI NFV ISG(Industry Specification Group)표준을 중심으로 NS(Network Service) policy 와 NFVI(Network Functions Virtualization Infrastructure) policy 를 정의하여 모델링하였다. NFV 환경에서 NS policy 는 end-to-end QoS 을 위한 Network Service 를 정의한다. 그리고 NFVI policy 는 자원 스케줄링에 대한 NFVI 을 정의한다[그림 1]. NS policy 와 NFVI policy 는 서로 다른 요구사항에 의해 충돌이 발생할 수 있다. 각 policy 를 모델링하여 충돌이 발생하기 이전에 검증하여 올바른 NFV 환경을 유지할 수 있도록 보장할 수 있다. 모델링의 정형 검증에는 SMT 를 기반으로 한 도구인 Z3 을 이용하였다.

이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(B0101-15-233, 스마트 네트워킹 핵심 기술 개발). 그리고 "본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성 지원사업의 연구결과로 수행되었음"(IITP-2015-H8501-15-1012).

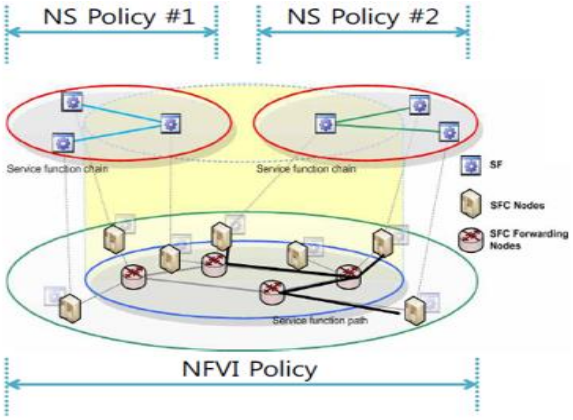


그림 1. NS policy 와 NFVI policy

2. 관련연구

2.1 NFV

NFV 기술은 통신사업자들이 사용하고 있는 네트워크 장비 내의 여러 기능들을 분리시켜 소프트웨어적으로 제어 및 관리가 가능하도록 가상화시키는 기술이다. NFV를 구현하는 방식은 다양하지만 일반적인 방법으로는 네트워크 장비내의 기능들을 데이터 센터 내에 위치하는 대용량 서버, 대용량 저장장치, 그리고 대용량 스위치로 분리하고, 표준적인 방법으로 액세스가 가능하며, 이 장비들에 소프트웨어적으로 개발된 네트워크 기능들이 자동적으로 설치되고, 동작하며, 이동할 수 있도록 하는 네트워크 구조를 만드는 것이다. NFV 기술을 이용하면 네트워크 장비 비용과 전력손실 절감으로 인한 CAPEX(Capital Expenditure) 및 OPEX(Operating Expense) 감소, 새로운 네트워크 서비스의 시장 투입에 필요한 시간단축과 투자비용 회수 증대, 유연한 서비스 진화성과 스케일 관리 용이, 가상 기기 및 순수 소프트웨어 참여 시장 개방, 그리고 새로운 혁신적 서비스개발 기회 증대 등의 효과를 기대할 수 있다[2]

2.2 Z3

Satisfiability modulo theory(SMT)[3]는 Boolean satisfiability(SAT)에 equality reasoning, arithmetic, fixed-sized bit-vector, array, quantifier 외 기타 first-order 이론들을 적용해 일반화한 것이다. SMT solver 는 이러한 이론들을 통해 만들어진 식의 충족 가능성(satisfiability)을 결정하는 도구이다. SMT solver 로는 extended static checking, predicate abstraction, test case generation, 그리고 bounded model checking over infinite domains 등의 작업을 수행할 수 있다.

Z3[4]은 Microsoft research 에서 제작한 SMT solver 이다. 이는 소프트웨어 검증 및 소프트웨어 분석에 대한 문제를 풀 수 있다. 따라서, 이는 여러 가지 이론들을 지원한다. 현재 MIT 로 라이센스가 이전되어 누구나 상업적 개발가능하여, Spec#/Boogie, Pex, HAVOC, Vigi-lante, VCC, Yogi 등 상업적으로 성공한 도구들이 발표되고 있다. 또한, 마이크로소프트사의 SLAM/SDV 등 프로젝트에도 적용되었다.

3. Policy Conflict 의 예

NS policy 와 NFVI policy 간의 충돌이 발생하는 여러 예를 모델링하고 검증하는 연구를 진행하고 있다. 한 가지 예로, NS policy 는 그림 2 에서 시간당 접속 횟수(cps)가 300k 를 초과할 경우 scale-out 하여 빠른 네트워크를 요구한다. 그러나 NFVI policy 의 경우, 가상머신의 수가 5 대이상이면 에너지의 효율을 위해 scale-out 을 허용하지 않는다. 만약 가상머신이 5 대이며 cps 가 300k 를 초과하면 각 policy 간의 충돌이 발생한다.

- NS#1 policy
condition: if cps > 300k
action: scale-out
- NFVI policy
condition: if number_of_VMs >= 5
action: scale-out is not allowed
(for energy efficiency)
- conflicts when:
number_of_VMs = 5 && NS#1_cps >300k

그림 2. NS policy 와 NFVI policy 의 Conflict

NS#1 policy 를 Z3 Library 를 이용해서 Python 코드로 구현하면 그림 3 과 같다. Policy Conflict#1 에 대해 검증하기 위해, Z3 library 를 import 한다. 그 다음으로 Z3 Solver()클래스를 s 라는 인스턴스로 생성한다. d1 은 scale-out 을 의미하는 Boolean 변수이며 Not(d1)은 no scale-out 을 의미한다. vm, cps 는 int 형 변수이며 cps >300k -> d1, vm>=5 -> ¬d1 의 policy 에서 cps==400k, vm==5 의 경우 d1 이 True 이고 ¬d1 가 False 이므로 Unsat 이며 conflict 발생을 나타낸다. 위의 코드를 실행하면 unsat 이 출력된다. Sat 은 Satisfiability 이며 한 수식이 Sat 하다는 의미는 그 수식이 참이 되는 모델이 하나는 존재한다는 것을 의미한다. 한 수식이 unsat 인 경우는, 그 수식이 참이 되는 경우가 전혀 없음을 의미한다. 그러므로 cps>300k 이면 d1 이 참이고 vm>=5 이면 ¬d1 가 참이다. 그림 3 에서 cps==400k, vm==5 이면 참인 수식이 없으므로 Unsat 이 되며 Conflict 가 발생함을 알 수 있게 된다.

```
from z3 import *

s=Solver();
d1=z3.Bool('d1');
vm, cps=z3.Ints('vm cps');

s.add(Implies(cps>300000, d1))
#NFVI Policy
s.add(Implies(vm>=5, Not(d1)))
#NS Policy
s.add(cps==400000)
s.add(vm==5)
#Conflict condition

if(s.check()==z3.sat);
    print("sat")
    print(s.model())
else:
   print("unsat")
```

그림 3. Z3 Library 를 이용한 policy 들의 일치성 검증 코드

또 다른 예로, 그림 5 에서 NS#2 policy 는 시간당 접속 횟수(cps)가 100k 미만으로 제한한다. NFVI policy 의 경우, 트래픽 지역화를 위해, 단일 NFVI-PoP 안에 모든 VNFs/VLs 을 둔다. 만약 cps 가 100k 미만이고 단일 NFVI-PoP 가 아닐 경우에 또한 Policy 간의 충돌이 발생한다. 이 경우의 일치성 검증은 그림 5 과 같다.

- NS#2 policy
constraint: cps < 100k
- NFVI policy
constraint: all VNFs/VLs in a single NFVI-PoP
(for traffic localization)
- conflicts when:
No such single NFVI-PoP with cps < 100k
(for cps<100k, requires VNFs/VLs in two NFVI-PoPs)

그림 4. Policy 간의 Conflict

```
from z3 import *

s=Solver();
cps NFVI_PoP=z3.Ints('cps NFVI_PoP');

s.add(NFVI_PoP==1)
#NFVI Policy
s.add(cps<100000)
#NS Policy
s.add(Implies(cps<100000, NFVI_PoP==2))
#Conflict condition

if(s.check()==z3.sat);
   print("sat")
   print(s.model())
else:
  print("unsat")
```

그림 5. Policy 들의 일치성 검증 코드

NS#2 policy 를 Z3 Library 를 이용해서 Python 코드로 구현하면 그림 5 와 같다. 검증을 위해 cps, NFVI_PoP 는 int 형 변수로 선언한다. NFVI Policy 에서 트래픽의 지역화를 위해 NFVI_PoP==1 로 설정하고, 초당 접속 횟수 cps 를 100k 미만으로 경우가 True 이다. cps<100k -> NFVI_PoP==2 의 경우 cps<100k 이 True 이고 NFVI_PoP==2 가 False 이므로 Unsat 이며 conflict 발생을 나타낸다.

4. 결 론

NFV 에서는 NS policy 와 NFVI policy 는 서로 다른 요구사항에 의해 충돌이 발생할 수 있다. 그러므로 policy 간에 충돌이 일어나기 이전에 이를 확인할 수 있는 연구가 반드시 필요하다.

본 연구에서는 NS policy 와 NFVI policy 를 모델링하고 SMT solver 인 Z3 를 이용하여 policy 간의 충돌(conflict)을 모델링하고 정형 검증하는 방법을 제시하였다.

참고문헌

[1] Network Functions Virtualisation, Introductory White Paper, ETSI, Oct. 2012.

[2] 이승익, 이종화, 신명기, 김형준, 손승원, "스마트인터넷을 위한 SDN 및 NFV 표준기술 동향분석", 2014Electronlcs and Telecommunications Trends, 2014.

[3] Alessandro Armando, Jacopo Mantovani, and Lorenzo Platania, ***Bounded Model Checking of Software using SMT Solvers instead of SAT Solvers***, International Journal on Software Tools for Technology Transfer (STTT), Volume 11 Issue 1, pp. 69-83 January 2009

[4] Leonardo de Moura and Nikolaj Bjørner, ***Z3: An Efficient SMT Solver,*** 14th international conference on Tools and algorithms for the construction and analysis of systems, pp. 337-340, 2008

Safety Critical System 에서 요구사항 검증 및 통합 명세화 방법제안

임혜선

아주대학교 소프트웨어특성화학과
경기도 수원시 영통구 월드컵로 206
rena@ajou.ac.kr

이석원

아주대학교 소프트웨어융합학과
경기도 수원시 영통구 월드컵로 206
leesw@ajou.ac.kr

요약:

Safety Critical System 의 안전성은 요구사항 단계부터 고려해야 하는 것으로, Generic Requirement 와 Safety Requirement 로 구분되고 System Requirement 로 통합된다. 하지만 자연어로 작성된 요구사항은 여러 이해관계자들에게서 도출되어 모호함과 부정확성 때문에 결함을 검출하기 어렵다. 이러한 문제를 해결하기 위해 요구사항을 표준문안을 적용한 자연어로 작성하여 모호함과 부정확성을 해결하고, 이를 Safety Case 작성에서 적합성을 인정받고 있는 GSN(Goal Structure Notation) 으로 Model 화하여 안전 적합성을 증명한다. 최종적으로 이러한 과정을 통해 발견된 요구사항의 결함을 수정하여 개발할 수 있는 요구사항의 통합 명세화 방법을 제안한다.

핵심어: Safety Critical System, Requirement Specification, Requirement Verification, Boilerplate, Goal Structure Notation, Safety Case

1. 서론

최근 대부분의 시스템은 SW 가 내장되어 사용자에게 편의성을 제공한다. 자동차와 같은 Safety Critical System 에서는 편리함과 동시에 SW 안전성 확보는 필수이다. 예를 들어, 최근 Toyota 자동차에서 두뇌역할을 하는 전자제어장치(Electronic Control Unit)에 내장된 SW 의 설계적인 오류로 인해 급 발진 사고가 유발되었다 [1]. 이처럼 Safety Critical System 에서 SW 의 결함은 심각한 결과를 유발하므로 개발의 첫 단계인 요구사항(Requirement) 단계부터 안전성을 고려한 개발을 수행해야 한다. 이를 해결하기 위해 실제로 소스코드나 시스템 수준에서 다양한 테스트를 통해 안전성을 검증 하지만, 요구사항의 결함은 테스트 단계에서 발견하기 어려우며, 결함을 발견하더라도 큰 비용이 발생하기 때문에 시스템을 제어하는 SW 의 안전성을 확보하기 위한 요구사항 단계는 매우 중요하다고 볼 수 있다 [2].

Safety Critical System 에서 System Requirement 는 기능적인 내용을 명시하는 Generic Requirement 와 품질요인을 명시하는 Safety Requirement 로 구분된다.

Figure1 [3] 에 보여 지는 것처럼 보편적으로 System Requirement 취합 시, Requirement Team 에서는 여러 이해관계자로부터의 요구사항을 통합하여 시스템 레벨로 구체화 시킨다.

특히 안전성을 위해 Safety Team 에서는 Hazard Analysis, Fault Tree Analysis 등의 체계적인 분석절차에 의해 Safety Requirement 를 도출하여 Requirement Team 에 전달한다. 이 때 각 팀에서 취합된 요구사항이 서로 상이한 표현이나, 추상화 수준으로 인해 혼선이 발생하고 있다 [4, 5]. 이를 해결하기 위해 요구사항 명세 단계에서부터 이해관계자들이 공통적으로 이해할 수 있는 형태로 요구사항을 명세하는 것이 필요하다.

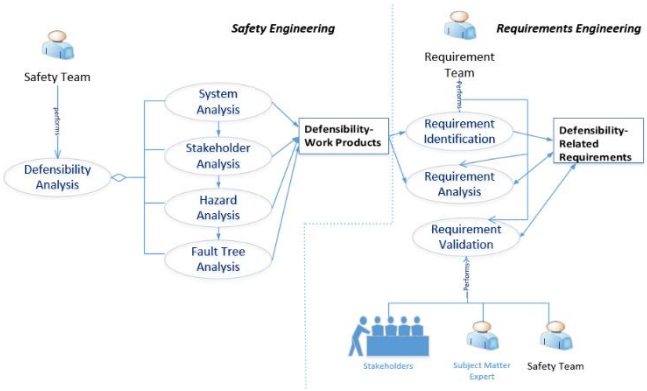


Figure1. Defensibility &Requirements Engineering

일반적으로 요구사항은 자연어(Natural Language)로 작성된다. 자연어로 작성된 요구사항은 이해하기 쉽지만, 여러 이해 관계자들에게서 도출되어 모호함과 부정확성을 갖고 있어 결함이 있어도 이에 대한 검증이 어렵다. Model 기반 작성법은 요구공학 프로세스에서 수학적, 논리적인 표현으로 요구사항의 결함을 식별하는데 체계적이다. 하지만, 자연어로 작성된 요구사항을 Model 로 변환하는 과정이 추가적으로 필요하다. 게다가, 이 과정은 여러 이해관계자들로부터 자연어 요구사항을 도출하기 때문에 모호함과 부정확성 문제가 발생한다.

이러한 어려움을 해결하고자 본 논문에서는 표준문안(Boilerplate)으로 자연어 요구사항을 작성하고 GSN(Goal Structure Notation) 표기법으로 Modeling 한다. 제안하는 방법에서는 표준문안 작성과 Modeling 을 통해 요구사항의 모호함과 부정확성으로 인한 결함을 찾을 수 있다.

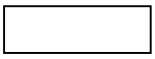
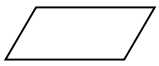
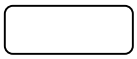
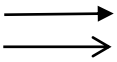
2. 제안방법

본 장에서는 논문에서 사용하는 배경기술들과 제안하는 Safety Critical System 에 적합한 요구사항 개발방법을 설명한다.

본 논문에서 활용하고 있는 배경기술로는 표준문안(Boilerplate)과 GSN(Goal Structure Notation)이 있다. 먼저 표준문안은 선 정의된 서식에 맞춰 요구사항을 작성하는 것으로, 표준문안에 맞춰 작성할 경우 이해관계자의 이해를 돕고 혼란을 방지할 수 있다. 또한 Requirement Specification 전체에 걸쳐 표현의 균일성을 가지는 Semi-formal 언어로 작성되기 때문에 모호함과 부정확성을 줄일 수 있다 [6].

GSN 은 시스템이 안전하다는 것을 관련기관에 증명하기 위한 Safety Case (Safety Evidence: Fault tree, Failure mode 등을 표현) 작성에서 적합성을 인정받고 있는 표기법이다. GSN 은 명시적으로 Safety argument 의 개별요소(Requirement, Claims, Evidence, Context) 사이에 존재하는 관계를 나타낸다 [7]. GSN 에서 표현하는 기본요소는 Table1 과 같다.

Table 1. GSN 주요요소

주요요소	표기법	설명
Goal/Sub-goal		안전, 신뢰도 등 시스템 속성에 대한 요구조건/목표
Strategy		모든 Hazard 제거에 의한 논증
Solution		FTA 등 Goal 에 대한 근거 자료
Context		식별된 모든 Hazards 의 결과로 도출된 Solution
Undeveloped Goal		추후 개발될 Goal
연결방법		수직적 전개 수평적 전개

본 논문에서는 앞서 설명한 두 가지 배경기술을 활용하여 자연어 요구사항을 표준문안에 맞춰 새롭게 작성하고 이를 활용하여 요구명세 단계에서부터 결함을 쉽게 찾고 Safety 적합성을 보증하여, 안전성 있는 시스템을 개발할 수 있는 새로운 요구사항 명세 방법을 제안한다.

Table 2. 제안하는 표준문안

필수	Entity: 요구사항을 수행하는 주체 Precondition: 요구사항 수행을 위한 선행조건 Transition: 상태(state), 행동(action)의 변화 Condition: 수행조건 – 환경, 속도, 시간 등
옵션	Exception: 예외 상황 Function: 수행하는 이벤트의 기능

제안하는 표준문안의 요소로는, Table2 에서 보이는 것과 같이 안전기능을 수행하는 주체인 Entity, 안전기능에 대한 시스템의 행위, Timing 과 관련된 요소를 표현하기 위한 Transition 과 Condition 이 있으며, 요구사항의 우선순위나 사전조건을 위해 Precondition 을 추가하였다. 부가적으로 시스템의 예외사항과, 그 요구사항이 수행하는 이벤트의 기능명칭을 추가할 수 있도록 하였다. 이렇게 제안한 표준문안 형식을 기반으로 자연어 요구사항 작성시 모호성과 부정확성에 의한 결함 검증이 가능하다.

다음으로 표준문안에 맞춰 작성된 자연어 요구사항을 Safety Critical System 의 요구사항으로 표현하기에 적합하도록 기존 GSN 에 몇 가지 연결선을 추가한 새로운 표기법을 사용하여 Model 화 시킨다. 기존 GSN 에서 추가된 요소는 Table3 과 같으며, Precondition 을 추가하여 요구사항의 우선순위를 표현하여 요구사항의 불일치성을 검증할 수 있고, Exception 속성을 추가하여 시스템에서 발생할 수 있는 예상치 못한 상황에 대한 요구사항까지 담아 Safety Critical System 의 요구사항을 표현하기에 적합하도록 하였다. 따라서 자연어 요구사항을 GSN Model 로 변이시키는 과정에서 요구사항의 결함요소를 검증하고, 안전적합성을 증명 할 수 있다.

Table 3. 확장된 GSN 표기

주요요소	표기법	설명
Precondition	Pc	Precondition 연결
Exception	Ex	Exception
Goal 분해		Decomposition

최종적으로 Safety Team 과 각 이해관계자들에게서 작성된 표준문안 요구사항 및 GSN Model 을 Requirement Team 에서 하나로 통합하여, System Requirement 로 작성 및 관리 된다. 이러한 과정으로 작성된 요구사항은 결함을 쉽게 파악하고 수정할 수 있다.

3. 사례연구

본 장에서는 사례 연구를 통하여 제안방법의 적합성을 입증한다.

여러 이해관계자들에게서 자동차가 예상치 못한 상황 인식 시, 안전을 위한 기능이 필요하다는 요구사항이 도출되어 “후측방 경보 시스템” 을 개발하기로 하였다.

3-1. Requirement Team: 표준문안 작성 및 배포

Requirement Team 에서는 관련 팀에 표준문안 서식에 맞춰 요구사항을 작성하고 요구사항은 GSN 기법으로 Modeling 할 것을 요청한다.

3-2. Safety Team: 요구사항 도출 및 Modeling

Safety Team 은 Hazard Analysis 과정을 통해 "후측방 경보 시스템" 의 Safety Requirement 를 작성한다. 만약 표준문안 없이 자연어 요구사항을 작성 할 경우 아래 (1)과 같은 형태로 요구사항이 도출된다.

(1) *후측방에 차량이 감지되면 실외미러에 경보등이 켜지고 헤드업 디스플레이 화면에 황색 경보등이 켜진다* [8].

작성된 요구사항은 결함이 없어 보이지만, 제안하는 표준문안의 요소와 맞춰보면 Precondition 이 없다는 것을 확인할 수 있다.

Entity: 운전자가 Transition: 실외미러에 경보등이 켜지고, 헤드업 디스플레이 화면에 황색 경보등이 작동 Condition: 후측방에 차량이 감지되면

이 요구사항의 경우 어느 상황에서 자동차가 후측방의 차량을 감지하는지 알 수 없다. 즉, 자동차가 멈춰있는 상태에서 동작하는지 혹은, 운행 중에 동작하는지에 대한 모호함이 있다. 다른 개발자나 이해관계자들이 자동차가 운행 중에 동작한다고 짐작은 해도, 확신은 할 수 없다.

이를 제안하는 방법의 GSN 을 통해 Model 화 하면 Figure2 처럼 표현되어 요소간의 중요한 연결을 찾지 못함으로써 해당 요구사항이 어떻게 도출되었는지 알 수 없다.

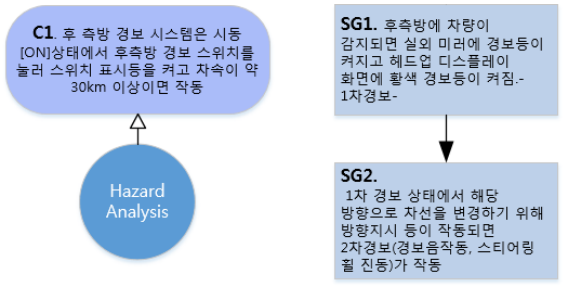


Figure 2. 표준문안 적용 안 된 자연어 요구사항의 GSN Model

하지만, 제안하는 표준문안에 따라 "후측방 경보 시스템" 의 자연어 요구사항을 작성할 경우 (2)와 같은 형태로 작성된다.

(2) *운전자가 후측방 경보 설정 후 후측방에 차량이 감지되면 실외미러에 경보등이 켜지고 헤드업 디스플레이 화면에 황색 경보등이 작동*

작성된 요구사항을 요소 별로 살펴보면 아래와 같다.

Entity: 운전자가 Precondition: 후 측방 경보 설정 후 Transition: 실외미러에 경보등이 켜지고, 헤드업 디스플레이 화면에 황색 경보등이 작동 Condition: 후측방에 차량이 감지되면

이처럼 표준문안의 조건에 만족하여 작성 된 요구사항은 *[Precondition: 후측방 경보설정 후]*를 명시함으로써 Safety Analysis 를 기반으로 도출된 *[C1. 후측방 경보 시스템은 시동[On] 상태에서 후측방 경보 스위치를 눌러 스위치 표시등을 켜고 차속이 약 30km 이상이면 작동한다]* 라는 사전조건과 요구사항이 연결됨으로써 다른 의미로 해석할 수 없고, 요구사항의 정확성을 갖춤과 동시에 안전 적합성도 증명한다.

이를 GSN 에 Model 화 시키면 Figure3 에 좌측하단에 보이는 것과 같이 표현되어 도출된 요구사항의 안전 적합성과 정확성을 확인 할 수 있다.

3-3. Requirement Team: 요구사항 통합

최종적으로 Requirement Team 에서는 Safety Team 에서 작성된 자연어 요구사항과 GSN Model 을 통합하여 전체 요구사항의 정확성과 안전적합성을 확인하고 시스템 개발에 착수 할 수 있다.

Figure3 은 제안방법에 따른 Safety Requirement 에 요구사항을 추가하여 완성된 시스템의 전체적인 GSN Model 이다. 본 논문에서 제안하는 GSN Model 로 요구사항을 도식화하여 Safety Requirement 의 적합성을 증명할 수 있다.

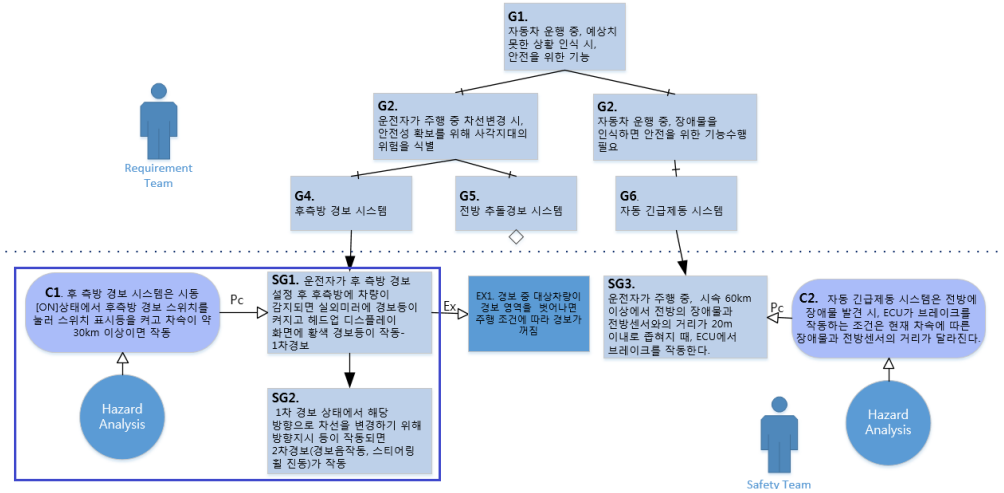


Figure 3. 표준문안 적용한 자연어 요구사항의 통합 GSN Model

4. 관련연구

기존의 관련연구들은 표준문안에 맞춰 자연어 요구사항을 작성하고 State Chart, EARS, Cause-Effect Graph, Use Case, Sequence Diagram 등의 Model 로 변환하는 방법을 사용하였다.

State Chart 는 정형명세의 하나로 State(상태)를 기반으로 시스템의 행위를 모델링 하여 시간의 상태에 따른 상태변화를 기술하기 좋은 방법이다 [9]. 형식적 요구사항 문법을 이용하여 Cause-Effect Graph 를 생성하는 방법은 [10] 요구사항의 누락을 판단하는데 효율적이며, EARS (Easy Approach Requirements Syntax)라는 형식적 요구사항 문법을 분석하여 자동으로 Use Case Diagram 으로 생성해주는 방법은 누락되는 요구사항이 존재하지 않도록 요구사항의 Type 을 정의하여 관리하는데 장점이 있다 [11]. 그러나 이러한 Model 변환기법 들에서는 Hazard Analysis 의 결과와 그로부터 도출된 Safety Solution 을 포함 시킬 수 없기 때문에 요구사항들이 안전에 적합하다는 것을 증명할 수 없다.

제안하는 방법은 Safety Case 작성에 적합한 GSN Model 에 Safety Critical System 에서 반드시 필요한 요구사항의 속성을 추가하여 요구사항의 결함요소를 검증하고 안전 적합성을 증명할 수 있다.

5. 결론

본 논문에서는 Safety Critical System 에서 표준문안을 적용을 통해 정확하고 모호함이 없도록 요구사항들을 명세화 하였고, GSN Model 을 사용하여 Safety 적합성 증명 및 요구사항의 불일치성을 검증할 수 있는 방법을 제안하였다. 이 방법을 통해 Requirement Team 과 Safety Team 의 요구사항 통합개발을 통해 전체 요구사항의 결함을 발견하고 정확성을 보장할 수 있으며, 이해관계자의 요구에 부합하는지 검증할 수 있다.

향후 연구방향으로, 실제 Safety Critical System 에서 사용되는 요구사항을 제안된 방법에 적용하여 적합성을 증명할 계획이다. 또한 연구 내용을 확장시켜, Web 기반 프로그램에서 요구사항을 자동으로 Model 화 하고 요구사항의 변경사항, 이력관리까지 가능하도록 하며 이를 지원하는 도구를 개발할 것이다.

6. Acknowledgement

이 논문은 2013 년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보・컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(2013M3C4A7056233)

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 실전적 SW 교육(SW 중심대학)지원사업의 연구결과로 수행되었음(R7115-15-1005)

참고 문헌

[1] 김재석, 2012. [Online]. Available: http://www.ddaily.co.kr/news/article.html?no=86276

[2] C. Technologies, "3 Excellent Reasons Why To Invest In A TCoE," 03 03 2015. [Online]. Available: http://www.cigniti.com/blog/test-center-of-excellence/

[3] D. Firesmith, "A Taxonomy of Safety-Related Requirements," 2004. [Online]. Available: https://resources.sei.cmu.edu/asset_files/WhitePaper/2004_019_001_29423.pdf

[4] N. L. Seoungike Yang, "The Case Study of ISO26262 Product Requirements Analysis Applying Requirements Engineering," in *한국자동차공학회 학술대회*, 2012

[5] D. G. Firesmith, "Engineering Safety and Security Related Requirements for Software Intensive Systems," in *ICSE*, 2007

[6] V. Johannessen, "CESAR – text vs. boilerplates," 2012. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:566314/FULLTEXT01.pdf

[7] R. W. Tim Kelly, "The Goal Structuring Notation – A Safety Argument Notation," 2004

[8] 현대자동차, "ABOUT HYUNDAI," [Online]. Available: http://blog.hyundai.com/1793.

[9] 김진현, 김창진, 심재환, 박승현 and 최진영, "Modeling Requirements in Natural Language with Statecharts," in *한국정보과학회*, 2006

[10] 조선영 and 조상훈, "Requirements Based Testing Technology for Development of Automotive E/E," in *한국자동차공학회*, 2012

[11] A. Mavin, "Using EARS+ (Easy Approach to Requirements Syntax Plus) to vary the level of detail in Natural Language requirements," *Tutorial sessions at Requirement Engineering Conference,* 2012

소프트웨어 연구 문서 산출물의 추적을 위한 연관성 링크 정보 모델 기반 전문가 시스템 설계

백두산 [1], 이병정 [2], 이정원 [1]

[1] 아주대학교 전자공학과
경기도 수원시 영통구 월드컵로 206
whitedusan@gmail.com, jungwony@ajou.ac.kr

[2] 서울시립대학교 컴퓨터과학과
서울 동대문구 전농동 163(전농동 90)
bjlee@uos.ac.kr

요약: 소프트웨어의 비가시적인 특성으로 인하여 형상 관리, 특히, 형상 식별과 추적성에 관한 연구들이 활발히 진행되고 있다. 하지만 이러한 연구들은 소프트웨어 개발 주기에 초점을 두고 진행되어 소프트웨어 연구 주기에서 산출되는 산출물들에 대하여 직접적으로 적용하기에는 한계를 가지고 있다. 본 논문은 소프트웨어 연구 주기에서 산출되는 산출물 가운데 가장 보편적이고, 일반적으로 사용되고 있는 종류인 소프트웨어 연구 문서 산출물에 대한 추적을 지원하기 위해서 전문가 시스템을 제안하였다. 본 논문에서 제안하는 전문가 시스템은 모델 체크부에서 연구 문서 산출물을 위한 TIM(Trace Information Model)인 RLIM(Relevance Link Information Model)의 무결성을 확인하고, 연관성 링크 생성부에서 연관성 링크를 생성한다. 그 뒤, 연관성 규칙 검증부에서 모든 연관성 링크에 대응 되는 연구 문서 산출물 내 형상 항목의 내용이 연관성 규칙을 준수하고 있는지를 검증함으로써 소프트웨어 연구 문서 산출물의 추적에 기여한다. 예비 실험을 통하여 본 논문에서 제안하는 전문가 시스템이 소프트웨어 연구 문서 산출물의 추적을 지원할 수 있음을 확인하였다.

핵심어: 소프트웨어 연구 문서, 문서 산출물, 추적성, 연관성 분석, 형상 관리

1. 서론

최근 소프트웨어 연구 개발의 규모가 커지고, 복잡성이 증가함에 따라 소프트웨어 연구 개발의 생명주기에서 산출되는 산출물들에 대한 관리의 필요성이 증대되고 있다. 이러한 필요성에 대응하기 위하여 소프트웨어 공학, 특히, 소프트웨어 형상 관리 분야에서의 연구들이 활발히 진행되고 있다[1]. [2,3]에 따르면 형상관리는 "형상 항목을 정의하고 식별하는 과정으로, 시스템의 생명주기 동안 항목을 제어하며 항목에 대한 변화요구의 형상 항목의 상태를 기록하고 완전성을 증명하는 활동" 이다. 또한, [4]에 따르면 소프트웨어 형상 관리는 "소프트웨어의 진전을 제어하는 것으로써 보다 실용적으로 표현한다면 개발자의 관리하에 소프트웨어 관련 산출물의 진전을 가능하게 하여 제한된 시간 안에 품질을 만족할 수 있도록 하는 활동"이라 할 수 있다. 다시말해 형상관리를 위해 설계된 요구사항, 설계, 구현물 등의 산출물 집합인 형상을 관리하는 총체적인 활동이다.

형상관리 표준인 [2]에 따르면 형상관리는 전통적으로 네 가지 기능으로 구별되며, 연구 개발의 주기에 맞추어 나열하자면 형상 식별, 형상 통제, 상태 보고 및 형상 감사와 검토로 나누어진다. 형상 식별 활동에서는 형상관리를 수행할 형상 항목을 식별하고, 식별된 형상 항목은 형상 통제 과정을 통해 관리된다. 구체적으로 형상 통제 과정에서는 기준선이 되는 형상 항목들에 대한 변경을 요청, 평가하여 승인 또는 비승인하고 변경을 검증, 구현 및 배포한다. 형상 상태 보고 활동에서는 프로젝트 형상 항목들의 상태를 기록하고 보고하며, 위의 모든 활동들은 형상 감사와 검토 하에 수행된다.

하지만, 소프트웨어의 비가시적인 특성 때문에 아무리 소프트웨어 생명주기에서 산출되는 산출물을 통하여 소프트웨어의 형상을 식별한다고는 하여도 형상 식별 활동에는 본질적으로 커다란 한계를 지니고 있다. 이와 같은 형상 식별은 단순히 요구사항 분석 활동과 같이 소프트웨어 생명 주기 초반에 수행되는 형상 항목 분별만을 의미하는 것이 아니며, 전주기에 걸쳐 지속적으로 변화하는 형상 항목을 추적하는 것 또한 포함한다. 따라서 이와 같은 한계를 개선하지 못한다면 형상관리를 통한 소프트웨어 품질 개선은 어려울 것이다.

한편, 소프트웨어의 비가시성으로 인한 형상 항목

"이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보・컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No. NRF-2014M3C4A7030504)."

식별의 애매함을 해결하기 위하여 추적성을 활용하는 연구들이 진행되고 있다[5]. 일반적으로 추적성이란 특정 대상이 갖고 있는 이력, 적용 현황 또는 적용 위치를 추적하기 위한 능력으로서, 소프트웨어 개발 분야에서는 구체적으로 요구사항 추적성이라 불리며, 고객의 요구사항으로부터 소프트웨어 생명주기에서 생산되는 산출물들 사이의 관계를 기술하고 추적할 수 있는 능력을 말한다[6]. 소프트웨어 개발 분야에서 추적성은 본래 영향 분석, 규정 준수 검증, 코드 역추적, 회귀 분석, 요구사항 타당성 검증 등의 소프트웨어 공학 활동을 지원하기 위한 능력으로서, 미국 국방부의 "Critical Code: Software Producibility for Defense" 보고서에 따르면 현재와 미래의 소프트웨어 집중적인 시스템의 안전과 정확성을 보장하기 위해 요구되는 7 개의 연구 분야 중에 하나로 소개되었다[7,8]. 추적성은 형상관리와 마찬가지로 형상을 식별하는 과정이 필요한데, 이를 위해서 가장 많이 사용되는 방법은 식별자 태킹을 통한 추적성 테이블의 활용이다. 또한 근래에는 추적의 자동화를 목표로 확률 접근방법, 벡터 공간 모델 등을 활용한 정보 검색 기법을 사용하거나, TIM(Traceability Information Model)을 활용하는 방법이 소개되고 있으며, 사용자 피드백을 통하여 정확도를 개선하는 방안도 소개되고 있다.

그러나, 앞서 설명한 소프트웨어 형상관리, 추적성에 관한 연구 및 그 방법들은 소프트웨어의 개발 주기만을 고려하고 있어, 소프트웨어 연구 주기(소프트웨어 요구사항 분석 이전에 진행되는 전산 및 그 응용기술에 대한 지적 탐구를 수행하는 기간)를 대상으로 적용하기에는 문제가 있다. 예를 들면, 소프트웨어 개발 주기에서 산출되는 산출물은 소프트웨어 요구사항 명세서, 소프트웨어 설계 기술서, 소프트웨어 테스트 명세서, 코드 산출물 등으로서 비교적 명확하고 상세한 요구사항을 기반으로 일정한 형식에 맞추어 작성되는 반면, 소프트웨어 연구 주기에서 산출되는 산출물은 연구 개발 과제 계획서, 연차 별 보고서, 최종 보고서 등과 같이 비기능적이며 매우 추상화된 요구사항이 연구자의 이해를 목적으로 하는 구조에 맞추어 작성된다. 이러한 이유 때문에 요구 사항 기반의 일정한 형식을 통해 형상을 식별하는 기존의 연구들은 연구 문서 산출물에 대응 할 수 없다. 비록, [6]에서 제시하는 바와 같이 요구사항을 위한 사전 요구사항 추적성을 'Pre-requirements specification traceability' 라고 칭하여, 이들에 대한 고려의 필요성 및 방안에 대한 논의를 진행해 왔으나, 이는 기존의 연구들과 마찬가지로 소프트웨어 연구 주기를 대상으로 하기 보다는 소프트웨어 개발 주기를 대상으로 하는 논의였다. 특히, 소프트웨어 원천 기술 연구와 같은 프로젝트는 개발 주기 이전의 연구 주기의 중요성이 크고, 기간이 길어 이에 대한 관리가 필수적이라고 할 수 있다. 하지만 앞서 논의한 바와 같이 기존의 형상관리, 추적성에 관한 연구들은 소프트웨어 연구 주기가 갖고 있는 이질적인 특성에 대응하지 못하기 때문에 소프트웨어 연구 주기를 대상으로 진행되는 연구가 더욱 중요한 실정이다.

이러한 문제점을 해결하기 위하여 본 논문에서는 소프트웨어 연구 주기에서 산출되는 여러 종류의 산출물들 중에서 가장 보편적이고, 일반적으로 산출되고 있는 종류인 문서 산출물에 초점을 두고, 이들의 추적을 위한 전문가 시스템을 제안한다. 본 논문에서 제안하는 전문가 시스템은 기 연구를 통해 정의한 연관성 링크(연구 문서 산출물을 대상으로 하는 추적성 패스)의 메타 모델인 Relevance Link Information Model (RLIM) 을 활용하여 연구 주기의 지식 베이스를 구축하고, RLIM 의 형식 적절성, 유효성 등을 판단한다. 그 뒤, RLIM 과 실제 연구자가 생성한 문서를 바탕으로 연관성 링크를 생성 하고, 해당 연구 문서의 내용에 기반하여 소프트웨어 연구 문서 산출물의 추적성을 검증한다. 이와 같은 과정을 통해 본 논문에서는 기존의 연구들이 수행하지 못하였던 소프트웨어 연구 주기에서 산출되는 연구 문서 산출물을 추적함으로써 소프트웨어의 품질 향상을 도모한다.

본 논문의 구성은 다음과 같다. 2 장에서는 관련 연구를 설명하고, 3 장에서는 소프트웨어 연구 주기 내에서 산출되는 연구 문서 산출물과 개발 주기 내에서 산출되는 개발 문서 산출물과의 비교 및 기 연구 내용의 결과를 바탕으로 요구사항을 도출한다. 4 장에서는 소프트웨어 연구 문서 산출물의 추적을 위한 전문가 시스템의 구조 및 방안에 대해서 제안한다. 5 장에서는 본 논문에서 제안한 구조에 맞추어 구현한 전문가 시스템의 예비 실험을 통해 본 논문의 타당성을 확인한다. 끝으로 6 장에서 결론과 함께 마무리한다.

2. 관련연구

소프트웨어의 품질을 보장하기 위해서 소프트웨어 생명주기에서 산출되는 산출물들에 대한 형상관리 연구들이 수행되고 있다.

[9]에 따르면 소프트웨어 형상관리의 기능 요구사항은 그림 1 과 같이 총 8 개의 영역으로 나누어진다. 각 영역은 박스로 나누어져 있으며, 상위 박스는 하위박스 특성에 의존적이다.

- 구성요소(Component): 소프트웨어 제품을 구성하는 부분들에 대한 식별, 저장 및 평가를 지원해야 함.
- 구조(Structure): 소프트웨어 제품의 전체 구조를 표현해야 하며, 이를 구성하는 부분들의 의존관계, 인터페이스, 일관성 등을 식별할 수 있어야 함.
- 구현(Construction): 스냅샷 등을 통한 효율적인

빌드 환경을 지원해야 함.

- 감사(Auditing): 모든 변경 이력을 추적할 수 있어야 하며, 작업 현황을 제공해야 함.
- 상태 보고(Accounting): 제품의 상태를 평가하고, 제품과 프로세스의 모든 관점에 대한 기록을 용이하게 제공할 수 있어야 함.
- 형상 통제(Controlling): 요구사항에서 제품에 이르는 추적성을 토대로 사용자가 수행하려는 변경이 끼치는 영향에 대한 정보를 제공하고, 결점, 버그 등을 추적할 수 있어야 함.
- 프로세스(Process): 소프트웨어 생명주기 모델과 정책에 기반하여 구성원이 자신의 역할을 식별할 수 있게 함.
- 팀(Team): 구성원들의 협업을 도우며, 분쟁을 식별하고, 이를 해결할 수 있도록 함. 또한, 개별적인 작업 공간과 협업을 위한 작업 공간 모두를 제공해야 함.

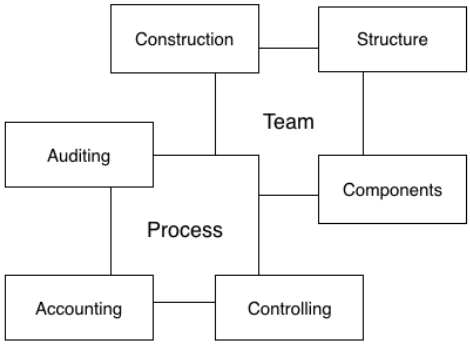


그림 1 소프트웨어 형상관리의 기능 요구사항

[9]가 게재된 1991 년 이후 많은 연구들과 이를 바탕으로 설계된 도구들이 위 기능 요구사항을 지원하기 위하여 노력하고 있다. 하지만 이러한 연구들 및 형상관리 도구들은 소프트웨어 개발 주기만을 고려하고 있기 때문에 소프트웨어 연구 개발 주기에서 산출되는 연구 산출물들을 대상으로 직접적으로 적용 불가능 하다. 특히, [9]에서 제시한 기능 요구사항 중 구성요소, 구조, 형상 통제 등의 요구사항에서 문제가 발생한다.

첫 째, 구성 요소 관련 기능 요구사항을 위한 연구들은 소프트웨어 개발 주기에서 산출된 문서 산출물, 소스 코드, 유즈 케이스, 시나리오 등의 형상을 식별하는 방법, 이들을 저장하는 방법, 평가하는 방법 등에 초점을 맞추어 연구를 진행하였다. 이 중 형상 항목을 식별하는 방법은 [10]에서 제시하는 바와 같이 소프트웨어 생명주기 내에서 생성되는 서로 다른 형상을 구별 가능하게 하는 독립적인 식별자를 갖도록 하며, 사용자가 식별자만을 이용하여 직관적으로 그 종류 및 기능을 유추할 수 있도록 하는 방법이 일반적이다. 하지만, 이와 같이 단순하고 직관적인 방법 조차 소프트웨어 연구 산출물을 대상으로 직접 적용하기 힘들다. 이러한 이유는 소프트웨어 개발 주기에서 생성되는 소프트웨어 요구사항 명세서(SRS), 소프트웨어 설계 기술서(SDD), 소프트웨어 품질 명세서(SPS), 소프트웨어 테스트 명세서(STD)와 같은 문서 산출물은 그 구조가 소프트웨어 요구사항 기반의 일정한 형식을 갖고 분류되어 있어, 이들 각각을 하나의 식별자로 구별하는 것이 [10]에서 문제점으로 지적하는 추상화의 과립 상태 (granularity) 불균형을 일으키지 않지만, 소프트웨어 연구 주기에서 생성되는 연구 계획서, 연차 보고서, 연구 실적보고서 등과 같은 문서 산출물은 연구자의 이해를 목적으로 각기 다른 형식을 통해 분류되어 있어, 이들을 각각 하나의 식별자로 구별할 시, 추상화의 과립 상태 불균형을 일으킨다. 이는 단순히 식별자를 통한 형상 분류 및 분별 시에는 문제가 되지 않지만 추후 형상 통제, 프로세스 등에 쓰일 자동화 과정에서 많은 문제를 야기하기 때문에 이에 대한 고려가 필요하다[11].

둘 째, 구조 관련 기능 요구사항을 위해 소프트웨어 개발 주기에서 산출되는 문서 산출물과 소스 코드 혹은 소스 코드와 소스코드 사이의 의존관계 파악에 대한 연구를 진행하였다[12]. 하지만 이와 같은 연구들은 문서 산출물의 구조 및 식별 방법을 앞서 논의한 바와 같이 개발 관련 문서에 초점을 두고 있어, 구성 요소 관련 기능 요구사항에서의 문제와 같이 연구 관련 문서 산출물에 적용하기에는 한계를 지니고 있다.

셋 째, 형상 통제 관련 기능 요구사항은 형상 추적성, 심각도 예측, 버그 리포팅 등의 분야로 나누어져 연구가 되고 있다. 이 중 추적성에 관한 연구는 형상 통제 관련 기능 요구사항뿐만 아니라, 대부분의 기능 요구사항의 요소 기술이기 때문에 매우 중요한 연구라고 할 수 있다. 특히, CoEST(Center of Excellence for Software Traceability)에서는 'Grand Challenge of Traceability [13]' 라는 연구 의제를 설정하고, 2035 년까지 유비쿼터스 추적성을 달성하기 위하여 연구 중이다. 이와 같은 추적성에 대한 연구는 드래그 엔 드롭 방식, 추적성 테이블을 이용하는 방식과 같이 단순하면서 직관적인 방법뿐만 아니라, TIM 을 활용하는 기법, 확률 접근 방법 또는 백터 공간 모델 등을 활용한 정보 검색 기법과 같이 반자동 혹은 자동으로 추적성을 확보하는 방법에 이르는 다양한 방법으로 진행되고 있다. 또한, 단순히 추적 패스를 생성하고, 관리하는 것 이외에도 효과적으로 이들을 사용자에게 제공하기 위한 Sunburst, Netmap [14] 과 같은 정보 시각화 연구들이 진행 중이며, 자동화의 정확성을 개선하기 위한 사용자 피드백에 대한 연구도 진행 중이다[15]. 하지만 이와 같은 연구들의 기본 전제는 하나의 추적성 패스를 구성하고 있는 두 형상 항목이 요구사항 관점의 의미적 동등함을

갖고 있다는 것에서부터 시작하기 때문에 요구사항 관점에서 작성되어 있지 않은 연구 관련 문서 산출물을 대상으로 위와 같은 추적성에 관련 연구 방법론을 적용하기에는 한계가 있다. 물론, 드래그 엔 드롭 방식 및 정보 시각화에 관한 연구들 그리고 사용자 피드백에 관한 연구들은 연구 관련 문서를 대상으로 하는 추적성 관련 연구에 적용은 가능해 보이나, 그 대상이 갖는 이질적인 특성으로 인하여 실질적인 적용을 통해 검증을 해야 할 것이다.

3. 요구사항 분석

본 논문에서는 소프트웨어 연구 주기에서 산출되는 여러 종류의 산출물 중에서도 가장 보편적이고, 일반적으로 산출되고 있는 종류인 문서 산출물에 초점을 두고, 이들에 대한 추적을 수행하는 것을 목적으로 하고 있다. 이를 위해 먼저, 본 장에서는 소프트웨어 연구 관련 문서 산출물과 개발 관련 문서 산출물의 차이점에 대하여 분석한 뒤, 소프트웨어 연구 관련 문서 산출물의 추적을 지원하기 위한 기 연구 결과인 '연관성 링크' 에 대하여 소개한다. 그 뒤, 소프트웨어 연구 문서 산출물의 추적을 위한 요구사항을 도출한다.

3.1. 소프트웨어 연구 문서와 개발 문서 차이점

앞서 논의한 바와 같이 기존의 연구들은 소프트웨어 개발 주기에 초점을 두고 연구를 진행하였기 때문에 소프트웨어 연구 주기에서 산출되는 산출물들에 대하여 해당 연구 결과를 적용하기에 한계를 지니고 있다. 이러한 근본적인 원인을 알아보기 위해 문서 산출물을 소프트웨어 연구 주기에서 산출되는 연구 문서 산출물과 소프트웨어 개발주기에서 산출되는 개발 문서 산출물로 나누어보았다. 표 1 은 이러한 분류를 통해 나누어진 문서 산출물들의 종류이다.

표 1 소프트웨어 문서 산출물의 종류

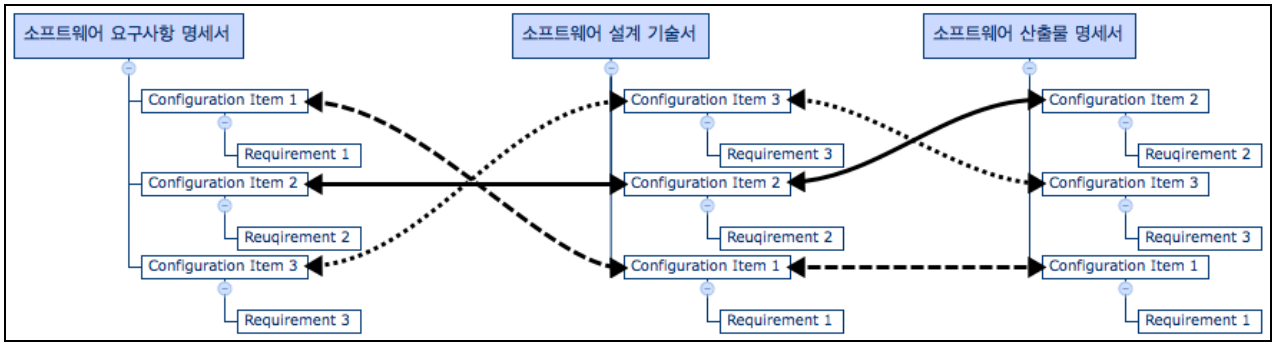
연구 문서 산출물	개발 문서 산출물
연구 계획서	소프트웨어 요구사항 명세서 (SRS)
연차 보고서	소프트웨어 설계 기술서 (SDD)
단계 보고서	소프트웨어 시험 명세서 (STD)
최종 보고서	소프트웨어 산출물 명세서 (SPS)
⋮	⋮

연구 문서 산출물과 개발 문서 산출물은 단순히 산출 시기가 다를 뿐만 아니라 작성 목적과 이를 구성하고 있는 항목, 항목들의 배치 등의 차이를 가지고 있다. 조금 더 구체적으로 살펴보자면 연구 문서 산출물의 경우에는 프로젝트의 목적, 목표, 실적, 진척 상황, 수행 내역, 할 일 목록 등의 매우 비기능적이며 추상화된 요구사항이 연구자의 이해를 목적으로 하여 구성되어 있다. 반면에 개발 문서 산출물의 경우에는 매우 상세적인 요구사항을 기반으로 하여 각 항목들이 정형화된 구조에 맞추어 구성되어 있다. 특히, 개발 문서 산출물은 그 구조를 미국 군사 표준 규격 (MIL-STD-498), IEEE 829, IEEE 830 등과 같은 표준으로 정의하였으며, 몇몇 안전 필수 시스템에서는 이와 같은 템플릿의 사용을 법제화 하거나 가이드하고 있다.

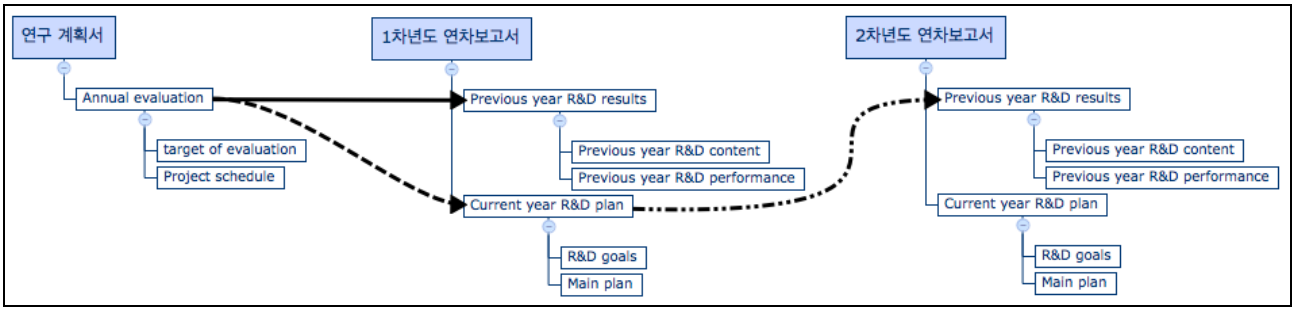
그림 2 는 위와 같은 이유로 인하여 기존의 연구 방법 및 그 기술을 적용할 경우에 발생하는 문제점을 설명하기 위한 그림이다. 그림 2 (a)는 기존에 연구가 되고 있던 추적성 링크 그래프의 예로서 서로 다른 개발 문서 산출물 내에 존재하는 형상 항목들 중에서 동일한 요구사항을 기반으로 작성된 형상들을 연결하여 추적성 링크로 표현한 그래프이다. 따라서, 그래프 내의 소프트웨어 요구사항 명세서와 소프트웨어 설계 기술서, 그리고 소프트웨어 설계 기술서와 소프트웨어 산출물 명세서의 두 추적성 링크를 통해서 소프트웨어 요구사항 명세서에서 소프트웨어 산출물 명세서로 이어지는 추적성 링크를 확보할 수 있으며, 이는 이행성이 성립한다고 표현할 수 있다. 또한 추적성 링크의 경우에는 방향성이 존재 않아, 양방향으로도 동일한 링크를 가질 수 있는 교환법칙이 성립한다. 이러한 이유는 모든 추적성 링크가 서로 다른 두 형상 항목이 동일한 요구사항을 기반으로 하여 작성되어 있다는 하나의 의미를 갖기 때문이다.

반면, 연구 문서 산출물의 경우에는 각 항목들이 요구사항을 기반으로 하고 있지 않아 추적성 링크를 갖지 않는다. 따라서, 본 논문에서는 연구 문서 산출물을 대상으로 하는 연관성 링크라는 개념을 도입하여 그림 2 (b)와 같이 연관성 링크 그래프를 나타내 보았다. 연관성 링크는 서로 다른 연구 문서 산출물 내의 형상 항목 중에서 의미적으로 연관이 있어 연구자가 추적해야 할 필요가 있다고 생각하는 형상 항목 간의 링크로서, 각 연관성 링크는 추적성 링크와는 다르게 서로 다른 의미를 가질 수 있다. 이러한 이유로 인하여 추적성 링크에서는 성립하였던 이행성이 성립하지 않으며, 방향성이 존재하여 교환법칙이 성립하지 않는다. 그림 2 의 연결선의 모양과 화살표는 각각 이행성과 방향성을 의미한다.

소프트웨어 연구 문서 산출물은 작성 목적과 구성 항목, 배치 등의 차이로 인하여 개발 문서 산출물과 이질적인 특징을 가지고 있다. 이로 인하여 기존 연구들이 제시하였던 추적 및 관리 방법들은 연구 문서 산출물을 대상으로 적용하기에는 한계를 갖고 있다. 다음 절에서는 이러한 문제점을 해결하기 위해서 기 연구의 결과인 연관성 링크에 대해 소개한다.



(a) 개발 문서 산출물의 추적성 링크 그래프의 예



(b) 연구 문서 산출물의 연관성 링크 그래프의 예

그림 2 개발 문서 산출물의 추적성 링크와 연구 문서 산출물의 연관성 링크

3.2. 연관성 링크 정보 모델 (RLIM)

본 논문에서는 연구 문서 산출물의 추적을 위하여 기 연구에서 제안한 '연관성 링크 정보 모델 (Relevance Link Information Model; RLIM)'을 사용하였다. 본 절에서는 기 연구에서 제안하였던 RLIM 과 더불어 몇 가지 관련 용어를 소개한다[16].

- 연관성: 서로 다른 문서 산출물에 존재하는 두 형상 항목간의 의미적 관계
- 대응 항목: 하나의 동일한 연관성을 갖는 두 형상 항목
- 연관성 규칙: 대응 항목이 준수해야 하는 규칙
- 연관성 링크 (Relevance Link; RL): 대응 항목과 연관성 규칙의 맵핑
- 연관성 링크 정보 모델 (RLIM): 연관성 링크의 모델로서 문서 종류를 대상으로 하고 있어, 재사용성 등에 이점을 갖고 있음

3.1 절에서 설명한 바와 같이 연관성 링크는 추적성 링크와는 다르게 여러 가지 의미를 가질 수 있기 때문에 연관성 규칙을 통해서 그 의미를 정해야 한다.

표 2 는 기 연구에서 정의한 연관성 규칙으로서 연구 문서 산출물의 형상 항목 추적 및 관리 위해 사용될 수 있다. 연구자는 관리하려는 문서 산출물을 기준 문서 산출물로 하여, 기준 문서 산출물의 형상 항목과 대응 항목을 이루는 비교 문서 산출물의 형상 항목 간에 연관성 규칙 준수 여부를 확인한다. 이와 같은 과정을 통해 연구자는 형상 항목을 추적 및 관리하며, 이러한 과정은 텍스트 추론 등의 고급 기법을 사용하여 자동화 가능하다. 또한 연관성 규칙은 추가, 삭제 및 수정을 통하여 다양한 종류의 연구 문서 산출물을 대상으로 이식 및 확장이 가능하다.

RLIM 은 연구자에 의해 그림 3 과 같은 과정을 거쳐 생성된다. 구체적으로 문서 산출물의 템플릿을 파싱하여 항목을 추출하고, 이들 중에서 연구자가 추적하기 위한 항목들을 형상 항목으로 식별한다. 그 뒤, 그림 2 (b)와 같이 연관성을 갖는 두 형상 항목을 대응 항목으로서 식별하고, 이들이 준수해야 하는 연관성 규칙과 매핑시켜 RLIM 을 완성한다. 완성된 RLIM 은 기본 문서 산출물의 종류와 형상 항목, 비교 문서 산출물의 종류와 형상 항목 등의 4 가지 항목으로 구성된 대응 항목과 이들이 준수해야 하는 연관성 규칙으로 구성되어 있다. RLIM 은 문서의 타입을 기준으로 하고 있어 동일한 문서 타입을 갖는 여러 문서에서 활용 가능하다.

본 논문에서는 위와 같은 과정을 통해 생성된 RLIM 을 입력으로 하는 전문가 시스템을 가정하고 있다.

표 2 연관성 규칙

연관성 규칙	설 명
구조적 (접두사)	형상 항목의 구조가 매우 단순해서 단순한 비교 방법을 통해 추론 가능
구조적 충분관계 (ST-SC)	기준 문서 문서 산출물 내 형상 항목의 모든 내용을 비교 문서 산출물 내 형상 항목이 포함
구조적 필요관계 (ST-NC)	기준 문서 산출물 내 형상 항목이 비교 문서 산출물 내 형상 항목의 모든 내용을 포함
구조적 동등관계 (ST-EC)	기준 문서 산출물 내 형상 항목과 비교 문서 산출물 내 형상 항목의 모든 내용이 동일
구조적 일부 동등관계 (ST-PEC)	기준 문서 산출물 내 형상 항목과 비교 문서 산출물 내 형상 항목의 일부 내용이 동일
의미적 (접두사)	형상 항목의 구조가 매우 복잡해서 수동 혹은 고급 기법을 통해서 추론 가능
의미적 충분관계 (SE-SC)	의미적 충분관계
의미적 필요관계 (SE-NC)	의미적 필요관계
의미적 동등관계 (SE-EC)	의미적 동등관계
의미적 일부 동등관계 (SE-PEC)	의미적 일부 동등관계

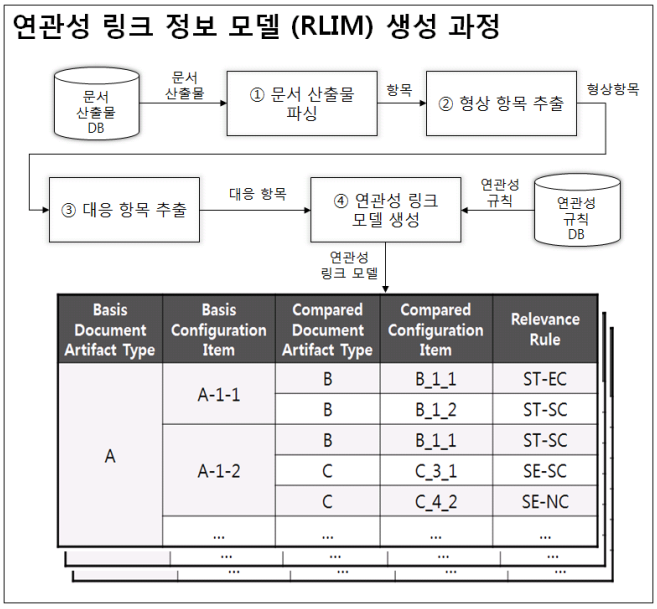


그림 3 RLIM 생성 과정

3.3. 요구사항 도출

소프트웨어 연구 문서 산출물의 추적을 위한 전문가 시스템 구현에 앞서, 요구사항을 도출하였다. 본 절에서 도출하는 요구사항은 소프트웨어 개발 주기에서 산출되는 산출물에 대한 추적성을 지원하기 위하여 제시된 [17,18] 등의 연구에서 도출된 요구사항을 토대로 하였다. 본 논문에서 도출한 요구사항은 다음과 같다.

- *요구사항 1.* 다양한 연구 문서 산출물을 대상으로 적용 가능해야 함
- *요구사항 2.* RLIM 을 사용해야 함
- *요구사항 3.* 불필요한 연관성 링크 생성을 최소화 해야 함
- *요구사항 4.* 점진적으로 연관성 링크가 생성 가능해야 함
- *요구사항 5.* 질의를 통해 추적 가능해야 함
- *요구사항 6.* 연관성 링크가 갖는 여러 속성들을 추가, 삭제 및 수정 가능해야 함
- *요구사항 7.* 소프트웨어 생명주기 전체를 커버 해야 함

위와 같은 요구사항을 충족할 때, 소프트웨어 연구 생명주기에서 생성되는 다양한 연구 문서 산출물에 대한 지속적인 추적이 가능할 것이다. 또한 RLIM 을 사용할 경우, 형상 항목의 추적뿐만 아니라 전략 수립 단계에서의 전략 검증, 문서화를 통한 연구자들의 직관적인 이해, 서드 파티 도구와의 통합 등에 이점을 취할 수 있다. 다음 장에서는 이와 같은 요구사항을 반영하여 연구 문서 산출물에 대한 추적을 지원하기 위한 구체적인 방법을 제안한다.

4. 소프트웨어 연구 문서 산출물의 추적 방법

앞서 정의한 *요구사항 1* (다양한 연구 문서 산출물을 대상으로 적용 가능성), *요구사항 3* (불필요한 연관성 링크 생성을 최소화 해야 함), *요구사항 7* (생명주기 전체를 커버) 은 *요구사항 2* (RLIM 의 사용)을 통해 만족 가능하기 때문에 RLIM 의 사용은 더욱 더 요구된다고 볼 수 있다.

본 논문에서는 RLIM 을 사용하기 위해서 전문가 시스템을 고려해보았다. 전문가 시스템은 본래 전문가와 같은 지적 능력을 갖는 소프트웨어 체계를 목표로 개발된 시스템으로서 감시, 계획, 교육, 모의실험, 선택, 설계, 예견, 제어 등을 문제 유형을 해결하기 위하여 사용되고 있다[19]. 전문가 시스템은 새로운 규칙의 추가뿐만 아니라 생성 규칙을 통한 지식 베이스의 확장이 가능하기 때문에 앞에서 정의한 *요구사항 4* (점진적 연관성 링크 생성 가능성)을 만족한다. 또한 사용자 인터페이스를 통한 *요구사항 5* (질의를 통한 추적 가능성)을 만족할 수 있으며, 규칙의 추가, 삭제 및 수정을 통해 *요구사항 6* (연관성 링크의 속성 추가, 삭제 및 수정 가능성)을 만족한다.

이와 같은 이유로 본 연구에서는 전문가 시스템을 도입하여, RLIM 과 연구 문서 산출물들의 지식 베이스를 구축하고, 몇 가지 규칙을 통해서 RLIM 의 형식 적절성 및 유효성 테스트, 연관성 링크 생성 및 검증 등을 수행한다.

본 장에서는 먼저 소프트웨어 연구 문서 산출물을 추적하기 위한 전문가 시스템의 구조를 소개하고, 각 부분에 대해 설명한다.

4.1. 소프트웨어 연구 문서 산출물 추적을 위한 전문가 시스템의 구조

소프트웨어 연구 문서 산출물의 추적을 위한 전문가 시스템은 그림 4 와 같이 모델 검증부, 연관성 링크 생성부, 연관성 규칙 검증부로 구성되어 있다. 모델 검증부에서는 입력으로 받은 RLIM 의 형식 적절성, 유효성 등을 확인하여 무결성을 검증하고, 연관성 링크 생성부에서는 RLIM 과 연구 주기에서 산출된 문서 산출물의 형상 항목 식별자를 기반으로 하여 연관성 링크를 생성한다. 마지막으로 연관성 규칙 검증부에서는 이전 단계에서 생성된 연관성 링크 내 형상 항목들이 연관성 규칙을 준수하고 있는지 검증함으로써 소프트웨어 연구 문서 산출물의 추적성을 검증한다.

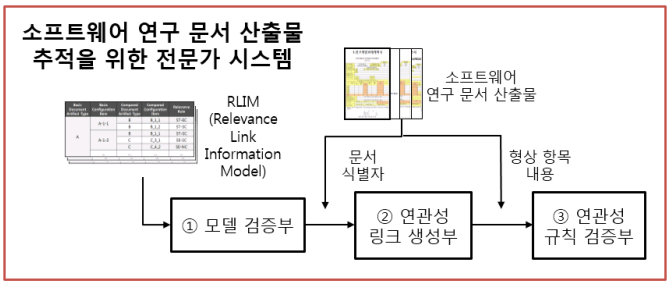


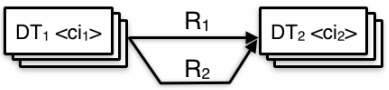
그림 4 소프트웨어 연구 문서 산출물의 추적을 위한 전문가 시스템의 구조

4.2. 모델 검증부

소프트웨어 연구 문서 산출물 추적을 위한 전문가 시스템은 3.2 절에서 설명한 바와 같은 과정을 통해 생성된 RLIM 을 입력으로 한다. RLIM 은 생성 과정에서 연구자의 실수, 생성 프로그램의 오류 등으로 인하여 결점을 포함 할 가능성을 배제할 수 없으므로 이에 대한 검증을 필요로 하게 된다. 따라서, 모델 검증부에서는 결점을 갖고 있는 RLIM 을 찾아내기 위해서 여러 규칙들을 정의하고, 이들을 걸러낸다.

그림 5 는 이러한 규칙들 중 하나인 이중 연관성 링크 결점(Duplicated Relevance Link Problem)을 걸러내기 위한 규칙을 도식화한 그림이다. 이중 연관성 링크 결점은 연관성 링크를 이루는 대응 항목의 경우에는 하나의 연관성 규칙만을 가지고 있어야 함에도 불구하고, 서로 다른 둘 이상의 연관성 규칙을 갖는 연관성 링크들을 걸러낸다.

본 논문에서 제안하는 전문가 시스템은 이와 같은 과정을 통해 품질을 보장한다.

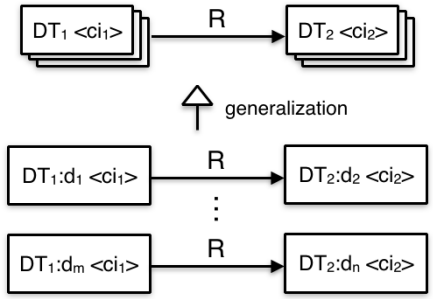


DT : Document Type (문서의 종류)
<ci> : Configuration Item (형상 항목)
R : Relevance Rule (연관성 규칙)

그림 5 이중 연관성 링크 결점

4.3. 연관성 링크 생성부

연관성 링크 생성부에서는 모델 검증부에서 무결성을 검증받은 RLIM 과 소프트웨어 연구 주기에서 산출되는 문서 산출물의 식별자를 통해 연관성 링크를 생성한다. 다시 말해, 연관성 링크 생성부에서는 일반화 되어 있는 RLIM 과 실제 연구 주기에서 산출된 연구 산출물을 특수화(↔일반화) 시켜 연관성 링크를 얻어낸다. 그림 6 는 연관성 링크의 생성을 도식화한 그림이다.



DT : Document Type (문서의 종류)
d : Document (문서)
<ci> : Configuration Item (형상 항목)
R : Relevance Rule (연관성 규칙)

그림 6 연관성 링크의 생성

연관성 링크 생성 과정은 전문가 시스템의 지식 베이스 확장을 위한 규칙으로 정의되어있으며, 수식 (1) 은 이러한 규칙을 술어 논리(1 차 논리)로 표현한 수식이다.

$$(1)\quad \exists dt_1 \exists ci_1 \exists dt_2 \exists ci_2 \exists r \exists d_1 \exists d_2\ \mathrm{RLIM}(dt_1, ci_1, dt_2, ci_2, r) \wedge \mathrm{DocumentArtifact}(dt_1, d_1) \wedge \mathrm{DocumentArtifact}(dt_2, d_2) \rightarrow \mathrm{RL}(dt_1, d_1, ci_1, dt_2, d_2, ci_2, r)$$

(dt: Document Artifact Type, ci: Configuration Item, r: Releavnce Rule, d: Document Artifact)

수식 (1)의 술어 RLIM 은 RLIM 의 5 가지 요소를 튜플로 갖는 술어이며, DocumentArtifact 술어는 연구 주기 내에서 산출된 연구 문서 산출물의 문서 종류와 문서 식별자를 튜플로 갖는 술어이다. 전문가 시스템의 지식 베이스에 RLIM 과 RLIM 의 두 형상 항목(대응 항목)인 DocumentArtifact 술어가 사실로서 존재할 경우 수식 (1)의 규칙이 점화되어 새로운 지식(RL)이 추론되게 된다. 연관성 링크는

RLIM 을 특수화한 인스턴스로서 기준 문서 종류, 기준 문서 식별자, 기준 형상 항목, 비교 문서 종류, 비교 문서 식별자, 비교 형상 항목 그리고 두 형상 항목(대응 항목)이 준수해야 하는 연관성 규칙 등의 7 요소를 튜플로 갖는 술어이다.

연관성 링크 생성부는 일반화된 RLIM 을 사용하였을 때 문제가 될 수 있는 개별 사실(RL)들에 대한 접근을 가능하게 한다. 이와 더불어 전문가 시스템이 갖고 있는 장점 중의 하나인 지식 베이스의 확장을 이용하여 *요구사항 3* (불필요한 연관성 링크 생성을 최소화)을 만족시킬 수 있다.

4.4. 연관성 규칙 검증부

연관성 규칙 검증부에서는 지식 베이스에 존재하는 모든 연관성 링크 내 형상 항목들이 연관성 규칙을 준수하고 있는지를 검증하는 역할을 수행한다. 그림 7 은 연관성 규칙을 준수하지 않는 연관성 링크를 도식화한 그림이다.

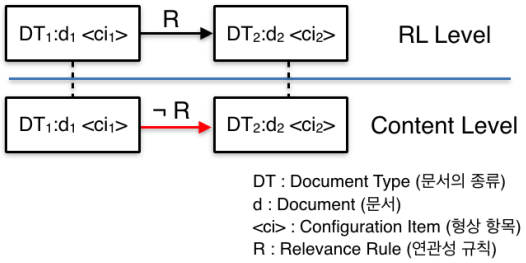


그림 7 연관성 규칙 위배 결점

그림 7 에서 RL level 은 연관성 링크 레벨로서 지식 베이스에 존재하는 연관성 링크를 나타내며, Content level 은 내용 레벨로서 연구 주기에서 실제로 산출된 연구 문서 산출물을 의미한다. 따라서, RL level 에서 정의된 연관성 링크의 두 형상 항목은 Content level 에서의 실제 연구 문서 산출물들 내에서 작성되어 있어야 하며, RL level 에서 정의된 연관성 규칙은 Content level 에서의 실제 연구 문서 산출물에서 준수되어야 한다. 만약 Content level 의 실제 연구 문서 산출물 내 형상 항목들이 연관성 규칙을 만족하지 않는다면 이는 계획된 추적을 수행할 수 없음을 의미하며, 사용자에게 이를 인지시켜 수정을 유도해야 한다. 이러한 검증 과정은 Content level 의 두 형상 항목과 연관성 규칙을 사용자에게 보여주어 수동적으로 검증할 수도 있으며, 텍스트 추론 특히 텍스트 함의 인식, 패러프레이즈 식별 등의 고급 기법을 사용하여 자동으로 검증할 수도 있다[20]. 본 논문에서 제시하는 전문가 시스템은 이러한 과정을 통해서 기존의 연구들이 수행하지 못하였던 소프트웨어 연구 문서 산출물을 추적함으로써 소프트웨어의 품질 향상을 도모한다.

5. 전문가 시스템 구현

본 논문에서 제안한 전문가 시스템의 타당성을 확인하기 위해서 예비 실험을 진행하였다. 예비 실험을 위해 사용된 소프트웨어 연구 문서 산출물의 종류는 연구 계획서와 연차 실적계획서이며, 연차 실적계획서는 1~4 차년도 실적계획서를 사용하였기 때문에 연구 문서 산출물의 인스턴스는 총 5 개를 사용하였다. 그림 8 는 예비 실험에서 사용되었던 연구 문서 산출물의 연관성 링크 그래프이다

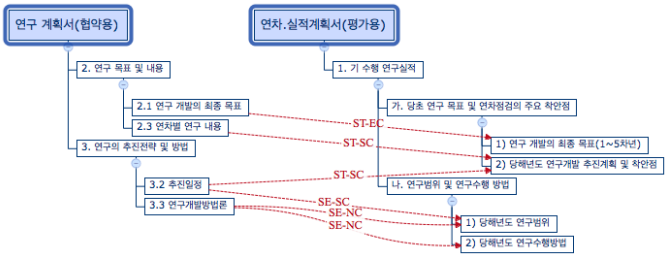


그림 8 연구 문서 산출물의 연관성 링크 그래프

5.1. 실험 환경

전문가 시스템의 예비 실험 구현을 위해서 java 기반의 Jess rule engine library 를 사용하여 전문가 시스템을 구성하였다. 한글(.hwp) 포멧의 연구 문서 산출물을 파싱하기 위해서 H2TLib [21] 를 사용하였다. 연관성 규칙 검증의 자동화를 위해서 꼬꼬마 분석기[22]를 사용하여 형태소를 분석한 뒤, R 환경을 이용하여 백터 공간 모델에서의 코사인 유사도를 확인하였다. 하지만 본 논문에서 제시하는 연관성 규칙 중 대부분은 연구자들에게 질의를 통한 수동적인 방법을 통해서 검증을 진행하였는데, 이러한 이유는 한글을 대상으로 하는 텍스트 함의 인식에 대한 참조 연구가 부족하였기 때문이다.

5.2. 구현

전문가 시스템을 구성하기 위해서 지식 베이스와 규칙을 구현하였다. 또한, R 환경과의 데이터 처리를 위해 XML 형식을 사용하였다.

지식 베이스의 경우, 전문가 시스템의 입력에서 'RLIM' 형태의 지식을 추가하였고, 연관성 링크 생성부에서는 RLIM 을 토대로 추론된 '연관성 링크' 형태의 지식을 지식 베이스의 확장을 위해 사용하였다. 또한, 연구 문서 산출물의 각 형상 항목을 '형상 항목' 형태로 정의하여 지식 베이스에 추가하였다. 마지막으로, 모델 검증부와 연관성 규칙 검증부에서는 결점 및 검증의 결과를 나타내는 지식을 정의하여 사용하였다.

규칙의 경우, 4 장의 세부 절에서 설명한 바와 같이

모델 검증을 위한 규칙, 연관성 링크 생성을 위한 규칙, 연관성 규칙 검증을 위한 규칙 등을 전문가 시스템에 정의하였다. 그림 9 는 이러한 규칙 중 하나인 이중 연관성 링크 결점 식별을 위한 규칙이다.

```
(defrule duplicated-relevance-link-problem ?f1 <- (RLIM
(BasicDocumentArtifactType ?x1) (BasicConfigurationItem ?x2)
(ComparedDocumentArtifactType ?x3)
(ComparedConfigurationItem ?x4) (RelevanceRule ?x5)) ?f2 <-
(RLIM (BasicDocumentArtifactType ?x1)
(BasicConfigurationItem ?x2) (ComparedDocumentArtifactType ?x3)
(ComparedConfigurationItem ?x4) (RelevanceRule ~?x5)) =>
(printout t ?f1.RelevanceRule " of " ?f1 " and " ?f2.RelevanceRule "
of " ?f2 " have a duplicated relevance link problem" crlf) (assert
(SystemError TRUE)))
```

그림 9 이중 연관성 링크 결점 식별을 위한 규칙

이중 연관성 링크 결점은 지식 베이스 내에 존재하는 RLIM 중에서 RLIM 의 RelevanceRule 값만이 서로 다른 두 RLIM 이 존재하는 것을 점화 조건으로 갖고 있으며, 조건을 만족할 시에는 이들의 내용을 결과화면에 알리고, (SystemError TURE) 지식을 지식 베이스에 추가하여 사용자에게 오류를 알린다.

지면의 한계를 이유로 지식 및 규칙의 세부적인 설명은 생략하나 모든 지식 및 규칙은 위와 같은 방식을 통해 전문가 시스템 내에 구현되었다.

5.3. 실험 결과

예비 실험을 위해 구현된 전문가 시스템에 총 6 개의 RLIM 과 5 개의 연구 문서 산출물을 입력하였다. 그림 10 은 예비실험 과정을 도식화한 그림이다. 모델 검증부를 통해 RLIM 을 검증하여, 본 예비 실험에서 사용되는 모든 RLIM 의 무결성을 확인하였다. 추가적으로 모델 검증부의 동작을 확인하기 위해서 임의의 결함을 지니고 있는 RLIM 을 입력하여 모델 검증부가 동작함을 확인해보았다. 그림 11 은 본 논문에서 예로 들었던 이중 연관성 링크 결점(Duplicated Relevance Link Problem)을 갖고 있는 RLIM 한 쌍이 지식 베이스에 존재하여 규칙이 점화된 결과화면이다.

그림 11 에서 보이는 바와 같이 <Fact-1>에서는 RLIM 이 ST-EC 관계의 연관성 규칙을 갖고 있는 동시에 <Fact-7>에서는 ST-SC 관계의 연관성 규칙을 갖고 있기 때문에 이중 연관성 링크 결점 식별을 위한 규칙이 점화되고, 이를 사용자에게 알린다.

모델 검증부를 통해 RLIM 의 무결성이 검증된 이후에는 연관성 링크 생성부에서 연관성 링크가 생성된다. 본 예비 실험에서는 6 개의 RLIM 이 4 쌍의 산출물(연구 계획서와 1~4 차년도 실적 계획서)을 대상으로 연관성 링크 생성을 하기 때문에 총 24 개의 연관성 링크가 생성되어야 했다. JESS 의 기본 질의문을 활용하여 지식베이스 내의 연관성 링크가 총 24 개 생성된 것을 확인하였으며, 연관성 링크 내부를 확인하여 그 내용 또한 문제가 없음을 확인하였다.

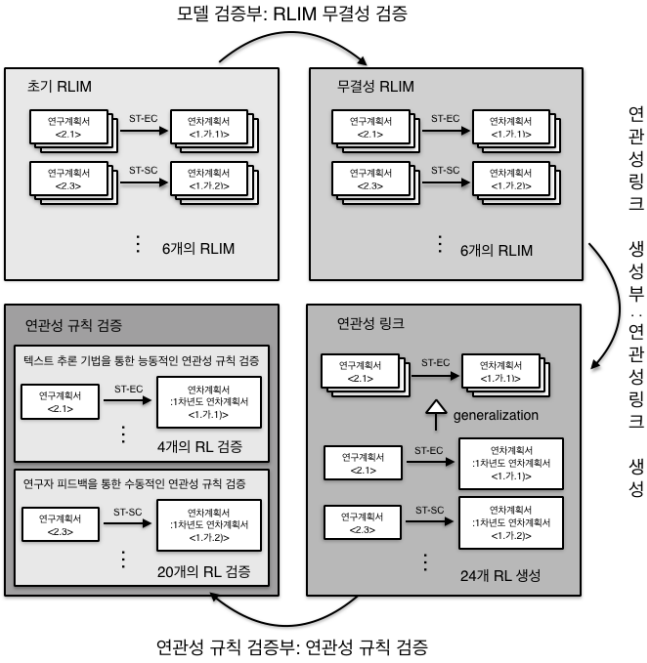


그림 10 예비 실험 과정

```
---------------------------------- JESS RESULT ----------------------------------
ST-SC of <Fact-7> and ST-EC of <Fact-1> are duplicated relevance link problem
ST-EC of <Fact-1> and ST-SC of <Fact-7> are duplicated relevance link problem
2
---------------------------------------------------------------------------------
```

그림 11 이중 연관성 링크 결점 식별을 위한 규칙 점화 결과화면

마지막으로 연관성 규칙 검증부에서는 24 개의 연관성 링크 각각이 연관성 규칙을 준수하고 있는지를 확인한다. 연관성 규칙 검증을 위해서는 연구자의 수동적인 평가를 이용할 수 있으며, 텍스트 추론 기법을 통한 자동화 방법을 사용할 수도 있다. 본 예비실험에서는 24 개의 연관성 링크 중 ST-EC 연관성 규칙을 갖는 4 개의 연관성 링크는 코사인 유사도를 통해 검증을 수행하였으며, 나머지 20 개의 연관성 링크는 사용자의 수동적인 평가를 통해 검증하였다. 실험 결과 총 24 개의 연관성 링크 내 연관성 규칙이 잘 성립함을 확인하였으며, 예비 실험을 통해 본 논문에서 제안하는 전문가 시스템이 소프트웨어 연구 문서 산출물을 대상으로 추적 가능함을 확인 할 수 있었다.

5. 결론 및 향후 계획

본 논문은 소프트웨어 연구 주기에서 산출되는 산출물 가운데 가장 보편적이고, 일반적으로 사용되고 있는 종류인 소프트웨어 연구 문서

산출물에 대한 추적을 지원하기 위해서 전문가 시스템을 제안하였다. 본 논문에서 제안하는 전문가 시스템은 모델 체크부에서 RLIM 의 무결성을 확인하고, 연관성 링크 생성부에서 연관성 링크를 생성하며, 연관성 규칙 검증부에서 모든 연관성 링크에 대응 되는 연구 문서 산출물 내 형상 항목의 내용이 연관성 규칙을 준수하고 있는지를 확인하여 연관성 규칙을 검증한다. 본 논문에서 제안하는 전문가 시스템은 이러한 과정을 통해 연구 문서 산출물에 대한 추적을 지원하며, 문제 발생시 이를 사용자에게 알려 소프트웨어 연구 주기를 관리 가능하게 하여 소프트웨어 품질 향상에 기여한다.

향후 보다 선진화된 텍스트 추론 기술을 본 논문에서 제안하는 전문가 시스템의 연관성 규칙 검증부에 적용하여 전문가 시스템의 전 과정을 자동화할 계획이다.

참고문헌

[1] Abran, Alain, et al. Guide to the software engineering body of knowledge-SWEBOK. IEEE Press, pp.1-1, 2004.

[2] Software Engineering Technical Committee. "IEEE Standard for Software Test. IEEE Standard IEEE Std 829-1998." IEEE Computer Society 345: 10017-2394.

[3] 한국정보통신기술협회. “소프트웨어 형상관리 계획 표준 TTA.IE-828.”

[4] Estublier, Jacky. "Software configuration management: a roadmap." Proceedings of the Conference on the Future of Software Engineering. ACM, 2000.

[5] 김대엽, and 윤청. "객체지향 개발에서의 효율적인 변경 관리를 위한 추적성 관리 및 영향 분석 방법." 정보과학회논문지 42.3 (2015): 328-340.

[6] Gotel, Orlena CZ, and Anthony CW Finkelstein. "An analysis of the requirements traceability problem." Requirements Engineering, 1994., Proceedings of the First International Conference on. IEEE, 1994.

[7] Cleland-Huang, Jane, et al. "Software traceability: trends and future directions." Proceedings of the on Future of Software Engineering. ACM, 2014.

[8] National Research Council (US). Committee for Advancing Software-Intensive Systems Producibility. Critical Code: Software Producibility for Defense. National Academies Press, 2010.

[9] Dart, Susan. "Concepts in configuration management systems." Proceedings of the 3rd international workshop on Software configuration management. ACM, 1991.

[10] Mader, Patrick, et al. "Strategic traceability for safety-critical projects." Software, IEEE 30.3 (2013): 58-66.

[11] Radatz, Jane, Myrna Olson, and Stuart Campbell. "Mil-std-498." Crosstalk, the Journal of Defense Software Engineering 8.2 (1995): 2-5.

[12] McMillan, Collin, Denys Poshyvanyk, and Meghan Revelle. "Combining textual and structural analysis of software artifacts for traceability link recovery." Traceability in Emerging Forms of Software Engineering, 2009. TEFSE'09. ICSE Workshop on. IEEE, 2009.

[13] Gotel, Orlena, et al. "The grand challenge of traceability (v1. 0)." Software and Systems Traceability. Springer London, 2012. 343-409.

[14] Merten, Thorsten, Daniela Jüppner, and Alexander Delater. "Improved representation of traceability links in requirements engineering knowledge using Sunburst and Netmap visualizations." Managing Requirements Knowledge (MARK), 2011 Fourth International Workshop on. IEEE, 2011.

[15] Shin, Yonghee, and Jane Cleland-Huang. "A comparative evaluation of two user feedback techniques for requirements trace retrieval." Proceedings of the 27th Annual ACM Symposium on Applied Computing. ACM, 2012.

[16] 백두산, and 이정원. "SW 연구 개발 문서 산출물 간의 대응 항목 추적을 위한 연관성 분석." KCSE 2015 논문집 17 권 1 호, (2015): 13~20.

[17] Marcus, Andrian, Xinrong Xie, and Denys Poshyvanyk. "When and how to visualize traceability links?." Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering. ACM, 2005.

[18] Mader, Patrick, et al. "Strategic traceability for safety-critical projects." Software, IEEE 30.3 (2013): 58-66.

[19] 이재규, et al. "전문가 시스템-원리와 개발." (1996).

[20] 최성필, et al. "텍스트 추론 (Textual Inference) 연구 동향 분석." 정보과학회지 30.11 (2012): 68-77.

[21] H2TLib-Hwp to Text. “https://sites.google.com/site/h2tlib/”

[22] 이동주, et al. "꼬꼬마: 관계형 데이터베이스를 활용한 세종 말뭉치 활용 도구." 정보과학회논문지: 컴퓨팅의 실제 및 레터 16.11 (2010): 1046-1050.

위기대응 매뉴얼 신뢰성 향상을 위한 검증방안 연구

*이 혁, **최진영

*고려대학교 컴퓨터전파통신공학부, **고려대학교 정보보호대학원
서울특별시 성북구 안암로 145
{hlee, choi}@formal.korea.ac.kr

요약: 사고란 원하지 않은 사건들이 발생하는 것으로 인적 또는 재산의 피해를 초래할 수 있다. 이런 사고의 원인으로는 기술적 결함, 인간 결함, 조직/체계의 구조적 결함 등으로 발생할 수 있다. 사고가 발생했을 때 적절한 대응은 그 피해를 최소화 시킬 수도 있지만, 잘못된 대응으로 상황을 악화시킬 수도 있다. 본 논문에서는 위기대응 매뉴얼에 대한 검증방안을 제시함으로 대응매뉴얼의 모호성, 불일치성을 제거하여 매뉴얼의 신뢰성을 향상시키고자 하였다.

핵심어: 위기대응 매뉴얼, 정형명세, 정형검증

1. 서론

'사고란 원하지 않은 사건들이 발생'하는 것으로 인적 또는 재산의 피해를 초래할 수도 있다. 이런 사고에 대한 인식은 시대가 흐름에 따라 사회적인 분위기에 의해 변화하였다. 1970 년 말까지 사고에 대한 인식은 사고가 발생하면 운영자 혹은 담당자에게 모든 책임을 묻는 개인적 접근방식(Person Approach)를 띄고 있었다. 하지만, 1980 년쯤부터 사회학자들은 사고 발생의 또 다른 원인을 사회의 구조적인 문제로 꼽으면서 개인의 결함행위를 사전에 방지하지 못하는 체계 및 제도에 문제를 제기하는 시스템적 접근방식(System Approach)을 보이고 있다.[1]

시스템의 복잡도 및 결합도가 높아짐에 따라 대형화된 시스템에서의 사고를 막는 것에 사실상 한계가 있음을 말하고 있는 정상사고이론(Normal Accident Theory)에서는 이러한 문제를 해결하기 위해서 시스템을 단순화 시키고 결합도를 낮추는 것을 권고하고 있다.[2]

시스템에서 소프트웨어의 사용이 매우 증가함에 따라 안전기능과 같은 부분이 소프트웨어로 대체되고 있으며, 이러한 부분에서 소프트웨어의 실패는 사고로 이어질 수 있는 치명적인 원인이 되기도 한다. 뿐만 아니라, 시스템적 접근방식에서 사고의 요인을 줄이기 위해서 절차적인 측면에 대한 검증을 수행함으로 전반적인 시스템의 신뢰성을 향상시킬 수도 있다.[3][4]

시스템의 운영매뉴얼 이나 비상시 대응매뉴얼은 관련 도메인 전문가들에 의해 작성된다. 전문가들은 여러 상황들은 예측 및 분석하여 적절한 매뉴얼을 작성하게 되는데, 문서화되는 일반적인 지침은 전문가(사람)에 의해 자연어로 기술된다. 사람의 힘으로 모든 조건의 조합을 확인하는 것은 불가능하며, 이는 논리적 모순이 발생할 수 있는 여지를 남기게 된다. 따라서, 전문가들에 의해 작성된 운영매뉴얼 또는 대응매뉴얼에 절차적 모순점이 존재할 수 있다.

본 논문에서는 사고 발생시 대처해야 할 행위들을 기술한 대응 매뉴얼에 정형명세 및 정형검증을 통해 모호성, 무결성을 찾아내어 매뉴얼 전반의 신뢰성을 향상시킬 수 있는 방안을 제시하였다. 2 절에서는 신뢰성 향상을 위해 사용한 정형기법에 대해 간략히 살펴보고 3 절에서는 명세 및 검증의 대상인 대응매뉴얼에 대해서 설명한다. 그리고 4 절에서는 모델에 대해 설명한 뒤 5 절을 결론에 대해 설명한다.

2. 정형기법

정형기법은 소프트웨어 및 하드웨어 시스템을 명세, 설계, 개발, 검증하기 위한 수리 논리 기반 기술로 개발 절차 중 오류 생성을 줄여 시스템의 신뢰성을 향상시킬 수 있다.[5]

- 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성 지원사업의 연구결과로 수행되었음" (IITP-2015-H8501-15-1012)

2.1 정형명세

정형명세는 수학적으로 정의된 문법과 의미를 가진 언어를 사용하여 추상화를 통해 시스템 및 시스템이 만족해야 할 속성들을 표현한다. 정형명세는 문서화, 설계단계, 확인, 의사소통 등의 소프트웨어 개발 활동에 사용되어 품질 및 신뢰성 향상에 도움이 된다.

2.2 정형검증

정형검증은 정형명세 된 시스템이 지켜야 할 속성을 만족하는지 분석하는 기법으로, 크게 모델체킹[6]와 정리증명[7]로 나눌 수 있다. 모델체킹은 명세로부터 생성된 모델과 시스템이 만족해야 할 속성을 만족하는지 확인한다. 모델체킹은 시스템 모델의 모든 상태를 탐색하며 주어진 속성 만족 여부를 확인하고, 만족하지 않을 시 반례를 제공한다. 모델체킹은 자동화의 이점이 있으나 모델의 크기가 커지면 상태 폭발 문제가 발생할 수 있다. 정리증명은 시스템과 증명하고자 하는 속성을 수학적 논리에 기반을 둔 논리식으로 표현한다. 표현된 속성들은 정형 시스템의 공리와 추론규칙들을 통해 증명된다. 정리증명은 무한 상태 공간을 처리할 수 있는 장점이 있으나 완전 자동화가 불가능한 단점이 있다.

3. 대응매뉴얼

모든 조직에서는 사고 발생시 원만한 대처를 위해 사고대응매뉴얼을 가지고 있다. 이것은 사고 발생 시 각 담당자들이 반드시 수행해야 할 임무들과 대처방안들을 기술하고 있다. 자연/인적재해들은 예고 없이 찾아올 수 있기 때문에 사고 발생 시 원만한 대처를 위해서는 대응매뉴얼에 대한 주기적인 훈련을 통해 위기 상황을 대비해야 한다. 하지만, 분야별 전문가들에 의해 작성된 대응매뉴얼일지라도 자연어로 작성되며 사람을 통한 검토에 의존하기 때문에 모든 가능성에 대해서 완벽할 수는 없다. 지하철 화재 대응매뉴얼 분석을 통해 절차적 모순을 찾아보고자 한다.

3.1 운전자 매뉴얼

지하철 화재 발생시 운전자의 대응매뉴얼을 간략히 요약해보면 아래와 같다.

표 1 화재 발생시 운전자 매뉴얼

- o 화재 /가스 누출 인지 → 상황 파악 후 운영팀 통보
- o 폭발 가능성, 가스 유독성, 이동거리 등을 고려 → 열차 이동여부 결정
- o 선로에 정차 할 경우 반대편 선로 및 연기방향 확인/예측
- o 열차문 개방 및 승객 대피
- o 플랫폼에 정차 할 경우 플랫폼까지 이동
- o 열차문 개방 및 승객 대피
- o 승무원 탈출 시 안전상의 이유로 마스터키 제거

화재 발생시 운전자의 최종목표는 승객을 안전하게 대피시키는 것이다. 이를 위해서 운전자는 열차를 승객을 대피시키기 안전하다고 판단되는 장소로 이동하여 승객을 대피시킨다. 또한, 운전자는 마스터키를 열차에 방치해서는 안된다.

지하철의 열차는 기본적으로 운전자의 조작으로 동작하며 여러 가지 운전모드를 통해 자동/반자동/수동으로 조작이 가능하다. 열차의 동작 방식은 아래와 같다.

표 2 열차 시스템 동작방식

- o 정지 명령 → 열차 정지
- o 이동 명령 → 열차 운행
- o 열차 정지상태 → 문열림 버튼 → 출입문 열림
- o 문닫힘 버튼 → 출입문 닫힘
- o 마스터키 제거 → 열차 정지 / 출입문 닫힘
- o 열차 이동 중 → 비상 문열림 버튼 → 열차 정지

기본적인 열차의 동작은 운전자의 조작에 의해 이루어 지지만, 일반적인 운행 중에 출입문 개방이 불가하나 출입문이 개방되었을 경우 열차가 비상 정지하는 것과 같은 안전 인터락킹 시스템(Safety Interlocking system)이 적용되어 있다. 표 2 에서 간략하게 나타낸 지하철의 동작과는 달리 실제 지하철 시스템은 운영스케줄, 운영모드, 신호체계 및 다량의 열차로 구성된 상당히 복잡한 운영체계를 가지고 있다. 본 논문에서 이에 대한 부분은 적용범위를 벗어나므로, 운전자의 대응절차를 중점을 두고 매뉴얼에 대한 모델링을 진행하였다.

4. 매뉴얼 모델링

4.1 운전자 매뉴얼

운전자, 열차, 승객, 운영팀의 행위를 추상화하여 각각 프로세스로 표현하였다. 아래 그림은 열차의 이동 및 출입문 개폐에 대한 행위를 나타낸 상태 다이어그램이다.

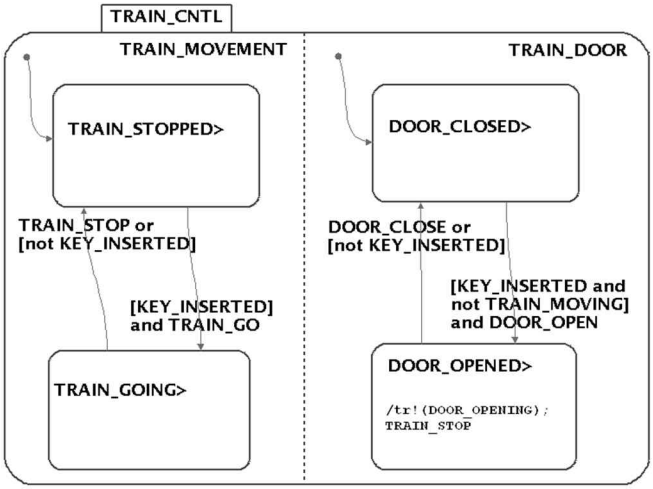


그림 1 열차 이동 및 출입문 상태 다이어그램

열차의 이동은 '이동중'인 상태와 '정지중'인 상태로 나타내었으며, 출입문의 개폐상태도 '열린'상태와 '닫힌'상태로 나타내었다. 동작매뉴얼 상의 제한사항(Constraints)인 열차가 이동 중에 출입문이 열리면 열차는 비상 정지하는 부분도 이벤트를 통해 표현하였다.

매뉴얼검증을 위한 열차 운행 모델을 NuSMV 로 표현하면 표 3 과 같다.

표 3 NuSMV 로 표현한 열차운행 모델

```
MODULE train(dr_opened_msg)
   VAR
      state : {tr_stopped, tr_going};
      transition : {tr_go, tr_stop};
ASSIGN
   init(state) := tr_stopped;
   next(state) :=
      case
         state = tr_stopped & transition =
tr_go : tr_going;
         state = tr_going & transition =
tr_stop : tr_stopped;
         TRUE : state;
      esac;
   next(transition) :=
      case
         state = tr_stopped & !dr_opened_msg
      : tr_go;

…
```

열차/출입문/승객/비상상황을 프로세스로 표현하여 행위를 모델링 하였다. 명세 된 모델에 대해서 검증을 원하는 속성은 표 4 와 같다.

표 4 검증을 위한 LTL 속성

비상 발생시, 출입문이 개방되어 승객이 대피할 수 있어야 한다.

```
LTLSPEC G ! (proc1.state=tr_stopped &
proc2.state=dr_closed &
proc3.state=pss_onboard  &
proc4.state=emergency)
```

위의 LTL 표현식의 의미는, 비상상황 시 승객이 탑승한 채 열차 출입문이 닫혀있는 상태이며, 모델이 이러한 상태의 조합에 도달여부를 확인하여 나타낸다.

위의 검증속성에 대한 만족여부를 확인해본 결과 반례에서는 검증속성이 만족하지 않음을 보이고 있다. 검증을 통해 찾아낸 이러한 사항들은 운영매뉴얼의 신뢰성 향상을 위해 수정 및 보완되어야 할 것이다.

5. 결론

본 연구에서는 대응매뉴얼의 신뢰성을 높이기 위한 검증방안을 제안하였고, 사례연구로 화재매뉴얼 모델링하여 검증을 수행하였다.

대응매뉴얼과 같이 자연어로 작성된 절차서와 같은 경우, 여러 단계의 세분화된 추상화 과정을 거치면 완성도가 높은 모델 작성이 가능할 것으로 보인다.

또한, 다양한 모델링 언어와 명세방법을 통해 대응매뉴얼의 다양한 측면에 대한 접근이 가능할 것이다.

참고문헌

[1] Turner, Barry A., and Nick F. Pidgeon. Man-made disasters. Butterworth-Heinemann, 1997.

[2] Charles Perrow, "Normal Accidents : living with high-risk technologies", Princeton Univ., 1984.

[3] Sommerville, Ian. Software Engineering. Tenth ed. Boston: Pearson, 2016

[4] Christophe Damas, Bernard Lambeau, "Analyzing Critical Process Models through Behavior Model Synthesis", ICSE '09, May 16-24, 2009

[5] Edmund M. Clarke, Jeannette M. Wing, “Formal methods: state of the art and future directions”, ACM Computing Surveys (CSUR), 1996.

[6] Clarke, E. M. AND EMERSON, E. A. 1981. Synthesis of synchronization skeletons for branching time temporal lock. In Logic of Programs: Workshop,(Yorktown Heights, NY), Vol. 131 of Lecture Notes in Computer Science, Springer-Verlag.

[7] Kaufmann, Matt, J. Strother Moore, and Panagiotis Manolios. Computer-aided reasoning: an approach. Kluwer Academic Publishers, 2000.

국방 전투관리체계 개발을 위한 상호작용 유즈 케이스 기반 요구사항 모델링 기법

김두환[1], 홍장의[2], 김동환[3]
[1,2]충북대학교 컴퓨터과학과, 충북 청주시 서원구 충대로 1번지
[3]LIG 넥스원, 경기도 성남시 분당구 판교로 333번지
[1]dhkim@selab.cbnu.ac.kr, [2]jehong@chungbuk.ac.kr, [3]kimdonghwan@lignex1.com

요약: UML 기반의 객체지향 개발 방법론이 일반화되면서, 국방 임베디드 소프트웨어 개발 분야에서도 객체지향 개념을 적용한 소프트웨어 분석 및 설계가 확대되고 있다. 그러나 객체지향 개발 방법론이 초기 트랜잭션 중심 소프트웨어 시스템에 적합하도록 개발되어졌음에도 불구하고, 요구사항을 분석하는 초기 단계에 핵심적으로 고려되어야 하는 데이터 요구사항에 대하여 충분히 반영하지 못하는 어려움이 있다. 본 논문에서는 UML 기반 기능분석을 수행하기 위한 모델링 단계에서 효율적으로 데이터를 고려할 수 있도록 지원하는 Interactive Use Case 기반 요구사항 모델링 기법을 제시한다. 제시하는 기법은 국방 전투관리체계 소프트웨어 모델링에 있어서 초기단계에 보다 명확한 요구사항 표현이 가능하기 때문에 요구사항의 모호성을 줄이고, 변경의 가능성을 줄일 수 있다는 효과를 제공한다.

핵심어: 객체지향 분석, 요구사항 모델링, 데이터 기반 모델링, Interactive Use Case 모델링

1. 연구배경

객체지향 개념은 실 세계의 환경을 직관적으로 모델링할 수 있도록 국방분야의 전장 환경을 객체 단위로 표현하고자 하는 시뮬레이션 접근 방법에서 비롯되었다[1]. 개념의 우수성과 다양하고 융통성 있는 표현력으로 인하여 객체지향 개념은 일반적으로 소프트웨어 개발 영역으로 적용 범위가 확대되었고, 또한 임베디드 소프트웨어의 특성을 반영하기 위하여 UML 2.0 [2]이 제시되면서, 그 유용성이 더욱 높아지고 있다.

초기 UML 기반 객체지향 개발 방법론은 방법론 자체가 갖는 직관성과 비정형성으로 인하여 – 물론 일부는 준정형성 (Semi-formalism)을 갖기도 하지만 – UML 다이어그램과 Z [3], CSP [4] 등의 정형 기법을 통합하여 다양한 특성의 시스템을 모델링 하고자 하는 시도가 있어 왔다. 그러나 이러한 통합은 적용 분야가 한정되고, 또한 전문적인 모델링 능력을 요구하고 있기 때문에 활용 범위가 제한되었으며, 일반적인 국방 임베디드 소프트웨어 또는 지휘통제통신 소프트웨어 개발에 적용하기에는 쉽지 않다는 문제가 있었다[5].

본 연구는 국방 전투관리체계 소프트웨어 개발에 있어서 UML 기반의 객체지향방법론이 갖는 부족한 점을 보완하기 위하여 객체지향 모델링 방법에 구조적 분석 기법[6]의 특성을 반영한 요구사항 모델링 기법을 제시한다. 제시하는 기법은 특히 국방 소프트웨어 시스템의 요구사항 분석 단계에서 중요하게 여겨지는 핵심 데이터의 도출 및 식별활동을 포함하도록 고안되었다. 본 연구의 동기는 일반적으로 객체지향의 분석 단계에서 Use Case 명세서를 작성하는 세부적인 기능 식별 과정을 수행함에도 불구하고, 데이터에 대한 별도의 명세 및 관리가 이루어지고 있지 않는다는 것에 있다. 이로 인하여 추후 데이터 식별 과정을 설계 과정에서 별도로 진행해야 하는 부담을 갖게 되며, 또한 데이터 분석 시점에서 소프트웨어 기능 관점의 데이터 설계로 인하여 사용자 관점에서의 데이터 중요도가 낮아질 수 있다는 문제점이 발생할 수 있다.

따라서 본 연구에서 제시하는 Interactive Use Case (IUC)기반 요구사항 모델링 기법은 기존의 객체지향 기반 Use Case 모델링 방법을 확장한 것으로써, 소프트웨어 기능적 모델링을 위해 개발되는 Use Case 다이어그램에서부터 시스템의 핵심 데이터를 Use Case와 연결시키는 Interactive Use Case 개념을 제시하였다. 또한 이러한 개념을 기반으로 요구사항을 분석하고 명세하기 위한 절차를 제시하였다.

제시된 방법을 통해, 국방 전투관리체계 소프트웨어를 모델링할 때, 요구사항의 초기 분석단계에서부터 핵심 데이터를 고려한 소프트웨어 개발을 가능하게 함으로써, 데이터 중심의 국방 정보시스템의 개발에 유용하게 적용될 수 있을 것이다.

2. 관련연구 분석

기존의 UP(Unified Process) 기반의 객체지향 모델링 과정[7]은 주어진 사용자 요구사항을 기반으로 기

능 모델링을 수행한다. 기능 모델링의 기본적인 산출물은 Use Case 다이어그램으로써, 시스템과 시스템의 외부에 존재하는 액터(Actor)간의 상호 작용을 외부의 사용자 관점에서 표현한다[8]. 이러한 모델링에서는 Use Case 를 비즈니스의 기본적인 활동(Elementary Business Activity)으로 정의하고 있으며, 핵심 Use Case 는 최소한 하나 이상의 액터와 관계를 갖도록 정의하고 있다.

데이터 관점에서 요구사항을 모델링하기 위한 기존의 연구에는 CRUD Use Case [9,10]에 대한 연구가 있다. 이 연구는 데이터를 공동 처리하는 트랜잭션을 하나의 Use Case 로 도출하는 방법을 제안하는 것으로써, 데이터 생성(Create), 읽기(Read), 갱신(Update) 그리고 삭제(Delete)를 수행하는 트랜잭션을 묶어 하나의 Use Case 로 정의한다. 이렇게 함으로써 데이터의 활용 관점을 초점화하여 시스템을 표현할 수 있다는 장점을 제공하지만, 시스템 구성 기능간의 모듈화 척도를 향상시키기는 쉽지 않을 수 있다.

Use Case 기반 요구사항 분석 및 명세와 관련된 또 다른 연구는 4 단계로 Use Case 의 수준을 명세하는 방법[11]이다. 이 연구에서는 Use Case 를 (1) 작업 영역(Work Division), (2) 응용 중심 시스템 기능(Application-specific system Functions), (3) 작업 중심 시스템 기능(Work-specific system Functions) 그리고 (4) 대화 (Dialogue)의 4 수준에서 명세하였다. 이중에서 "응용 중심 시스템 기능" 명세에서는 시스템이 사용하는 데이터를 중심으로 Use Case 를 표현하고 있으며, 특히 이를 위하여 구조적 분석 기법에서 사용하는 자료 흐름도(Data Flow Diagram)을 이용하여 데이터 정보를 명세하도록 하였다.

이외에도 요구사항을 명세하기 위한 방법으로 다양한 방법들이 제시되고 있지만, 객체지향 개념이 초기 요구사항 명세를 위하여 사용자의 기능 모델링을 기반으로 출발한다는 관점에서 데이터의 중요성을 크게 강조하지는 못하고 있다.

3. IUC 기반 요구사항 모델링

3.1 모델링의 기본 개념

일반적으로 소프트웨어 개발을 위한 방법론을 정의하기 위해서는 다음과 같은 4 가지 구성요소가 정의되어야 한다.

- 적용 개념(Applicable Concepts): 소프트웨어 모델링을 위해서는 제공하는 방법론이 어떤 도메인 특성을 지원하며, 이를 위하여 어떠한 공학적 개념을 지원하는가? 하는 것을 의미한다. 특히 방법론의 적용과정에서 일관성 있게 지원하는 기본적인 Building Block 은 무엇인지 제시되어야 한다.
- 절차 (Procedure): 절차는 요구사항을 분석하고 설계하는 세부적인 일련의 활동들로 정의된다. 이러한 절차에서는 이를 구성하는 활동의 목적이 무엇이며, 이를 통해 어떠한 산출물을 얻을 수 있는가가 정의된다. 또한 산출물을 작성하기 위한 방법 및 기법이 제공되어야 한다.
- 용어 및 표기법(Notation and Terminology): 방법론은 요구사항에 대한 분석과 설계 과정에서 작성되는 다이어그램 또는 모델에 대한 그래픽 표현을 위하여 적절한 의미를 갖는 심볼을 정의하고 이에 대한 정확한 시맨틱을 제공해야 한다. 또한 방법론에서 사용하는 용어도 정의하며, 이를 사용자와의 소통 수단으로 활용해야 한다.
- 평가 기준 (Evaluation Criteria): 모델링 절차에 의해 생성된 모델 및 산출물들은 상호 참조 및 활용을 위하여 사용될 수 있는 적절한 가이드라인이나 모델 요소 또는 속성들에 대한 품질 평가를 위한 기준들이 제시될 수 있어야 한다.

본 연구에서 제시하는 IUC 기반의 요구사항 모델링 기법에서는 전술한 4 가지 구성요소에 대하여 다음과 같은 방법으로 디자인 되었다.

[적용 개념] 본 연구에서 제안하는 모델링 방법은 기본적으로 국방 전투관리체계와 같이 기반 데이터를 사용하여 의사결정, 시뮬레이션 등과 같은 다양한 응용에서 활용될 수 있는 체계를 표현하기에 적합하도록 제시되었다.

특히 기존의 UML 표기법을 근간으로 하고는 있지만, 일반적으로 객체지향 개발 방법론과 같이 객체/클래스를 중심으로 소프트웨어를 개발하는 접근 방법을 계승하지는 않았다. 다만 기능적 모델링을 위해 사용하는 Use Case 개념을 사용하였으며, 특별히 데이터의 특성을 표현하기 위해 확장되었다. 그림 1 은 Use Case 간에 자료 저장소를 통해 상호 작용하는 IUC 를 포함한 Use Case 다이어그램의 예를 나타낸다.

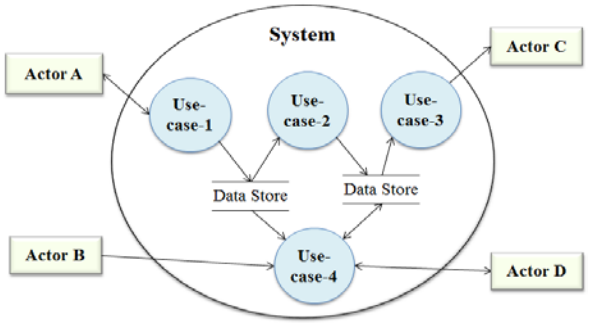


그림 1 IUC 를 포함하는 Use Case 다이어그램 예

본 논문에서 제시하는 IUC 는 사용자의 관점에서 시스템의 상위수준 기능과 기능간에 존재하는 상호작용의 관점을 표현하는 논리적인 모델 요소로 정의할 수 있다.

[절차] 본 논문에서 제시하는 모델링 절차는 3.2 절에서 상세히 설명할 것이다. 다만 제시하는 모델링 방법은 다음과 같이 크게 4 개의 단계(Step)로 구성되었다.

- Step 1: 시스템 경계(Boundary)를 결정한다.
- Step 2: 시스템 수준의 Use Case 를 식별하고 다이어그램을 완성한다.
- Step 3: 시스템 수준의 자료 사전을 작성한다.
- Step 4: Use Case 명세서를 작성한다.

[용어 및 표기법] IUC 기반 요구사항 모델링을 위하여 앞서 설명한 바와 같이 UML 의 Use Case 다이어그램이 갖는 표기법에 기존의 구조적 방법론에서 사용되었던 몇 가지 표기법을 추가하였다. 확장된 표기법에 대하여 표 1 과 같이 정리하였다.

표 1 IUC 기반 요구사항 모델링의 표기법

Name	Symbol	Meanings
Navigation relationship	⟶	One directional interaction
	⟷	Bi-directional interaction
Data store	═══	Storage to contain data
Label	label ⟶	Transferring data between actor and use case
Transaction	⟶•⟶•⟶	Process flow for the instance of a use case

표 1 에서 정의하는 표기법 중에서 Data store 에 저장되는 데이터는 IUC 와 연관된 시스템 데이터를 표현하기 위한 정보들로써, 구조적 방법론에서 제시하는 자료 사전의 기술 방법[12]을 지원하는데, 이에 근거한 자료의 표현 방법은 다음과 같다.

A = a + b + c	A is composed of a, b and c data item
$A = {}_n\{a + b + c\}_m$	A is composed of a set of a, b and c data item n : minimum occurrence (default : 0) m : maximum occurrence (default : ∞)
A = {a + (b) + c}	b is optional
A = {a + [b | c]}	b and c are selectable
A = ["a" | "b" | "c"]	A means primitive data element having the values "a", "b", or "c"
A = * ----- *	---- is the comment of A

또한 표 1 에 나타난 Transaction 은 IUC 를 포함하는 Use Case 다이어그램의 인스턴스(instance)를 표현하기 위해 사용된다. Use case 의 인스턴스는 일반적으로 시나리오로 간주되는 것으로써, 액터(Actor)로부터 생성되는 이벤트에 의해 트리거되는 시스템의 행위를 표현하는 것으로써, 이벤트에 의해 시스템으로 입력되는 자극(Stimulus)에 의해 시스템 내부에서 발생하는 상태 변화가 존재함을 의미한다. 즉 자극으로 입력되는 정보는 상태 변화를 거치면서 전체적인 시스템의 상태를 결정하게 되며, 필요시 이러한 상태 정보는 Data store 에 저장되게 된다.

[가이드라인 및 평가 기준] 평가 기준은 앞서 설명한 것처럼 모델링 가이드라인이나 모델링 과정에서 준수되어야 하는 규칙들을 포함한다. 본 연구에서 제시하는 IUC 기반 모델링 방법의 설계 규칙을 다음과 같이 정의하였다.

- 모든 Use Case 는 적어도 하나의 Data store 와 연결되어야 한다. 임의의 Use case 가 갖는 기능을 수행하기 위해서는 시스템의 현재 상태에 기반한 기능의 동작 모드가 결정되기 때문이다.
- 액터와 Use Case 간에는 반드시 시스템의 행위를 트리거하기 위한 데이터 또는 시그널이 전달되어야 한다.
- Data store, 입력(Stimulus), 출력(Response)에 표현되는 데이터들은 반드시 자료 사전을 기술 방법에 의거하여 정의되어야 하며, 이들은 일반적으로 인지 가능한 자료 형이 도출될 때까지 계층적으로 분할되어야 한다.
- 각 IUC 는 하나의 Use Case 명세서에 내부의 행위를 단계적으로 기술해야 한다. 이러한 Use Case 명세에 있어서는 기존의 객체지향 방법에서 정의된 명세 규칙을 준수하여야 한다.
- Use Case 명세서 작성시에는 자료 사전에 정의된 데이터 항목의 명칭과 일관성을 유지해야 한다.
- Use Case 명세서에는 하나의 Use Case 가 가질 수 있는 비기능적 요구사항을 포함하여 기술한다.

3.2 IUC 기반 요구사항 모델링 절차

3.1 절에서 제시한 기본 개념을 기반으로 IUC 기반 요구사항 모델링 절차를 정의하면 그림 2 와 같다.

그림 2 에서 정의하는 모델링 절차에 대한 각 단계별 활동에 대한 정의는 다음과 같다. 각 단계에서의 절차 설명을 위하여 다음과 같은 요구사항을 갖는 군사용 탐사 로봇의 제어 시스템을 사용한다.

> 일정한 작전 지역에 매설된 지뢰를 탐지하여 지도상에 표시하고, 보병 병력이 이동하여 수 있도록 안전한 경로를 찾는다. 이때 탐사 로봇은 전방, 좌우측의 장애물을 탐지하기 위한 센서를 가지고 있으며, 매설된 지뢰를 찾는 탐지기를 보유한다. 탐지기에 의해 지뢰로 가정되는 물체를 발견하면 해당 위치 정보를 전송하여 지뢰의 위치를 표시한다.

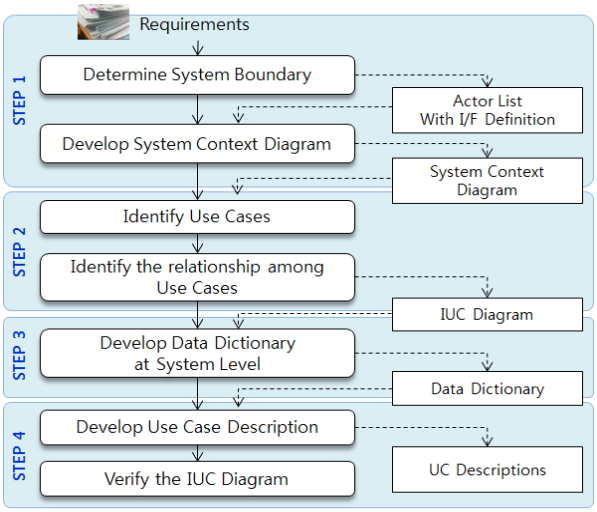


그림 2 IUC 기반 요구사항 모델링 절차

3.2.1 시스템 경계 설정

사용자 요구사항을 이용하여 시스템의 경계를 설정한다. 이 단계는 사용자 요구사항에 대한 브레인스토밍 과정을 거쳐 대상 시스템에 대한 전반적인 이해를 통해 진행된다. 시스템 경계가 정해지면 시스템 외부에 존재하는 액터를 식별하고, 각 액터가 시스템에 전달하거나 시스템으로부터 받는 인터페이스가 정의된다. 군사용 탐사로봇의 액터 목록은 표 2 와 같다.

표 2 군사용 탐사 로봇의 액터 목록

No	Actor Name	Description (Interface)
1	Frd_Sensor	앞쪽 장애물을 탐지하여 충돌 회피 (signal: Frd sensor.input)
2	L_Sensor	왼쪽 장애물을 탐지하여 진행을 위한 회전 방향 결정 (data(bool): Left sensor.input)
3	R_Sensor	오른쪽 장애물 탐지 및 회전 방향 결정 (data(bool): Right sensor.input)
4	M_Sensor	전방 2 미터 영역을 스캔하여 지뢰의 유무를 탐지 (data(bool): Mine sensor.input)
5	Pos_Sender	탐지센서 값이 True 인 경우 해당 위치를 출력 (data(double, double): Position.output)
6	Actuator	장애물 정보를 기반으로 탐사 로봇의 이동을 위한 구동기 출력 (data(int): Actuator.output)

3.2.2 시스템 문맥도 개발

시스템 경계, 액터 목록이 정의되면, 이를 이용하여 구조적 방법론에서 정의하는 시스템 문맥도를 작성한다. 시스템 문맥도는 전체 시스템을 블랙 박스의 개념으로 정의한 것으로써, 시스템과 외부와의 상호작용만을 표현하는 다이어그램이다. 군사용 탐사 로봇의 시스템 문맥도를 모델의 직관성으로 인해 생략하였다.

3.2.3 Use Case 및 관계 식별

시스템 문맥도에 나타난 블랙박스 시스템을 화이트 박스로 변환하기 위하여 시스템이 갖는 기능을 중심으로 Use Case 를 식별한다. Use Case 를 식별하기 위한 가이드라인은 다음과 같다.

(1) 액터와 직접 연결되어 상호작용하는 인터페이스 Use Case 를 먼저 정의한다.
(2) 인터페이스 Use Case 로부터 생성되는 데이터를 Data store 에 저장한다. 이러한 저장 정보는 시스템의 상태 정보를 결정하는 변수들이다.
(3) 시스템 외부로 특정한 값을 출력해야 하는 인터페이스 Use Case 가 앞서 정의된 Data Store 로부터 필요한 정보를 찾을 수가 없다면 시스템에서는 내부 행위를 갖는 기능적 Use Case 가 정의되어야 한다.
(4) UML 2.0 에서의 Use Case 다이어그램에 나타나는 <<extend>> 관계와 <<include>> 관계[13]를 갖는 Use Case 가 나타날 수 있다. 그러나 이러한 관계의 Use Case 들은 반드시 Data store 와 연결되어야 하는 것은 아니다.

군사용 탐사로봇 제어 소프트웨어에 대하여 작성된 스텝 2 의 모델링 산출물은 그림 3 과 같이 제시할 수 있다.

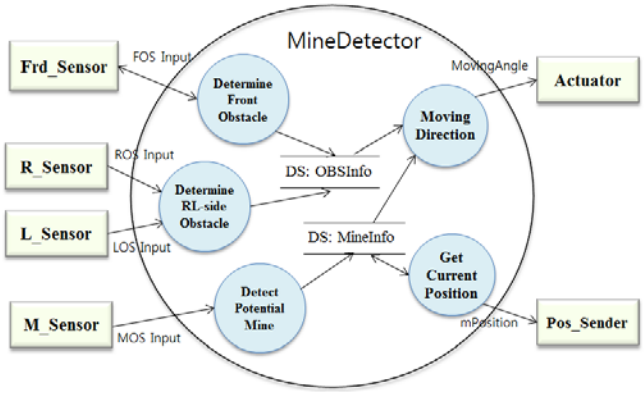


그림 3 군사용 탐사로봇의 IUC 다이어그램

3.2.4 시스템 수준 자료 사전 작성

IUC 다이어그램 작성이 완료되면, 보다 구체적으로 Data store 에 저장되는 데이터를 정의하기 위하여 자료 사전을 작성한다. 그림 3 에 나타난 Data store 중에서 "DS: MineInfo" 저장소의 자료 사전은 다음과 같이 정의할 수 있다.

```
DS: MineInfo
    = mineType + Probability + nPosition + ePosition
  mineType = [for_man | for_vehicle | for_bullet | for_scatter
| for-smart ]
  Probability = [1..100]
  nPosition = * The value of latitude *
```

ePosition = * The value of longitude *

3.2.5 Use Case 명세서 작성

Use Case 명세는 각 IUC 의 내부 행위에 대한 기능 흐름을 시간적 순서와 논리적 처리 관점에서 순차적으로 작성하는 과정이다. 이는 모델링 과정에서 식별되는 소프트웨어 시스템 기능에 대한 실행 관점을 나타내기 위한 것으로써, UML 순차 다이어그램을 작성하기 위한 입력으로 사용된다. 본 연구에서 정의하는 IUC 명세서의 내용은 다음과 같다.

UC Specification = UC Basic Information + Main flows + (Sub flows) + (Alternate flows) + (Nonfunctional requirements)

UC Basic Information = UC Name + ID + Primary Actor + Brief description + Triggering stimulus + Related Data store

Main flows = * Behavior description in step by step with unique index number *

Sub flows = * Behavior description for mandatory sub routine *

Alternative flows = * Handling routine of the exceptions occurring from Main flows or Sub flows *

Nonfunctional Requirements = * Specification of the requirements like resource, performance and security in quantitative manner *

3.2.6 IUC 다이어그램 검증

IUC 기반 요구사항 모델링의 마지막 단계는 앞서 개발된 IUC 다이어그램, 자료 사전, Use Case 명세서를 이용하여 모델링 결과물의 정확성, 일관성 및 완전성을 점검하기 위한 활동이다. 이러한 점검 활동은 크게 다음과 같은 2 가지 측면에서 이루어진다.

(1) IUC 식별의 적절성 점검

IUC 식별의 적절성은 각 Use Case 의 모듈화 특성이 충분한 것인가를 점검하기 위한 것이다. 즉 식별된 IUC 간의 기능 중복성이나 데이터 중복성 등을 점검하기 위함이다.

적절성 점검을 위하여 본 연구에서는 CRUD 매트릭스를 활용하였다. CRUD 매트릭스는 시스템에 존재하는 각각의 데이터에 대한 수명주기 즉 생성부터 삭제까지에 대한 정보를 담고 있는 테이블이다. CRUD 매트릭스의 세로 축은 각 IUC 와 이에 대한 UC 명세서의 Main flows index number 가 나열된다. 가로 축은 앞서 개발된 자료 사전에 정의된 데이터 항목을 나열한다. 다음과 같은 절차에 따라 점검이 이루어진다.

① UC 명세의 각 스텝에서 출현하는 데이터에 대하여 CRUD 를 정의한다.

② 모든 UC 의 스텝에 대한 CRUD 가 정의되면, 데이터 항목을 기준으로 매트릭스에 나타난 각 행(Row)를 재정렬한다.

③ 재정렬된 CRUD 매트릭스에 나타난 CRUD 표기를 사각형 영역으로 클러스터링(Clustering) 한다.

④ 클러스터링된 사각형 영역과 UC 의 경계를 비교한다.

⑤ 클러스터와 UC 의 경계가 일치하면 식별된 IUC 가 높은 기능적 모듈화 특성을 갖는다고 할 수 있다.

위에서 설명한 검증 과정을 개념적으로 표현하면 그림 4 와 같다.

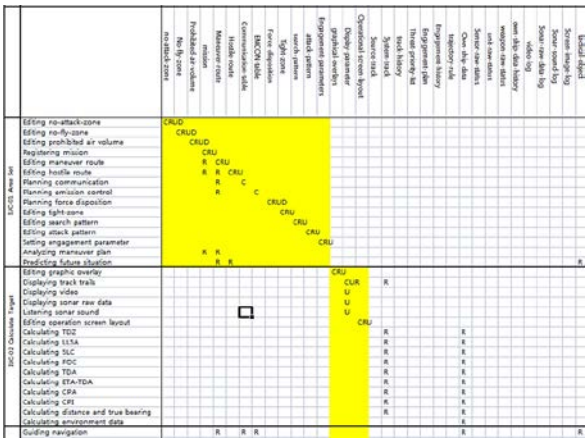


그림 4 IUC 적절성 판단을 위한 CRUD 매트릭스

(2) IUC 명세의 정확성 점검

IUC 명세 정보에는 Use Case 가 갖는 기능의 시나리오를 절차에 따라 기술하고 있는 Main flows 가 있다. 이 부분이 일반적으로 Use Case 가 가져야 할 속성을 만족하고 있는가? 다시 말해서 Use Case 의 실행을 트리거링 하는 입력과 종료 행위의 종결성(Termination)이 보장되고 있는가를 점검하고, 또한 Main flows 에 따른 기능 흐름에서 누락된 데이터는 없는지 등을 점검한다.

명세의 정확성을 점검하기 위해서는 일반적으로 사용하는 Use Case Realization 방법[14]을 적용하였다. 이는 시간적 순서에 따른 기능 흐름을 가시적으로 보여주며, 이를 통해 단계적으로 누락된 데이터, 사용하지 않는 데이터 등을 쉽게 찾아낼 수 있다.

4. 개발 프로세스와의 연계

앞 장에서 설명한 IUC 기반 요구사항 모델링 방법을 국방 전투관리체계 소프트웨어 시스템 개발을 위한 방법론 차원에서 연계 활용할 수 있다. 그림 5 는 통합된 개발 방법론에 대한 전체적인 절차를 간략하게 표현한 것이다.

본 논문에서 제시한 IUC 기반의 요구사항 모델링의 산출물은 IUC 다이어그램과 이들에 대한 Use Case 명세서 그리고 자료 사전이다. 이들을 기능 베이스라인[15]으로 하여 다음의 설계 개발 활동을 진행할 수 있다.

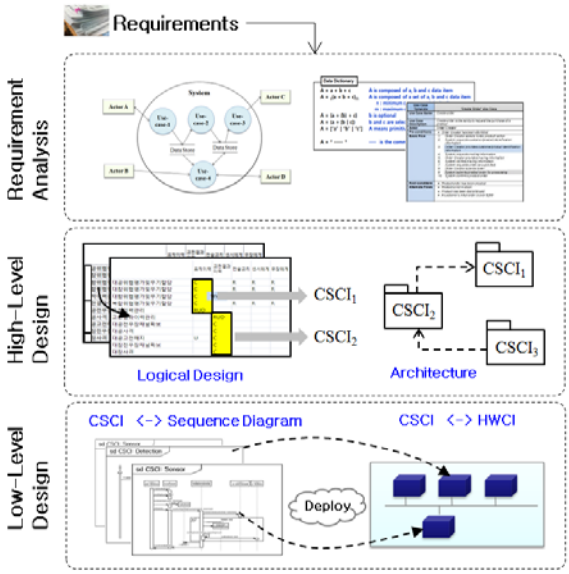


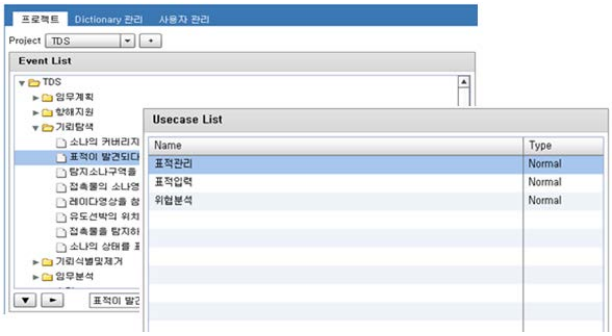
그림 5. 특화된 국방 SW 개발 프로세스

그림 5 와 같이 특화된 국방 전투관리체계 소프트웨어의 개발 프로세스에서는 상위 수준의 설계활동으로 CRUD 를 기반으로 하는 CSCI (Computer Software Configuration Item)를 식별[16]하고, 이를 기반으로 전체 소프트웨어 아키텍처를 개발한다. 하위 설계 즉, 상세 설계 단계에서는 식별된 각 CSCI 에 대한 실행 모델을 순차 다이어그램을 이용하여 설계하고, HWCI (Hardware Configuration Item)와의 매핑 작업을 수행한다. 상세 설계 과정에서 자료 사전에 대한 정제 작업이 부가적으로 수행될 수 있다.

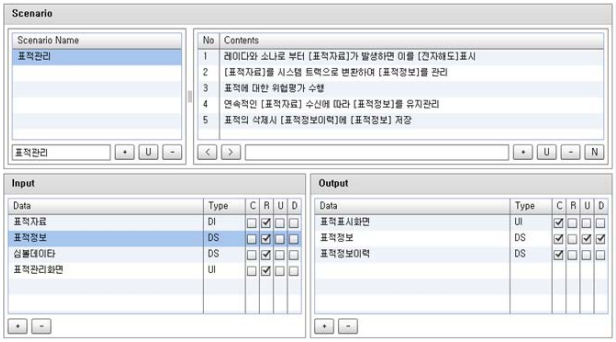
5. 도구 지원

본 연구에서 제안하는 요구사항 모델링을 지원하기 위한 도구가 개발되었다. 전체 도구가 갖는 기능은 요구사항 관리기능, IUC 기반 모델링 기능, 테스트 지원 기능, 요구사항 추적성 관리 기능 등과 같이 국방 소프트웨어 개발을 위한 전체수명주기를 지원하고 있다.

본 논문에서는 도구의 전체 기능 중에서도 특히 요구사항 모델링을 위해 Use Case 를 식별하고, 또한 Use Case 와 관련된 자료 사전 정의, 그리고 Use Case 에 대한 명세(도구에서는 Scenario 로 표현됨)를 작성하는 기능에 대하여 설명한다. 그림 6 은 요구사항 모델링을 위한 도구 화면 중에서 (a) Use Case List 와 Event List 출력 화면, (b) Use Case 명세와 Use Case 에 대한 자료 사전에서의 CRUD 정의 화면, 그리고 (c) 자료 사전 생성 및 조회 기능 화면을 보여준다.



(a) Use Case List 와 Event List 출력 화면



(b) Use Case 명세와 데이터 CRUD 정의 화면

(c) 자료 사전 생성 및 조회 화면

그림 6 IUC 기반 요구사항 모델링 지원 도구

그림 6 의 (b)는 Use Case 의 Main flows 를 명세하는 과정에서 식별된 데이터에 대한 C, R, U, D 정보를 체크하는 화면이다. 이러한 화면을 제공함으로써, 사용자는 보다 쉽고 정확하게 CRUD 매트릭스를 작성할 수 있게 된다.

6. 결 론

본 논문에서는 시스템에서 중요하게 고려되는 데이터의 특성을 반영하는 국방 전투관리체계 소프트웨어 개발을 위한 Interactive Use Case 기반의 요구사항 모델링 방법을 제안하였다. 이 방법에서는 초기 사용자의 요구사항으로부터 기능적 요구사항을 분석하기 위한 방법으로 Use Case 를 도출하게 되는데, 이 과정에서 각 Use Case 와 연관된 데이터를 식별하도록 하는 IUC 를 정의하였다. 또한 기존의 무거운 개발 방법론 적용을 간소화하여 CRUD 매트릭스를 활용한 소프트웨어 시스템의 논리적 설계 등이 이루어질 수 있도록 제안하는 모델링 방법과 연계하여 개발 방법론을 정립하였다.

기존에 지원되지 못했던 요구사항 분석의 초기 단계에 시스템이 관리하는 핵심 데이터 영역을 고려함으로써, 보다 명확한 요구사항 이해 및 도출이 가능해지고, 설계에 대한 가시성도 높일 수 있으며, 또한 기능점수 및 Use case 점수에 의한 소프트웨어 규모 예측도 용이할 수 있다는 장점도 제공할 수 있다.

Acknowledgement

본 논문은 LIG 넥스원 마리타임연구소에서 정립한 소프트웨어 개발방법론을 포함하고 있으며, 연구재단(미래창조과학부)의 부분적인 지원을 받아 수행된 연구임(No. NRF-2015M3C4A7030505).

참고문헌

[1] A. P. Black, “Object-Oriented Programming: Some history and Challenges for the Next Fifty Years,” Information and Computation, May 2012.

[2] OMG UML, “OMG Unified Modeling Language Superstructure, V. 2.4.1,” OMG, Aug., 2011

[3] E.T. Hvannberg, “Combining UML and Z in a Software Process,” GI Jahrestagung (1), 2001

[4] R. Elmansouri, H. Hamrouche, A. Chaoui, “From UML Activity Diagrams to CSP Expressions,” International Journal of Computer Science, 8(2), March 2011, pp.368-374

[5] LIG_NEX1, “System Requirements Analysis” TC-OPV80P-SSRA-01, August 2014.

[6] H. Gomma, Designing Concurrent, Distributed, and Real-Time Application with UML, Addison Wesley, 2000.

[7] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison Wesley. 1998.

[8] I. Jacobson, M. Christerson, P. Jonsson , G. Overgaard, “ Object-Oriented Software Engineering: A Use Case Driven Approach, Addison Wesley, 1992

[9] Prashant, S. Gupta, “Simplifying Use Case Models Using CRUD Patterns,” International Journal of Soft Computing and Engineering, 2(2), May 2012, pp. 104-106

[10] A. Cockburn, Writing Effective Use Cases, Addison Wesley, 2000

[11] B. Paech, “The Four Levels of Use Case Description,” REFSQ'98, 1998, pp.207-218

[12] Tom DeMarco, Structured Analysis and Systems Specification, YOURDON Press, 1979.

[13] K. Bittner, I. Spence, Use Case Modeling, Addison Wesley, 2003

[14] I. Jacobson, “Use Cases and Aspects – Working Seamlessly Together,” Journal of Object Technology, 2(4), July 2003, pp.7-28

[15] DoD, Configuration Management, MIL-DTD-3046, March 2013

[16] Charles S. Wasson, System Engineering Analysis, Design, and Development: Concepts, Principles, and Practices, Wiley, 2015

PbD 7 대 기본원칙과 GQM 을 활용한 프라이버시 요구사항 도출 방법론

조주혜

아주대학교 소프트웨어특성화 대학원
경기도 수원시 영통구 월드컵로 206
아주대학교 팔달관 913-1 호
chojuhye@ajou.ac.kr

이석원

아주대학교 소프트웨어 융합학과
경기도 수원시 영통구 월드컵로 206
아주대학교 팔달관 913-1 호
leesw@ajou.ac.kr

요약: 개인정보 유출 사건이 빈번하게 발생하면서 개인정보 보호에 대한 기대와 관심이 높아지고 있다. 이에 따라 법률을 기반으로 프라이버시 요구사항을 도출해내는 방법들이 연구되고 있지만, 여전히 요구사항 도출 단계에서는 프라이버시 요구사항에 대한 명확한 정의는 이루어지지 않고, 보안 기술 및 일반적인 기능 요구사항에만 초점을 두고 있다. 본 연구에서는 이러한 문제를 해결하기 위해, 기능 요구사항에 개인정보 보호 개념을 접목하여 새로운 프라이버시 요구사항을 도출하는 방법론을 제안하고, 사례연구를 통해 제안 방법을 검증한다.

핵심어: Privacy Requirements Elicitation, Privacy by Design 7 Principles, Goal Question Metric, Privacy Assurance Case

1. 서론

개인정보란 “살아 있는 개인에 관한 정보로서 성명, 주민등록번호 및 영상 등을 통하여 개인을 알아볼 수 있는 정보(해당 정보만으로는 특정 개인을 알아볼 수 없더라도 다른 정보와 쉽게 결합하여 알아볼 수 있는 것을 포함한다)를 말한다.” [1] 개인정보의 사용과 온라인 상에서의 개인정보 축적이 증가하면서, 여러 도메인에서 개인정보 보호가 중요시되고 있다. 특히 개인정보의 가치와 개인정보 침해에 따른 사회적 비용이 가장 높은 헬스케어 분야 [2]에서 개인정보 보호는 중요한 이슈이다. 실제로 2015 년에는 병원 청구 소프트웨어 개발업체 지누스가 7 억건에 해당하는 진료 기록을 무단으로 유출한 사건과 미국 2 위 규모의 건강보험사인 앤섬에서도 고객의 개인정보가 누출된 사고가 있었다.[1,2]

이처럼 세계적으로 개인정보 유출 사건이 빈번하게 발생하고 있어 개인정보 보호에 대한 관심은 높아지고 있지만, 실제 시스템 개발을 위한 요구사항 분석 및 도출 단계에서 프라이버시 요구사항을 명확히 정의하지 않는 경우가 많다. 이를 위해 시스템 구축 시 해당 도메인과 관련된 개인정보 보호 가이드나 법률을 기반으로 어떤 개인정보가 보호되어야 하는지, 개인정보 사용시 어떤 조치가 필요한지 등에 대한 명확한 이해와 정의가 우선시 되어야 한다. 하지만 개인정보 관련 문서의 분석을 통해 개인정보 보호를 고려한 요구사항을 도출하는 접근법은 아직까지 부족한 상황이다. 따라서 본 연구에서는 이러한 문제점을 해결하기 위해, 일반적인 기능 요구사항들이 개인정보 보호를 고려하도록 재정의되거나, 새로운 프라이버시 요구사항을 도출할 수 있는 방법론을 제안한다.

본 논문의 구성은 다음과 같다. 2 절에서는 선행 연구된 프라이버시 및 법률 요구사항 도출 방법론과 제안하는 방법론에 사용된 Privacy by Design 7 대 기본원칙과 Goal Question Metric 접근법의 개념을 살펴본다. 3 절에서는 이 개념들을 적용한 프라이버시 요구사항도출 방법론을 제안하고, 이 방법을 활용하여 개인정보 보호를 고려한 요구사항을 도출하는 사례 연구를 제시한다. 마지막 4 절에서는 연구의 결론 및 향후 연구 방향에 대하여 서술한다.

2. 관련 연구 및 배경 기술 소개

본 절에서는 프라이버시 및 법률 요구사항의 도출과 관련된 선행 연구와 제안할 프라이버시 요구사항 도출 방법론에 사용된 개념들을 소개한다.

2.1 관련 연구

기존의 프라이버시 요구사항 분석 및 도출은 개인정보 보호법과 같은 법률을 기반으로 자연어 패턴 분석을 통해 법률 요구사항을 도출하는 방법에 바탕을 두고 있다.

Travis 와 Annie 의 연구 [3] 에서는 개인정보 보호와 관련된 법률 문서에서 Semantic Model 을 추출하기 위한

[1] 지누스 진료기록 유출(http://www.bloter.net/archives/218452)

[2] 앤섬 개인정보 누출(http://www.zdnet.co.kr/news/news_view.asp?artice_id=20151220151142)

Semantic Parameterization 프로세스를 제안하였다. 해당 프로세스는 Semantic Model 과 Goal 접근법을 함께 이용하여 길고 복잡한 법 조항에서도 핵심 의미를 파악하는데 용이하다. 하지만 해당 연구의 범위가 법 조항 분석에 그치고 실질적인 프라이버시 요구사항 도출까지 연결되지는 않았다.

Alshugran 와 Dichter 의 연구 [4]에서는 HIPAA(Health Insurance Portability and Accountability Act)에서 역할(Role), 목적(Purpose), 의무(Obligation)를 추출하여 헬스케어 어플리케이션의 프라이버시 요구사항을 도출하는 프로세스를 제안하였다. 이 프로세스를 따르면 법률의 모호성과 복잡성을 제거할 수 있다. 또한 상호 참조되는 법률까지 고려하여, 법률이 개정될 경우 관련 프라이버시 요구사항을 바로 확인할 수 있다는 장점이 있다. 하지만 개발할 시스템의 특징은 고려하지 않았기 때문에 도출된 요구사항을 그대로 적용하기 어렵고, HIPAA 를 기반으로 하였기에 헬스케어 도메인에만 한정되어 있다.

본 연구에서 제안하는 방법은 개인정보 보호 및 보장이 필요한 다양한 도메인에 적용 가능하며, 시스템 개발 주기 전체를 고려한 실질적인 프라이버시 요구사항을 도출할 수 있는 방법론이다.

2.2 Privacy by Design 7 대 기본원칙

Cavoukian 의 Privacy by Design(PbD)의 7 대 기본원칙 [5]은 IT 기술의 발전에 따라, 시스템 개발 수명 전체에 걸쳐 이용되는 개인정보를 보호하기 위해 제안된 시스템 설계 원칙이다. 초기에는 PETs(Privacy-Enhancing Technologies)라 불리는 기술 영역을 중심으로 PbD 가 논의되었고, 현재는 개인정보 보호를 위한 방법론과 정책으로 확장되었다. 개인의 프라이버시를 보장하고, 개인정보에 대한 통제권의 확보 및 개인정보를 사용하는 기관들의 지속적인 경쟁 유지를 목적으로 하는 PbD 는 다음의 7 대 기본원칙 준수를 통해 가능하다.

1) 사후 대응이 아닌 사전 대비(문제점을 고치는 것이 아니라 예방)
2) 개인정보 보호를 기본 값(Defaults)으로 설정
3) 기획 단계에서부터 개인정보 보호 고려
4) 포괄적 기능성 보장(Zero Sum 이 아닌 Positive Sum 을 추구)
5) 전체 수명주기의 보호
6) 가시성과 투명성의 확보
7) 개인의 프라이버시 존중(사용자 중심의 설계와 운영)

2.3 GQM(Goal Question Metric)

Basili et al 의 Goal Question Metric (GQM) [6]은 목표 지향적인 소프트웨어 Measurement 방법 중 하나이다. 그림 1 처럼 GQM 은 3 단계의 구조로 정의된다. 첫 번째 단계에서는 개발하고자 하는 소프트웨어의 목표(Goal) 혹은 조직의 목표(Goal)을 결정하고, 두 번째 단계에서는 목표를 달성하기 위해 해결해야 할 질문(Question)을 정의한다. 마지막 단계에서는 정량적 혹은 정성적인 방법(Metric)으로 질문에 대한 대답을 도출하는 것이다. 이 과정을 통해 소프트웨어가 목표에 부합하게 개발되었는지 검증할 수 있다.

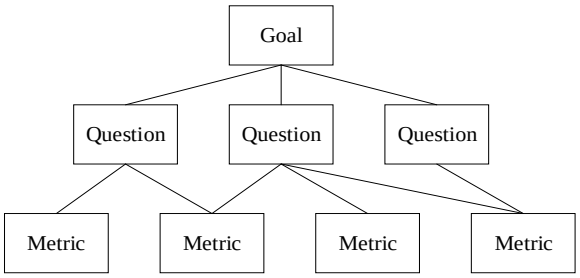


그림 1. GQM 접근법

3. 제안 방법론

본 절에서는 앞 절에서 살펴본 PbD 7 대 기본 원칙과 GQM(Goal Question Metric) 접근법을 이용하여, 기존의 일반적인 기능 요구사항이 개인정보 보호를 고려한 요구사항이 되도록 수정하거나 새로운 프라이버시 요구사항을 도출하는 방법론을 제안한다. 그리고 제안한 방법론을 적용한 사례 연구를 제시한다.

3.1 프라이버시 요구사항 도출 방법론

본 연구에서 제안하는 PbD 7 대 기본원칙과 GQM 접근법을 이용한 프라이버시 요구사항 도출 방법론은 5 단계로 구성된다.

[1 단계] PbD 7 대 기본원칙을 Privacy Goal 로 선언
[2 단계] 기존의 기능 요구사항과 Privacy Goal 매칭
[3 단계] 목표를 충족시킬 수 있는 질문을 추출
[4 단계] 해당 질문의 평가를 위해 정량적 혹은 정성적인 방법을 사용
[5 단계] 목표를 충족시키지 못한 기능 요구사항을 수정 혹은 새로운 프라이버시 요구사항 재정의

첫 번째 단계에서 PbD 7 대 기본원칙을 개인정보 보호를 고려한 요구사항이 되기 위하여 필수적으로 만족해야 하는 개념으로 사용한다. 이를 위해 해당 7 가지 기본원칙을 기능

요구사항들이 가져야 할 Privacy Goal 로 선언한다. 표 1 은 7 가지 Privacy Goal 을 정리한 것이다.

표 1. 7 가지 Privacy Goal

PbD 7 대 기본원칙	Privacy Goal
사후 대응이 아닌 사전 대비	개인정보 유출의 사전 예방
개인정보 보호를 기본 값으로 설정	개인정보 보호를 보장
기획 단계에서부터 개인정보 보호 고려	프라이버시를 고려한 설계
포괄적 기능성 보장	적절한 보안 방법의 적용
전체 수명주기의 보호	개인정보의 사용 및 폐기
가시성과 투명성의 확보	투명한 개인정보의 처리
개인의 프라이버시 존중	개인정보 사용에 대한 알림 및 통제권 보장

두 번째 단계에서는 일반 기능 요구사항에서 개인정보와 관련된 주어, 목적어, 동사를 도출하여 Privacy Goal과 비교한 후 적합한 Privacy Goal 로 매칭시킨다. 이 때, 매칭되는 Privacy Goal 은 하나 이상이 될 수 있다.

세 번째 단계에서는 각 Goal 을 달성할 수 있는 질문을 도출한다. 질문은 정형화되어 정해진 것이 아니라, 해당 도메인에 적합하게 수정 및 구체화되어 사용 할 수 있다. 본 연구에서는 7 가지 Privacy Goal 에 해당하는 대표적인 질문을 정리하여 표 2 에 제시하였다.

네 번째 단계에서는 앞 단계에서 추출된 질문을 정성적 혹은 정량적으로 평가할 수 있는 방법(Metric)을 이용하여 해당 기능 요구사항이 매칭된 Privacy Goal 을 만족하는지 평가한다. 본 연구에서는 2011 년 개인정보보호법 제정·시행에 따라 공공기관에서 의무적으로 준수해야 할 조치사항인 '개인정보 영향평가 수행 안내서' [7]를 활용하여, 질문에 대한 평가를 적용, 부분적용, 미적용, 해당 없음 4 가지로 분류하여 질문 평가 Metric 으로 사용했다. 질문의 평가 기준은 도메인에 적합한 소프트웨어 개발 가이드라인 등을 적용하여 부분적용, 미적용, 해당 없음에 해당되는 요구사항은 Privacy Goal 을 만족시키지 못한다고 판단할 수 있다.

마지막 단계에서는 앞 단계에서 기능 요구사항을 평가한 결과를 바탕으로 질문 내용이 반영되도록 요구사항을 수정하거나 혹은 새로운 프라이버시 요구사항을 도출한다.

제안하는 방법론을 따라서 새롭게 정의된 프라이버시 요구사항은 PbD 7 대 기본 원칙을 기본으로 하기에 시스템의 전체 수명주기에 걸쳐 사용자의 개인정보를 보호하는데 도움이 된다. 또한 GQM 을 이용하여 기존의 기능 요구사항을 Privacy Goal 과 매칭시키고 검증하여 Privacy Friendly System [8]을 개발하는데 기반을 제공한다.

3.2 사례 연구

본 논문에서는 비만 예방 어플리케이션 개발을 위한 기능 요구사항에 제안한 방법론을 적용하여 프라이버시를 고려한 요구사항을 도출하였다. 최근에는 웨어러블 기기의 발달과 함께 헬스케어 관련 모바일 앱 또는 플랫폼 등이 이슈가 되고 있다. 이에 따라 헬스케어 분야에서의 개인정보 보호의 중요성이 높아지고 있기에 사례연구 도메인으로 적합하여 비만 예방 어플리케이션을 사용하였다.

비만 예방 어플리케이션의 많은 기능 요구사항들 중에서 본 사례 연구에서는 '비만 어플리케이션은 위치정보를 이용하여 사용자의 정확한 운동량을 측정한다.'를 사용했다.

표 2. 7 가지 Privacy Goal 을 위한 질문(Question)

7 가지 Privacy Goal	Question
개인정보 유출의 사전 예방	– 예기치 못한 사건에 대하여 알림 기능이 있는가? – 개인정보 침해요인의 발생가능성을 고려하였는가?
개인정보 보호를 보장	– 민감한 개인정보 보호 방법이 적절히 제시되었는가? – 서비스 이용에 필요한 최소한의 정보 공개 설정이 가능한가?
프라이버시를 고려한 설계	– 도메인에 적합한 가이드라인을 참조하였는가?
적절한 보안 방법의 적용	– 개인정보가 개인 기기에 저장되는 경우, 취약점에 대하여 안내하는가? – 개인정보 저장 시, 적절한 암호화 방법을 사용하는가?
개인정보의 사용 및 폐기	– 회원 탈퇴 시, 개인정보 및 이용정보의 삭제가 고려되는가?
투명한 개인정보의 처리	– 이용약관 및 개인정보 취급방침 공개를 고려하는가? – 개인정보 취급방침의 변경이 발생하는 경우, 사용자에게 어떤 방식으로 공지를 하는가?
개인정보 사용에 대한 알림 및 통제권 보장	– 개인정보 공개 범위에 대한 통제권을 사용자가 갖고 있는가? – 이용자에게 앱 권한 및 개인정보 취급방침을 사전에 알리는 절차가 있는가?

가장 먼저 해당 요구사항에서 개인정보와 관련된 키워드(주어, 목적어, 동사)로 '비만 예방 어플리케이션이 위치정보를 이용'을 도출했다. 다음으로 키워드가 매칭되는 Privacy Goal 중에 '개인정보 사용에 대한 알림 및 통제권 보장'이라는 Privacy Goal 에 대한 질문과 평가 결과를 표 3 과 같이 도출하였다. 본 사례 연구에서는 '스마트폰 앱 개인정보보호 가이드라인' [9]을 기준으로 질문에 대한 평가를 진행하였고, 기존 요구사항이 Privacy Goal 을 충족시키지 못했기에 다음과 같은 프라이버시 요구사항을 도출하였다.

- *P_RE1. 단말기 정보에 접근 시, 사용자에게 접근을 알리고 동의 여부를 얻는다.*
- *P_RE2. 해당 앱이 접근하는 정보에 대하여 권한이 필요한 이유를 명확히 제시한다.*

표 3. 제안한 방법 활용 사례 – Metric 결과

질문 1			
– 모바일 앱 동작을 위해 이용자의 단말기 정보에 접근하는 경우, 해당 시점에 사용자에게 정보 접근 허용 여부를 안내하는가?			
적용	**부분적용**	**미적용**	**해당 없음**
		O	
평가 근거	스마트폰 앱 개인정보보호 가이드라인		
질문 2			
– 어떤 정보를 이용하는지 사용자가 인식 할 수 있도록 쉽고 명확하게 명시하는가?			
적용	**부분적용**	**미적용**	**해당 없음**
		O	
평가 근거	스마트폰 앱 개인정보보호 가이드라인		

4. 결론 및 향후 연구 방향

본 연구에서는 PbD 7 대 기본원칙을 바탕으로 기존의 기능 요구사항을 검증하여 프라이버시 요구사항을 도출하는 방법론을 제시하였다. 이 방법론은 기존의 선행 연구들과 달리, 소프트웨어의 전체 수명주기를 고려한 요구사항을 도출할 수 있으며, 다양한 도메인에서 사용이 가능하다. 그렇기 때문에 모든 도메인에서 Privacy Friendly System [8]을 개발하는 데 활용할 수 있다.

향후 크게 두 가지 측면에서 제안 방법을 보완 및 확장할 예정이다. 우선 기능 요구사항과 Privacy Goal 을 매칭시킬 때, 요구사항 엔지니어의 주관에 따라 다르게 매칭시킬 수 있다는 한계점이 있다. 이 부분을 Dzung 과 Ohnishi 의 연구 [10]를 참고하여 요구사항과 Goal 을 매칭시키는 온톨로지를 구축하여 보완할 예정이다. 개인정보를 고려한 기능 요구사항의 도출 방법에 대하여 다루었지만, 실제 적용할 수 있는 보안 기술까지는 연결되지 않았다. 이를 해결하기 위해, 요구사항을 토대로 엔지니어가 계획 및 예상한대로 시스템이 작동하고 개인정보 보호가 보장될 수 있도록 Security Assurance Case [11] 개념을 적용하여, Privacy Assurance Case 로 확장할 예정이다.

Acknowledgement

이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2010-0028631).
본 연구는 미래창조과학부 및 정보통신기술진흥센터의 실전적 SW 교육(SW 중심대학)지원사업의 연구결과로 수행되었음(R7115-15-1005).

참고 문헌

[1] 법. 제 13423 호, *개인정보 보호법,* 행정자치부, 개정 2014.03.24.

[2] *개인정보의 가치와 개인정보 침해에 따른 사회적 비용 분석,* (사)개인정보보호협회, 2013.

[3] T. D. Breaux and A. I. Anton, "Deriving Semantic Models form Privacy Policies," in *Policies for Distributed System and Networks(POLICY)*, 2005.

[4] T. Alshugran and J. Dichter, "Extracting and Modeling the 프라이버시 요구사항 form HIPPA for Healthcare Application," in *Systems, Applications and Technology Conference (LISAT)*, 2014.

[5] A. Cavoukian, *Privacy by Design, The 7 Foundational Principles,* 2006.

[6] V. R. Basili and G. Caldiera, The Goal Question Metric Approach, Encyclopedia of Software Engineering, 1994.

[7] *개인정보 영향평가 수행 안내서,* 행정자치부, 2015.03.

[8] S. Spiekermann and L. F. Cranor, *Engineering Privacy,* IEEE Computer Society, 2009.

[9] *스마트폰 앱 개인정보보호 가이드라인,* 방송통신위원회, 한국인터넷진흥원, 2015.

[10] D. V. Dzung and A. Ohnishi, "A Verification Method of Elicited Software Requirements using Requirements Ontology," in *Asia-Pacific Software Engineering Conference*, 2012.

[11] C. B. Weinstock, H. F. Lipson and J. Goodenough, *Arguing Security – Creating Security Assurance Cases,* Software Engineering Institute(SEI), Carnegie Mellon University, 2013.

2016 한국 소프트웨어공학 학술대회

지능형 화폐 인식 SW 개발의 Visualization 적용 사례 및 임베디드 SW 개발 조직의 Visualization 적용 로드맵 제안

[산업체 논문]

한동준	소프트웨어 안전성 보증 연구센터	handongjoon@gmail.com
김은비	상명대학교 컴퓨터과학과	aqua9131@gmail.com
오승원	상명대학교 컴퓨터과학과	ohsing0906@gmail.com
엄영석	기산전자㈜	eom.youngseok@kisane.com
정대식	기산전자㈜	jeong.daesik@kisane.com
조달호	기산전자㈜	cho.dalho@kisane.com
한혁수	상명대학교 컴퓨터과학과	hshan@smu.ac.kr

[본 연구는 정보통신산업진흥원 소프트웨어공학센터(NIPA)의 SW공학기술 현장적용지원사업 연구결과로 진행되었음(SA105-15-1102)]

[목 차]

1 SW Visualization 적용 필요성 및 목표

기산전자 적용 이전 현황

주요 기술 분야의 연구에서 큰 성과를 달성

(위폐감별, 권종 및 일련번호 인식 등의 지능형 소프트웨어 기능)

1. 현재까지는 <위조 변조 검출>을 위한 패턴 인식 알고리즘 개발에 노력
2. 시장 점유율이 높아지고, 전 세계적으로 기술력을 인정 받음

SW 공학 측면에서는 전문가의 도움이 필요함

1. 제품 인도 후 발생하는 결함으로 인한 비용 증대
2. 요구사항 변경 관리가 체계적으로 이루어 지지 않아, 재작업이 많음
3. 소스코드 통합관리, 버전관리 개선 필요
4. Nipa의 SW공학 멘토링 과제(2014년 7월)를 통해 Redmine과 Sonarqube 도구 사용 교육을 받았으나 거의 사용하지 않는 상태임

SW공학기술 현장적용 지원사업을 통한 목표

KCSE 2016 논문집 18권 1호

프로세스 기반의 작업

1. 컨설팅을 통해 기산전자의 표준 프로세스를 개발하고, 지원 환경을 구축
2. 요구사항 / 테스트 프로세스 정의

체계적인 개발 문화

1. 도구 기반의 프로세스 환경 구축
   - 이슈관리 도구 : Redmine
   - 형상관리 도구 : SVN (Subversion)
   - 지속적통합 도구: Jenkins
   - 정적분석 도구 : N'SIQ, CPD, 의존성 분석 도구
2. 로그 데이터의 분석을 통한 프로세스 준수 사항을 점검

S/W Visualization 대상

소프트웨어 공학 지식 전수

1. 현장에서의 **교육**을 통하여 작업에 차질 없이 SW 공학적 지식 전수
   - 요구사항 교육
   - 테스트 교육
   - 도구 활용 교육

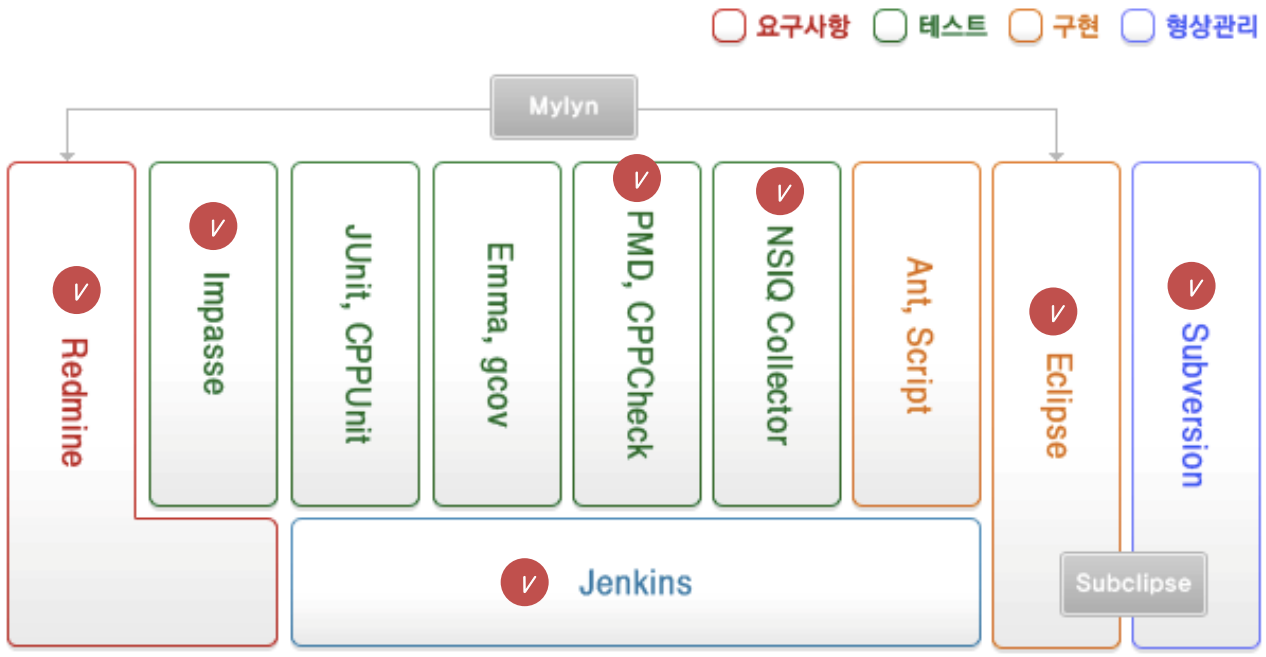
문제점 해결 방안 도출

1	표준 프로세스 및 개발 방법론 개발	○ 도구 기반 통일된 요구사항 명세 및 관리 프로세스 구축 ○ 도구 기반 단위 테스트 프로세스 구축
2	프로젝트 저장소 및 자산 라이브러리 구축	○ 모든 프로젝트 개발 관련 산출물의 자산화 ○ 개발 후 커밋 메시지 표준화
3	오픈 소스 도구 기반의 개발/지원 환경 구축	○ Redmine, SVN 등의 오픈 소스 도구 연동 방안 ○ Redmine, Jenkins 등의 오픈소스 도구 활용 방안
4	지속적이고 효과적인 교육 수행	○ 프로세스 기반의 개발 문화 형성 ○ 현재 기산전자 내에서 사용하는 도구들 간의 연동 프로세스 교육
5	품질 보증(QA)팀 구성	○ 품질 보증 활동의 전담 인력 확보

2 적용 기술

소스코드와 개발 프로세스를 관리하는 것을 목적으로 하고,
시각화와 문서화를 그 방안으로 하여 S/W 개발 품질관리를 수행하는 것

[출처: NIPA]



✓ 기산전자 적용 항목

[출처: NIPA]

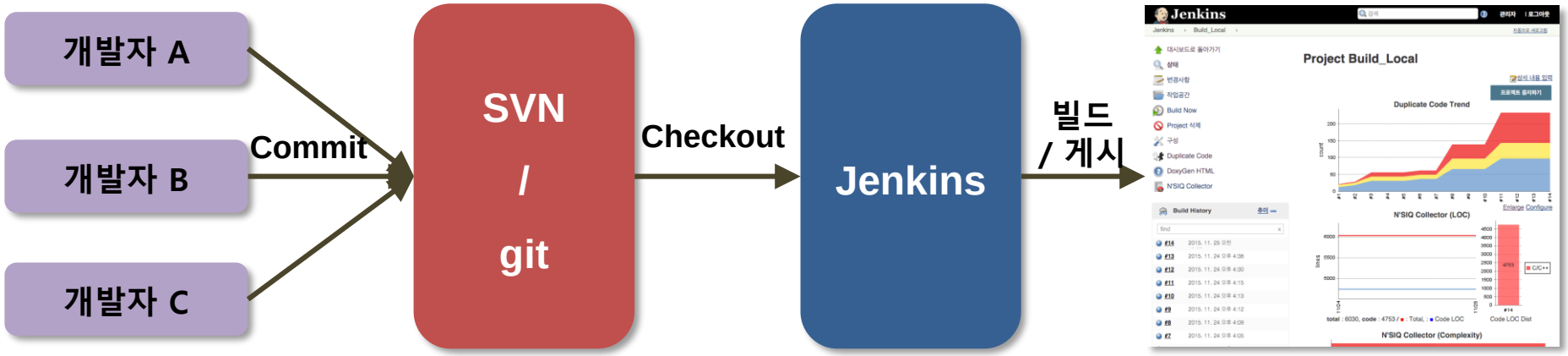
지속적 통합(CI: Continuous Integration)

KCSE 2016 논문집 18권 1호

소프트웨어 통합 오류를 개발 초기부터 예방하는 것

- 팀 구성원들이 자신이 한 일을 **자주 통합**하는 소프트웨어 개발 실천 방법
- 각 통합은 자동화된 빌드를 검증하여 **최대한 빨리 통합 오류**를 탐지
- 하루에 **여러 번 통합 빌드**를 수행하는 것

\- 마틴 파울러 [1]



임베디드 SW 개발 적용의 어려움 [2]

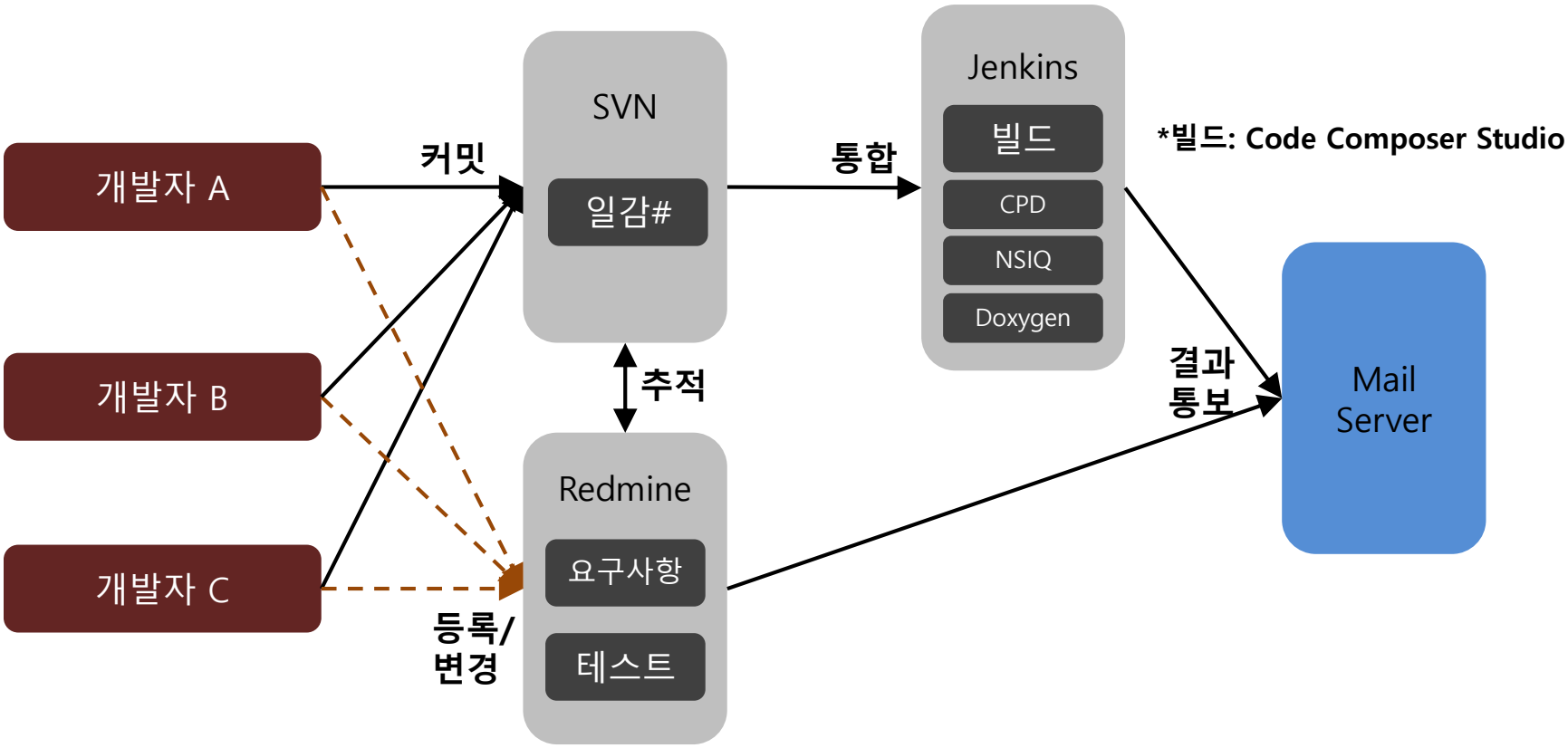
KCSE 2016 논문집 18권 1호

- 호환이 잘 안되는 너무 다양한 컴파일러 및 버전
- 민감한 빌드 환경
- 상용 도구 위주의 SW 품질 확보
- 적용 사례의 외부 공유 미흡
- SW 개발을 돕는 새로운 기술에 대한 적은 관심
- 웹/앱(java 기반) 개발에만 적용 가능하다는 오해

3 SW Visualization 적용

기산전자 SW Visualization 적용 구성도

NIPA의 SW Visualization을 기반으로 임베디드 SW에 적합한 도구 선택 및 적용

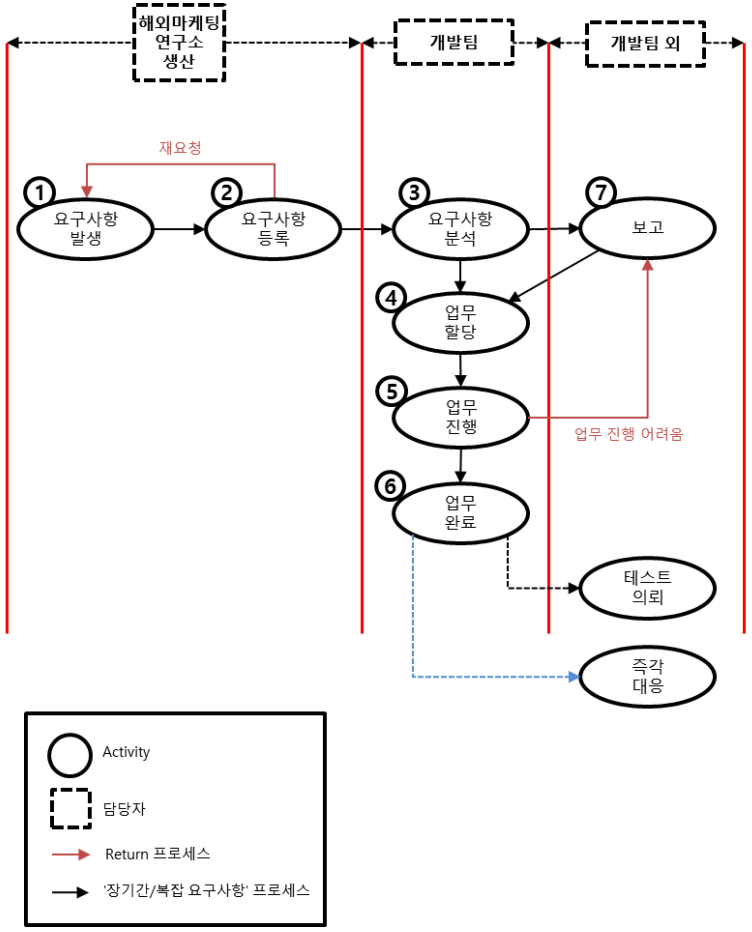


요구사항 관리 프로세스 개선 및 도구 적용

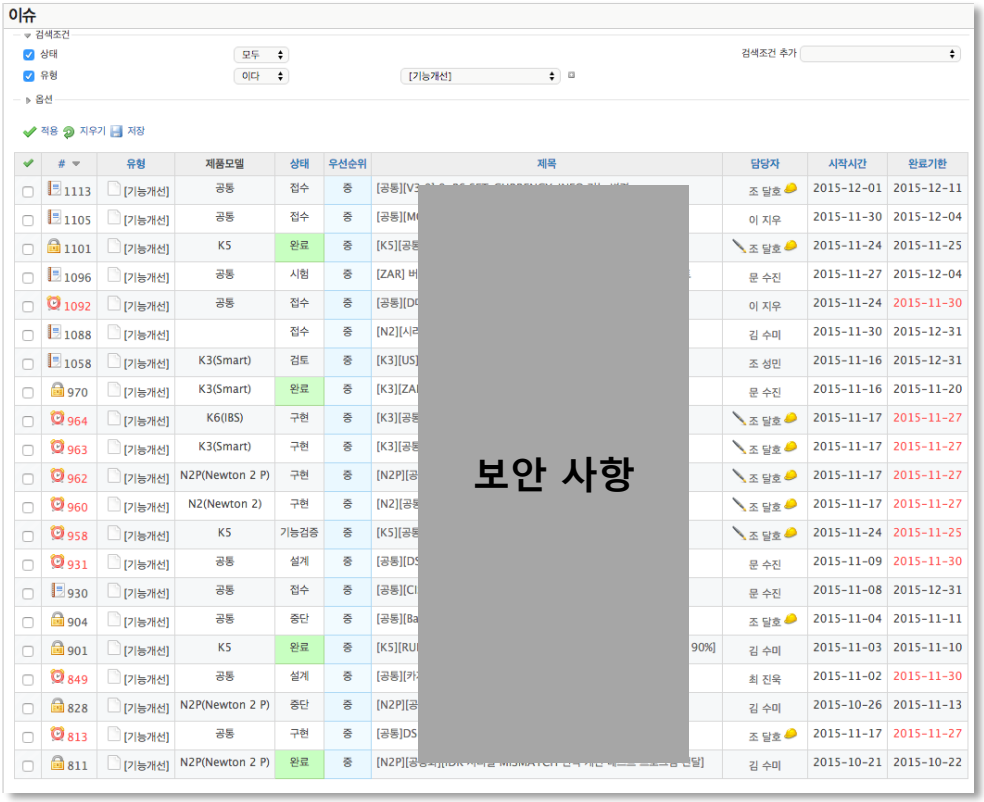
KCSE 2016 논문집 18권 1호

SW 개발팀 내 요구사항 등록 창구 단일화와 단계 개선 후 Redmine 적용 및 지속 활용

요구사항 관리 프로세스 개선



Redmine 일감/단계 적용 및 활용



요구사항과 코드 변경내역 추적 관리

Redmine 일감 기반 개발 및 변경내역 추적을 위한 도구간 연동과 커밋 메시지 표준화

SVN 커밋 메시지 표준화

(Issue #1234) [CA사][V2.0] DSP 프로그램 준비 및 전달
closed #1234

보안 사항

Redmine #1234 일감에서 변경 추적

이력

문 수진이(가) 4일 전에 변경 #1

- 상태을(를) 접수에서 구현(으)로 변경되었습니다.
- 진척도을(를) 0에서 90(으)로 변경되었습니다.
- 국가권에 ALL이(가) 추가되었습니다.

* 프로그램 개발 내용

[DSP] Single Mode로 UV Cal 진행 후, UV Cal 결과 값을 장비에 탑재되어 있는 모든 국가의 동일하게 적용이 될 수 있도록 Memcpy 기능 추가

* 진행 상황

관련된 개정판들

개정판 1636
문 수진이(가) 2일 전에 추가함

(Issue #1234) : [CA사][V2.0] DSP 프로그램 준비 및 전달 closed #1234 [DM648_Debug_trunk]

보안 사항

적용

저장소 수준에서 변경 추적

개요 | 작업내역 | 일감 | 새 일감만들기 | Dashboard | Gantt 차트 | 문서 | 위키 | 파일 | 저장소

code_integration

통계 | 개정판:

이름	크기	개정판	마지막 수정일	저자	설명
branches		1636	2일	문 수진	(Issue #1234)
tags		1615	6일	Admin	[DM648_Debu
trunk		1637	2일	문 수진	(Issue #1096)

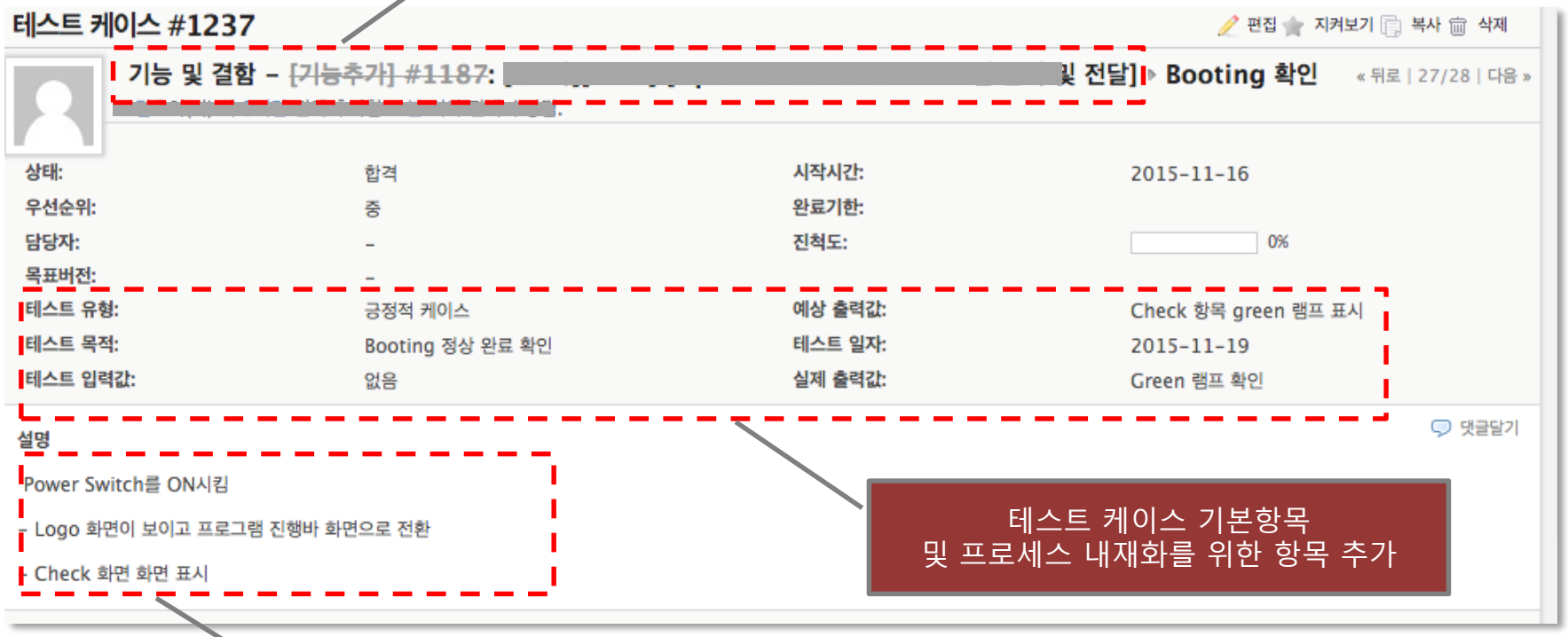
보안 사항

352

단위 테스트 프로세스 개선 및 도구 적용

개발 체크리스트를 테스트 케이스로 개선하고 Redmine 일감으로 적용 및 요구사항 연결

요구사항 #1187과 연결을 통해
요구사항-테스트 추적성 관리



테스트 케이스 기본항목
및 프로세스 내재화를 위한 항목 추가

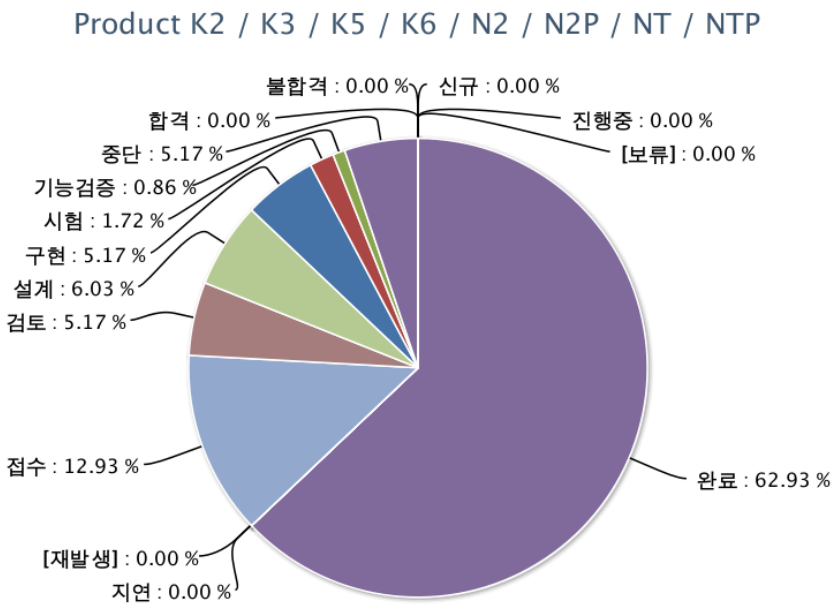
상세 테스트 단계 작성

353

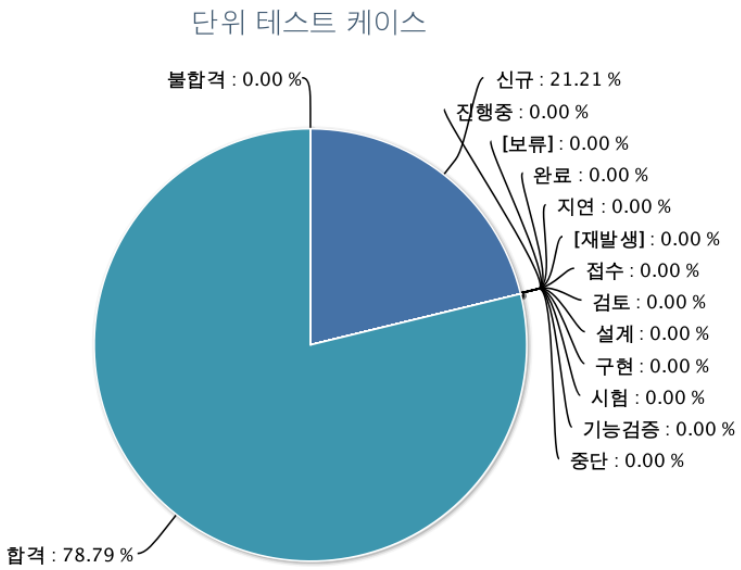
Redmine을 이용한 업무/품질 관리

요구사항 개발 유형/상태/테스트 결과를 Redmine 통계관리에서 확인 및 진행 관리

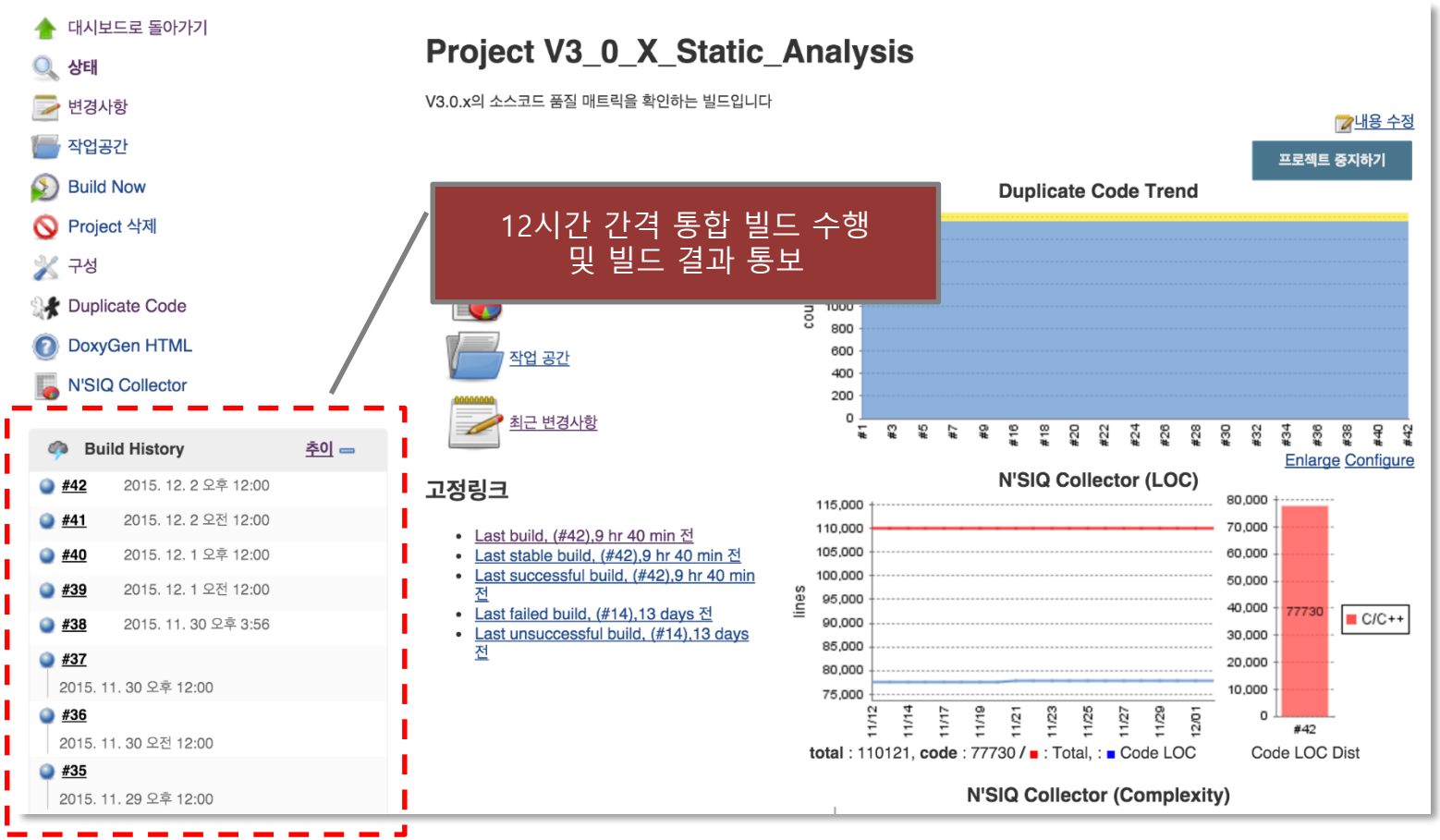
요구사항 진행 상태 확인



테스트 결과 및 진행 상태 확인



핵심 3개 프로젝트 빌드 환경 표준화 및 일 단위 통합 빌드 수행



소프트웨어 품질 향상을 위한 Jenkins 내 소스코드 분석 시스템 구축 및 품질 관리

소스코드 문제점 - 중복 항목 검사 및 조치 (CPD)

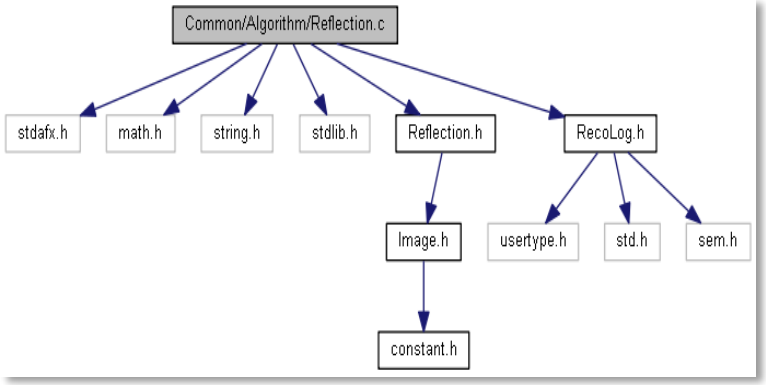
Duplicate Code - Normal Priority

Details

Folders | Files | Warnings | Details

Source Folder	Total	Distribution
Common	10	
Common/Algorithm	23	
Common/Algorithm/smu/SNReco	2	
Common/Algorithm/smu/UBS_KeyClub	5	
Common/MG	10	
Common/Socket	22	
Total	72	

불필요한 의존성 분석 및 조치 (Doxygen)

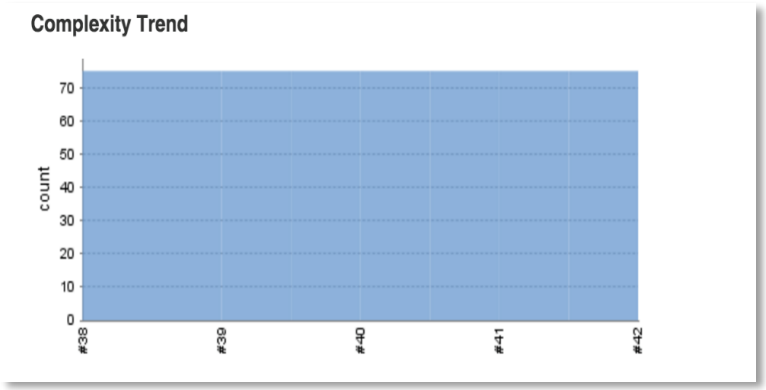


소스코드 문제점 – 높은 복잡도 검사 및 조치 (NSIQ)

Complexity Breakdown by Directory

Type	Directory	TotalLoc	CodeLoc	Over 30
Directory	[Common]	34579	23955	25
Directory	Algorithm	26752	19404	30
Directory	Algorithm\smu\CNY	1073	966	0
Directory	Algorithm\smu\DSP	633	464	0
Directory	Algorithm\smu\SNReco	3337	2471	7
Directory	Algorithm\smu\source	1466	1127	0
Directory	Algorithm\smu\UBS_KeyClub	1106	891	1
Directory	Config	5260	3772	0
Directory	dm648\dm648Include	9142	5120	0
Directory	dm648\dm648Src	1341	577	0

소스코드 품질 수준 추세 확인 (NSIQ)



4 성과 분석

자체 지표 성과 분석

■ 정량적 성과

지표항목	성과지표	목표 수준	초기 수준	최종 성과	분석
요구사항 관리	요구사항 변경율	20%	측정 불가	**19.2%**	- 메일/전화로 유입되던 요청 건의 Redmine 일원화 - 변경 요청 건의 진행 관리 및 변경 영향도 파악
제품 품질 검증	정적분석 이행율	30%	0%	**30%**	- 핵심 3개 프로젝트의 순환복잡도, 중복, 의존성 검사 - (참조) 사업 성과 분석의 "소스코드 품질 준수율"
	소스코드 품질 준수율	100%	측정 불가	**100%**	- 적용 이후 추가된 함수의 품질 수준 저하 없음 (순환복잡도, 중복코드, 의존성)
프로세스 개선	재작업 감소율	5건 /월	0건	**37건 /11월**	- 메일/전화로 유입되던 제품 이슈 건의 Redmine 일원화 - 문제 분석 및 해결 주체의 명확화로 재작업 감소
테스트 수준	단위 테스트 결함 유출율	50%	측정 불가	**46.2%**	- Redmine 결함 등록으로 결함 관리 가능 - 단위 테스트 수행으로 문제 조치비용 감소

■ 정성적 성과

- 개발팀 내의 프로세스 정착 및 개발 지원 도구 사용의 필요성 인식
  -> 전사 프로세스 및 도구 사용 확대 요구 증대
- 경영층의 SW 개발 프로세스 중요성 인지 및 지원 확대 검토
  -> 지속적 개선 및 전사 확산의 스폰서쉽 확보
- 소스코드 품질 기반의 개발 문화 확산
  -> SVN, Jenkins 활용 활성화 및 정적분석 도구 사용을 통한 소스코드 품질 중요성 인식 확대

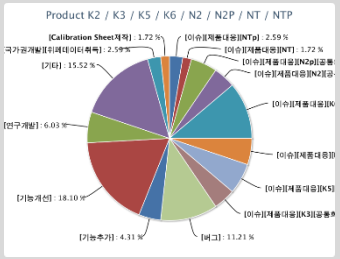
5 향후 계획

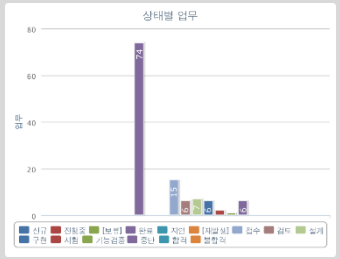
기산전자 2016 예상 시스템 구성도

KCSE 2016 논문집 18권 1호

[임원 전용]

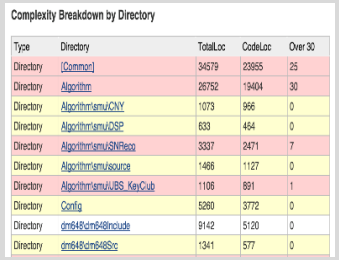
전략적 의사결정 지원 시스템

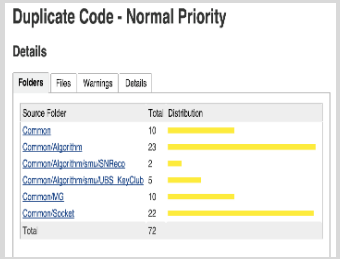




[연구원/개발자 전용]

소프트웨어 품질 포탈

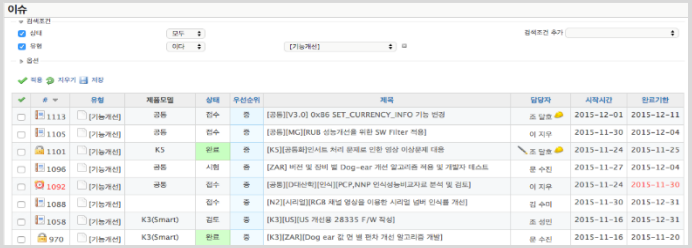




데이터 수집/분석
(Process Mining)

- 소프트웨어 품질실패비용[3]
- 코드 품질 수준

요구사항 관리 시스템 (Redmine/JIRA)



가상화 기반
소프트웨어 빌드/품질 분석 시스템
(Jenkins/CPD/NSIQ/CodeSonar/VMWare)

소프트웨어 자산 관리 시스템
(SVN/Mercurial)

단일 접수 창구
(전사 확산)

장애 | 신규개발 | 기능개선 | 아이디어

360

6 임베디드 SW 개발의 Visualization 적용 로드맵 제안

임베디드 SW 개발 조직의 Visualization 적용 어려움

다수 조직의 SW Visualization 컨설팅 및 적용을 통해 다음의 어려움을 발견할 수 있었음

- 참조할 수 있는 임베디드 분야 SW Visualization사례 부족
- Redmine, Jenkins 등을 사용하기 위한 웹 환경에 익숙하지 않음
- SW 품질을 위한 도구가 대부분 상용 및 고가
- 조직 내 관심 인력 또는 외부 지원 인력을 찾기 어려움
- SW 공학 기술에 대한 관심 부족
- 표준 빌드 환경 구성의 어려움

임베디드 SW Visualization 적용 로드맵 제안

KCSE 2016 논문집 18권 1호

임베디드 SW 개발 조직의 Visualization을 위해 5단계의 로드맵을 제안함

수준	수행 항목	활용 가능 도구	
		오픈소스	상용
Level 5	- PLM과 ALM의 통합(Item으로서 SW Binary 관리)	-	-
Level 4	- 가상 빌드 환경에서의 클린 빌드 수행	-	VMWare
Level 3	- 이슈 추적 시스템과 버전관리 시스템 연동 - 지속적 통합 시 정적분석/테스트 수행 - ALM(SW Visualization) 환경 구축 완료	Unity CPPUnit CPPCheck PMD-CPD NSIQ Doxygen	VectorCAST QAC CodeSonar PolySpace Lattix Imagix4D
Level 2	- 이슈 추적 시스템 사용 내재화 - 일 단위 지속적 통합 빌드 수행	Redmine Trac Jenkins	JIRA Bamboo
Level 1	- 버전 관리 시스템 사용 내재화	SVN git Mercurial	Perforce

참고문헌

[1] Martin Fowler, "Continuous Integration", http://www.martinfowler.com/articles/originalContinuousIntegration.html

[2] 한동준, "오픈소스를 활용한 임베디드 SW 지속적 통합 방안 및 사례 공유", 소프트웨어 안전성 보증을 위한 기술 세미나, 2015

[3] 한동준 외, "소프트웨어 개발 프로젝트에서의 품질비용", 한국 소프트웨어공학 학술대회, 2009

인지적 분석을 통한 임상 의사결정 지원 시스템 인터페이스 설계

고동균, 김유찬, 윤완철

한국과학기술원 지식서비스공학과
대전광역시 유성구 대학로 291 305-701
kodonggyun@kaist.ac.kr yoochana@kaist.ac.kr wcyoon@kaist.ac.kr

요약: 임상 의사결정 지원 시스템은 의료 종사자들에게 상황에 맞는 적절한 정보를 제공하고 의사결정을 돕는 시스템이다. 성공적인 임상 의사결정 지원 시스템의 개발을 위해서는 적절한 정보가 인터페이스에 효과적으로 표현되어야 한다. 본 연구에서는 인지적 직무분석 체계를 사용하여 의학 전문인의 진단 직무를 분석하고, 사용자의 가능한 전략과 정보요건, 그리고 적합한 제시 방법과 시기, 위치 등을 결정하였다. 또한 진단 전략 분석을 통해 사용자의 진단 전략 흐름과 시스템을 사용할 때의 흐름이 유사하도록 설계하였다. 전문가 평가를 진행하여 제안된 인터페이스의 유용성과 만족도를 평가하였다. 전문가 평가 결과는 인지적 분석을 통해 제안된 인터페이스 구성이 사용자에게 요구되는 지식을 최소화하고 인지적 사용성 문제를 해결 할 수 있다는 것을 보였고 향후 관련 연구에도 적용될 수 있음을 보여주었다.

핵심어: 의사결정 지원 시스템, 시스템 인터페이스, 인지부담, 혈액검사 진단, 진단 전략, 직무 분석, 정보요구사항

1. 서론

과학, 정보기술의 발달로 인해 혈액검사, 당뇨병, 심장병 등 다양한 영역에서 의료 종사자들의 의사결정을 돕는 시스템인 임상 의사결정 지원 시스템(Clinical Decision Support System)에 대한 연구가 진행되고 있다[2]. 임상 의사결정 시스템은 기계학습 알고리즘을 이용하여 진단예측 확률을 높이는 비-지식기반 임상 의사결정 지원 시스템(NonKnowledge-Based Clinical Decision Support System)[3]과 의사결정에 필요한 정보를 원활한 상호작용을 위해 효율적으로 제공하는 지식기반 임상 의사결정 지원 시스템(Knowledge-Based Clinical Decision Support System)[4] 으로 나눌 수 있다.

의학 분야에서는 잘못된 의사결정으로 발생하는 피해가 크고 즉시 발생한다[10, 12]. 또한 방대한 양의 의학서적으로 인해 의료 종사자들이 기계학습에 대한 불안함을 느끼고 있어 새로운 의사결정 시스템 도입에 대해 보수적이다[1]. 따라서 비-지식기반 임상 의사결정 지원 시스템보다는 지식기반 임상 의사결정 지원 시스템의 다양한 시스템 요소에 대한 체계적인 접근이 필요하다.

기존 연구에 따르면 성공적인 임상 의사결정 지원 시스템을 개발하기 위해서 사용자의 인지부담을 줄이고 사용자 태스크 전략에 맞는 인터페이스의 중요성이 강조되고 있다 [8, 9, 11]. 현재 많은 분야에서 인지직무분석(Cognitive Work Analysis)를 통해 직무수행의 목적을 위해 필요한 인지적 지식구조, 인지적 처리과정에 대해 분석하고 Rasmussen 의 사람의 4 가지 진단전략(mental strategy), 의사결정 사다리 모델(Decision Ladder Model)과 SRK 모델을 통해 사용자의 태스크 전략과 직무 수행 과정에서의 인지부담을 분석하여 시스템 인터페이스에 반영하고 있다[5, 6, 7].

따라서 본 연구에서는 임상 의사결정 지원 시스템에서 사용자가 요구하는 지식을 바탕으로 인터페이스의 정보의 표현 및 구성 방법을 제안하여 사용자 인지부담을 줄이고 사용성 문제를 해결하는 것을 목적으로 한다. 의학 전문가의 업무 흐름과 직무분석을 통해 찾은 기본 정보요구 사항과 인지적 전략 흐름 분석을 바탕으로 인터페이스를 설계하였고 전문가 설문 및 인터뷰를 통해 시스템 인터페이스를 평가하고 효율성을 증명하였다.

2. 사용자 진단 분석

2.1 사용자 직무 분석

의료 종사자 4 명의 직무수행을 관찰하고 이후 인터뷰를 통해 임상 진단 중에서 혈액검사 진단 직무를 파악하였다. 진단의 과정은 혈액검사 결과가 나오면 진단 가설이 생기고 그것에 대해 평가를 진행하고 마지막으로 소견이 나오는 과정이다. 작업 영역 분석(Work Domain Analysis)을 추상계층(Abstract Hierarchy) 분석을 통해(그림 1) 목적(Functional Purpose), 추상적 기능(Abstract Function), 일반화된 기능(Generalized Function), 물리적 기능(Physical Function), 물리적 형태(Physical Form) 등 5 가지 단계로 작업영역을 나눠 시스템 목표 달성을 위한 기능들의 계층적인 구조를 확인할 수 있다.

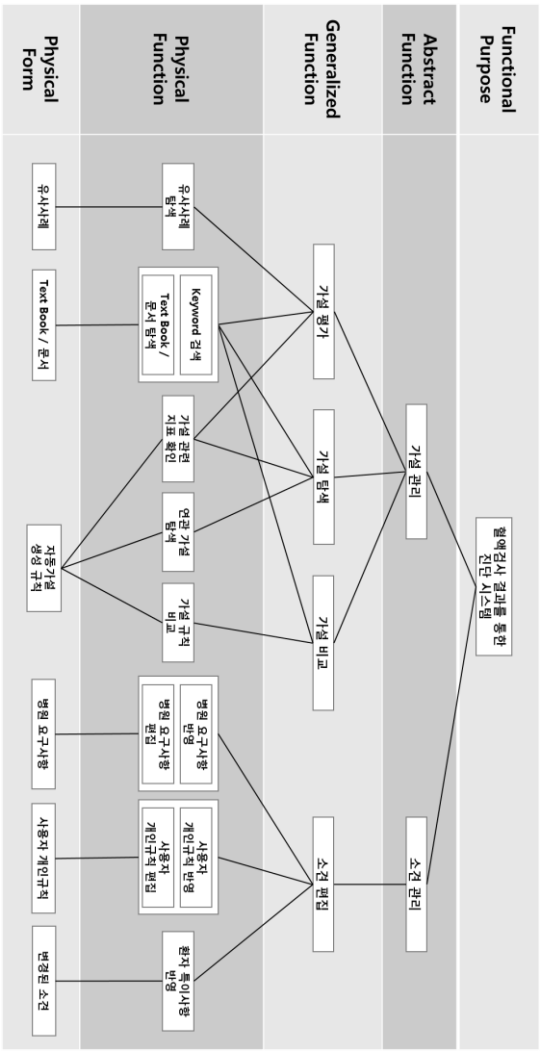


그림 1 혈액검사 진단 추상 계층 분석

인지적 직무 분석과 진단 전략 분석을 위해 데이터 흐름 다이어그램(Data Flow Diagram)을 사용자 직무 부석과 추상계층 분석을 바탕으로 만들 수 있고 이것을 통해 필요한 정보와 프로세스 흐름에 따른 정보의 변환을 파악할 수 있다(그림 2).

2.2 인지적 직무 분석

인지적 직무 분석을 위해 직무들을 계층화하여 태스크와 서브 태스크로 표현하는 계층적 직무 분석(Hierarchical Task Analysis)를 하였다. 계층적 직무 분석은 직무 분석에 자주 사용되는 방법으로 계획(Plan)을 통해 직무의 흐름을 쉽게 파악할 수 있다.

계층적 직무 분석을 통해 기본적으로 직무에 필요한 정보와 태스크 수행을 위한 요구사항을 정리한 기본 정보 요구사항을 도출할 수 있고 인지적 진단 전략 분석과 함께 태스크 별 정보요구분석을 할 수 있다.

혈액검사 진단을 위한 직무의 상위 개념으로는

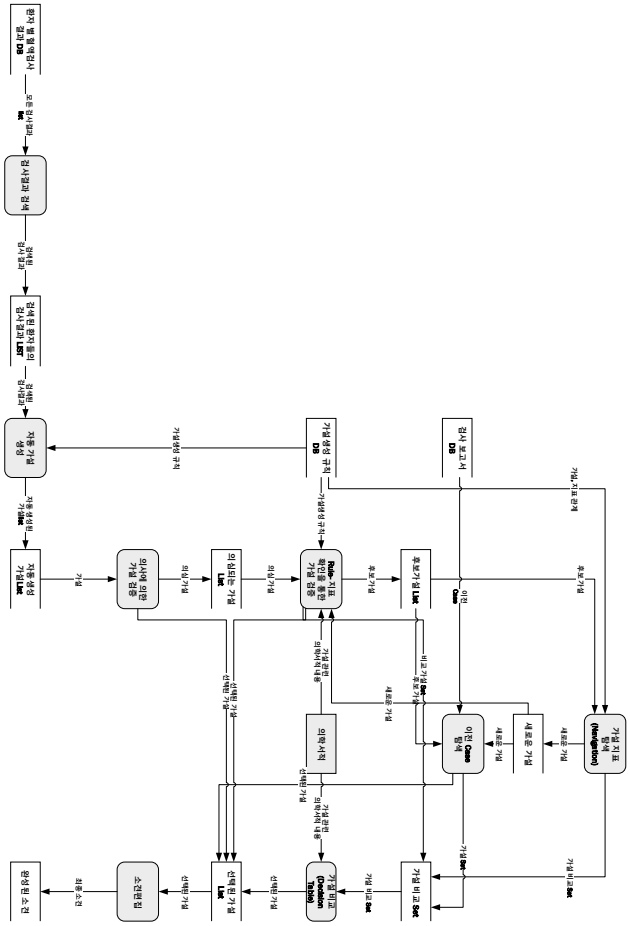


그림 2 혈액검사 진단 데이터 흐름 다이어그램

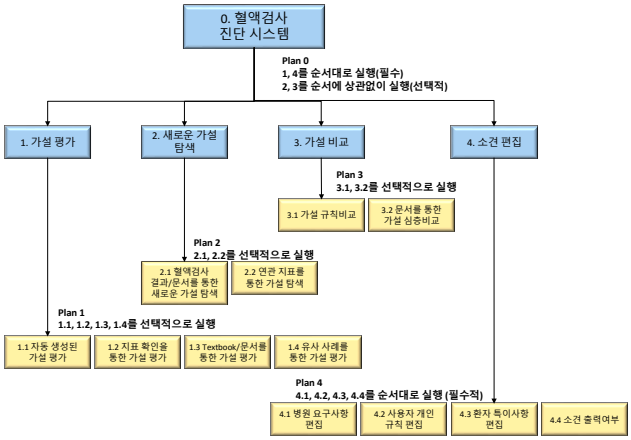


그림 3 혈액검사 진단 계층적 직무 분석 – 상위개념

가설 평가, 새로운 가설 탐색, 가설 비교, 소견 편집 4 가지 직무가 있다(그림 3). 이때 가설 탐색과 가설 비교는 가설 평가의 결과에 따라 선택적으로 수행된다.

가설 평가는 간단한 가설평가부터 지표, 관련 의학 문서, 유사사례를 통한 복잡한 평가과정까지 1.1 에서 1.4 로 이어진다(그림 4). 이때 각각의 정보들은 한눈에 알아보기 쉽게 표현되어야 하고 평가할 진단

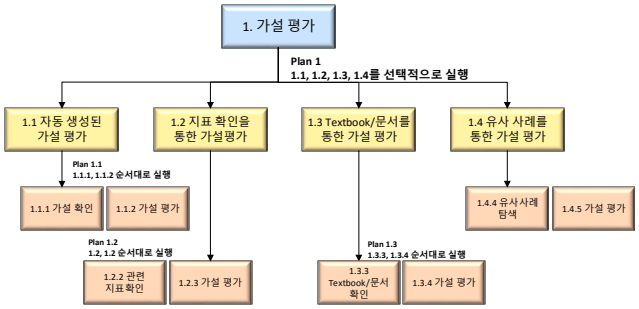


그림 4 계층적 직무 분석 - 가설평가

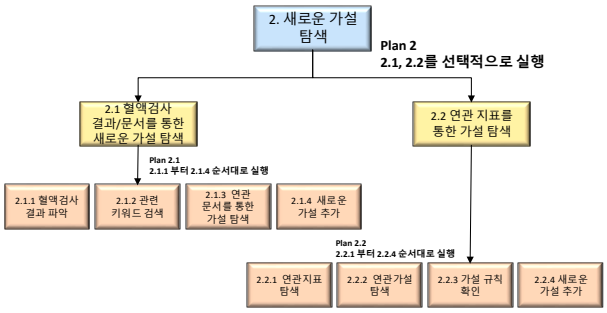


그림 5 계층적 직무 분석 - 새로운 가설 탐색

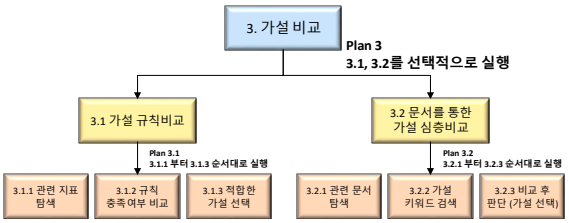


그림 6 계층적 직무 분석 - 가설 비교

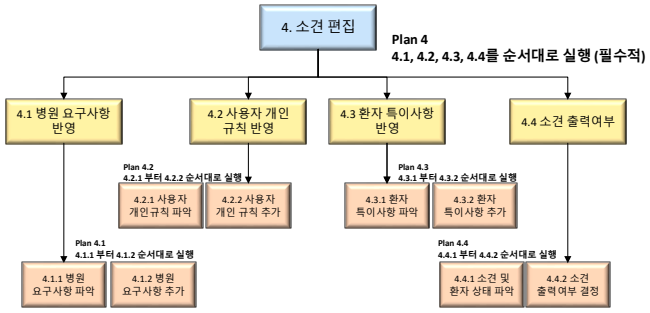


그림 7 계층적 직무 분석 - 소견 편집

가설과 관련된 자료가 제공되어야 한다. 또한 사용자가 추가 검색을 실시 했을 때 신뢰도 높은 자료를 제공할 수 있어야 한다.

새로운 가설 탐색에는 2.1 혈액검사 결과/문서를 통한 새로운 가설 탐색과 2.2 연관 지표를 통한 가설 탐색이 있다. 새로운 후보 가설을 찾는 단계이고 필요한 직무를 선택적으로 수행한다(그림 5). 2.1 의 경우 검사결과로부터 새로운 가설을 탐색하는 과정으로 환자의 모든 검사결과를 볼 수 있어야 하고 사용자가 추가로 정보를 검색할 수 있게 해야 한다. 2.2 의 경우 기존의 가설을 바탕으로 새로운 가설을 탐색하는 과정으로 기존 가설에 연관된 지표들을 선택하면 선택한 지표와 연관된 다른 가설들을 단계별로 모두 보여주어야 한다. 연관된 지표를 선택하는 과정에서 그 지표의 정상/이상 상태를 심각한 정도와 함께 보여주면 사용자의 인지부담을 줄일 수 있다.

가설 비교는 기존 가설과 후보 가설이 있을 때 서로 비교하고 평가해서 어떤 가설이 환자에게 더 적합한지 정하는 직무이다(그림 6). 규칙 충족 여부를 비교하는 과정인 3.1 가설 규칙비교는 규칙이 복잡해질수록 사용자가 알아보기 쉽도록 Graphical 한 방식을 사용하여 보여주어야 한다. 또 Decision Table 을 통해 규칙과 지표를 한번에 보여주는 방식도 있을 수 있다. 이런 과정을 통해 해결되지 않을 경우 3.2 관련 문서를 통한 심층 비교 직무를 수행한다. 전의 직무들과 같이 신뢰도 높은 자료를 알아보기 쉽게 제공할 수 있어야 한다.

소견 편집은 앞의 직무들을 통해 가설이 확정 된 후 환자에게 실제로 제공되는 소견을 편집하는 직무이다. 4.1 병원 별 요구사항이나, 4.2 사용자 별 개인 규칙, 4.3 환자 특이사항을 반영하고 4.4 소견출력여부를 판단하는 직무로 이루어져 있다. 개인적인 변경사항은 진단 실수로 이어질 수 있기 때문에 개인적인 변경사항을 따로 관리할 수 있는 시스템, 인터페이스가 필요하고 변경하기 전의 규칙을 항상 보여줘서 잘못된 변경사항을 등록하지 않도록 확인, 유도하여야 한다.

2.3 인지적 진단 전략 분석

진단 전략 분석은 사용자가 실제로 시스템을 사용하면서 행하는 인지적 의사결정 과정을 분석하여 진단 전략 별로 정보들간의 관계를 파악하는 과정이다. 전략 별로 필요한 정보의 종류와 구성방식을 시스템의 지식과 유사하게 구성하여 사용자의 인지부담을 줄일 수 있다. 사용자, 인지적 직무 분석을 바탕으로 사람의 혈액검사 진단의 의사결정을 Rasmussen 의 의사결정 사다리 모델에 적용하였다(그림 8).

혈액검사 결과를 바탕으로 가설을 생성하고 의심이 되면 가설평가를 한다. 가설 관련 지표 확인으로 평가가 되지 않는다면 다른 환자의 결과나 관련문서를 직접 찾아가면서 가설을 평가하고 새로운 가설을 탐색한다. 새로운 가설을 다시 관련 지표, 다른 환자의 결과, 관련문서를 통해 확인하고 평가하는 과정을 반복하고 그 과정에서 여러 개의 가설을 비교하는 과정을 통해 최종 가설을 선택한다. 모델에서 상위 전략으로 올라갈수록 사용자의 인지부담이 커져 많은 지식을 바탕으로 한 생각과 오랜 시간에 걸친 의사결정을 요구한다. 따라서 인지적 사용성 문제가 발생하고 기술기반 행동(Skill-Based Behavior), 규칙기반 행동(Rule-Based Behavior), 지식기반 행동

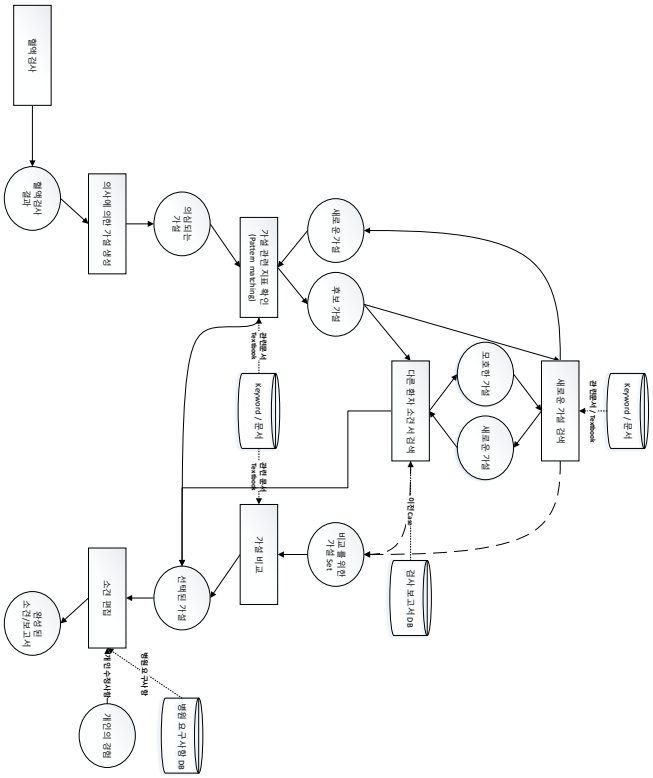


그림 8 혈액검사 진단 의사결정 사다리 모델

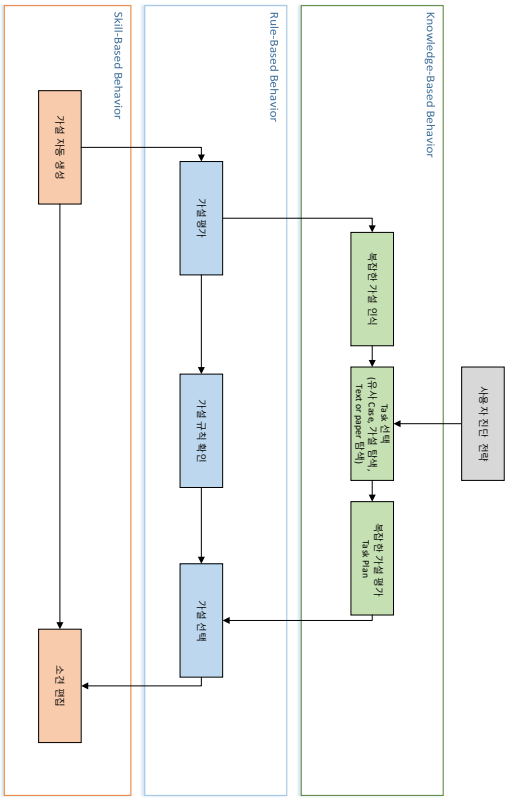


그림 9 혈액검사 진단 SRK 모델

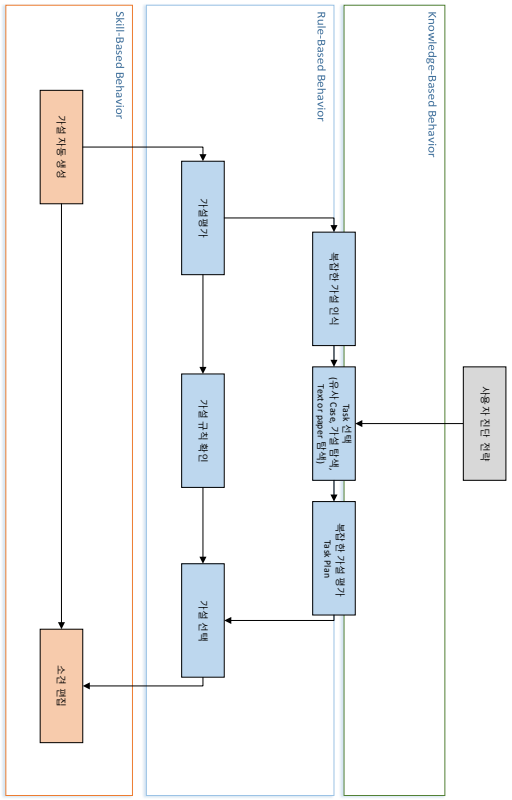


그림 10 연구진행 후 시스템 사용자 SRK 모델

(Knowledge-Based Behavior)으로 분류하여 시스템 설계에 참고, 반영해야 한다(그림 9).

혈액검사 진단에서 기술기반 행동은 가설을 보자마자 평가가 바로 이루어져서 소견 편집을 하는 행동이고, 규칙기반 행동은 가설규칙을 찾아보고 확인한 다음 가설을 선택하고 소견 편집을 하는 행동이다. 마지막으로 지식기반 행동은 유사 사례, 새로운 가설, 관련 문서 등을 탐색하고 지식을 찾아 가설을 선택하고 소견을 편집하는 행동이다. 실제 직무를 수행하는 환경에서는 대부분이 기술기반 행동으로만 수행되고 있고 상위 행동들은 이용 빈도가 낮다는 이유로 시스템에서 정보가 제공되지 않고 있다. 따라서 그 과정이 복잡하고 힘들어서 사용자의 인지적 부담이 크기 때문에 진단 실수로 이어질 확률이 높다. 본 연구에서는 모든 행동들의 인지부담을 줄이는 것을 목표로 하지만 특히 지식기반 행동에 해당하는 직무 수행에 필요한 정보와 기능을 제공하여 정보 제공 및 사용성 요구 문제를 해결하고 인지 부담을 줄여 규칙기반 행동에 가까워질 수 있도록 하는 것이 주 목표이다(그림 10).

혈액검사 진단 의사결정 사다리 모델과 SRK 모델을 반영하여 사용자의 인지전략과 정보흐름과 유사하게 시스템 의사결정 사다리 모델을 설계하였고 그 과정에서 Rasmussen 의 4 가지 진단전략을 바탕으로 시스템 기능을 구성하였다. 이를 통해 사용자가 기존에 사용하던 진단전략으로 시스템을 적은 인지부담

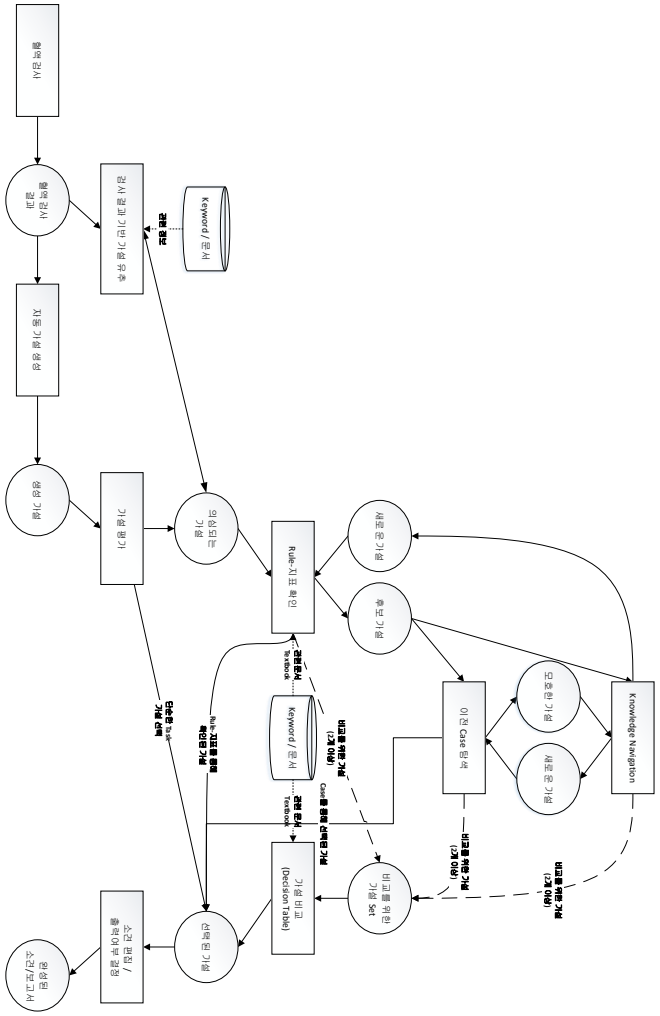


그림 11 혈액검사 진단시스템 의사결정 사다리 모델

Rasmussen 진단전략	시스템 기능	특징 및 설명
Pattern Recognition	Rule-지표 확인	규칙과 지표를 확인하고 가설을 평가
Decision Table Search	가설 비교 (Decision Table)	Decision Table 을 만들어 가설을 비교하는 단계
Hypothesis and test Search	가설 추가, 가설 평가	새로운 가설을 추가하고 다양한 방식으로 평가
Topological Search	Knowledge Navigation	관련 지표를 확인하면서 새로운 가설을 탐색

표 1 진단전략에 따른 시스템 기능

으로 사용할 수 있어 진단오류를 줄일 수 있다(그림 11, 표 1).

직무 분석과 진단 전략 분석을 바탕으로 직무 수행에서 5 가지의 자주 사용하는 직무를 추출하여 혈액검사 진단 시스템 의사결정 사다리 모델에 반영되는 모습을 나타낼 수 있다.

직무 1: 사용자가 자동으로 생성된 가설을 신뢰하고 선택하거나 지표 및 키워드를 확인하고 가설을 평가(그림 12).

직무 2: 가설과 상관없이 혈액검사 결과를 기반으로 검색된 문서를 통해 새로운 가설을 추가하여 평가(그림 13).

직무 3: 규칙에 관한 지표 확인과 이전 사례 탐색을 통해 가설을 선택(그림 14).

직무 4: 이전 사례를 탐색하고 적합한 가설을 찾기 위해 지표와 연관된 가설 Knowledge Navigation 을 통해 탐색(그림 15).

직무 5: 앞에서 언급된 직무들을 거친 후 복수의 경합하는 두 가설을 비교(그림 16).

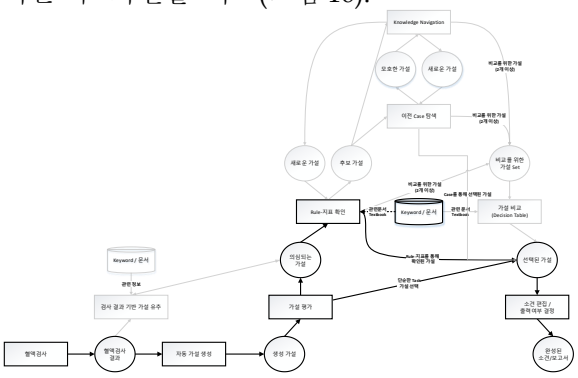


그림 12 의사결정 사다리 모델 – 직무 1

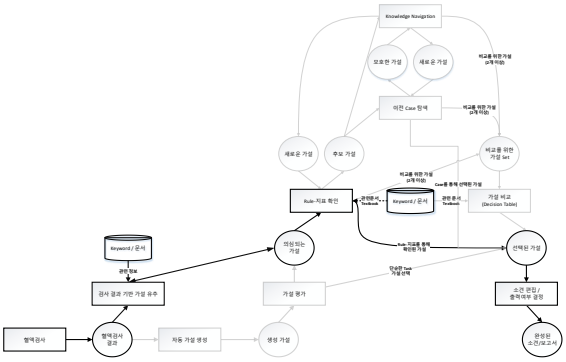


그림 13 의사결정 사다리 모델 - 직무 2

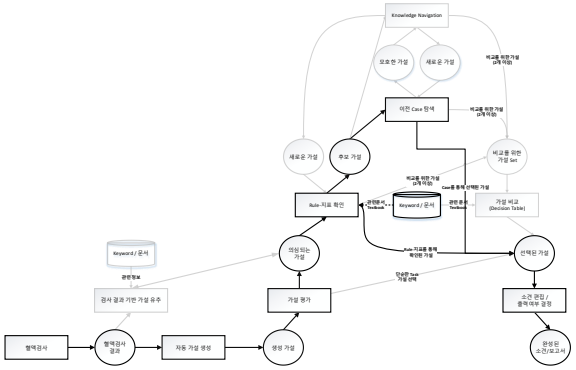


그림 14 의사결정 사다리 모델 - 직무 3

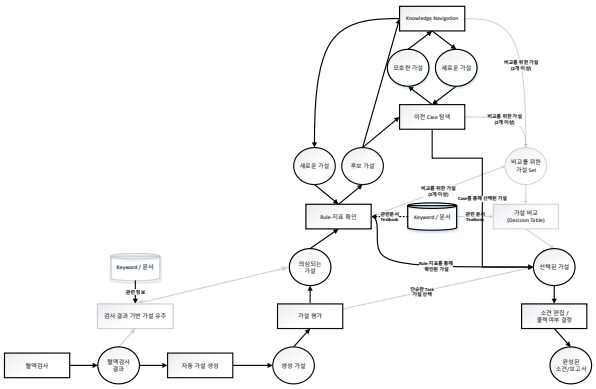


그림 15 의사결정 사다리 모델 - 직무 4

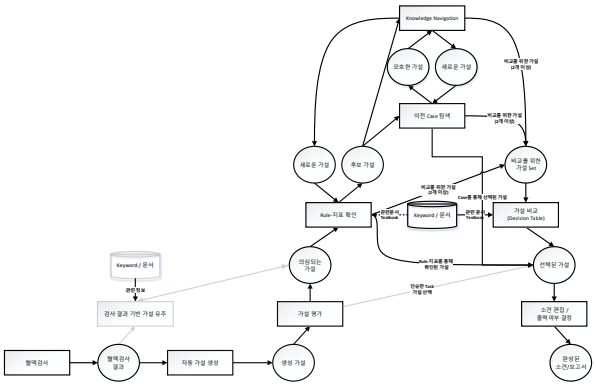


그림 16 의사결정 사다리 모델 - Task 5

3. 직무 중심 의사결정 지원 시스템 설계

3.1 직무 별 정보 내용과 표현 방식 도출

5 가지의 직무들은 일반적인 기능(Function)들의 집합으로 이루어진다. 그 관계는 기능-직무 연관 분석을 통해 각 직무에서 어떤 기능들을 사용하고 어떤 정보들을 요구하는지 분석할 수 있다(표 2). 시스템의 일반적인 기능들은 앞의 분석내용을 바탕으로 기본 가설 평가, 검사결과 기반 가설 유추, 규칙-지표 확인, 이전 사례 탐색, Knowledge Navigation, 가설 비교(Decision Table), 소견 편집으로 구성된다.

직무 1 에서 기본 가설 평가의 경우 자동 가설 규칙 및 관련 지표에 대해서 쉽게 파악할 수 있도록 그래프 방식으로 표현하여야 하고 각각의 지표들도 색을 사용하여 심각성을 표현해 주어야 한다.

직무 2 에서는 검사결과 기반 가설 유추가 중요하다. 가설 유추 외에도 관련 지표와 관련 키워드를 추천해주고 검색된 정보들을 효율적으로 표현해주어야 한다.

직무 3 에서는 이전 사례 탐색이 중요하다. 현재 환자와 유사한 이전의 사례를 담당의사, 검사 지표, 평가결과, 날짜 등과 함께 제공하고 신뢰하는 담당의사나 최근 사례 평가를 찾을 수 있어야 한다.

직무 4 에서는 Knowledge Navigation 이 중요하다.

기능/직무	직무 1	직무 2	직무 3	직무 4	직무 5
기본 가설 평가	O		O	O	O
검사결과 기반 가설 유추		O			
규칙-지표 확인	O	O	O	O	O
이전 사례 탐색			O	O	O
Knowledge Navigation				O	O
가설 비교 (Decision Table)					O
소견 편집	O	O	O	O	O

표 2 기능-직무 연관 분석

복잡한 지식을 그래프 방식으로 한눈에 알 수 있게 표현해 주어야 하고 선택된 가설 또는 지표와 관련된 내용들을 효과적으로 보여주어야 한다.

직무 5 에서는 가설 비교가 중요하다. 가설을 쉽게 비교할 수 있도록 표를 이용하여 규칙에 만족하는지 직관적으로 나타내주어야 한다.

3.2 인터페이스 설계

사용자 직무분석과 직무 별 분석을 바탕으로 기능별 상세 정보 요구사항과 표현 방법을 도출할 수 있고 이것을 바탕으로 인터페이스 설계를 진행하였다(표 3). 필요 정보는 기능을 사용할 때 제공되어야 하는 정보들을 의미하고, 정보 특성 및 표현 방식은 필요 정보가 인터페이스에서 어떤 식으로 나타내져야 하는지를 의미한다.

	필요 정보	정보 특성 및 표현 방식
기본 가설 평가	1. 자동 생성 가설 및 관련 지표	- 관련 규칙에 대한 그래프 표현 - 관련된 지표만을 선택적으로 표시
	2. 가설과 소견에 관련된 정보	- 가설과 소견을 함께 볼 수 있도록 구성 - 가설 평가에 필요한 연관된 지표
검사결과 기반 가설 유추	3. 환자 혈액 검사 결과	- 위험, 주요 지표 표시 (색깔의 차이를 이용하여)
	4. 기본 키워드, 추천 키워드	- 검사결과와 연관된 키워드 - 환자의 특성을 반영
	5. 검색 결과 (문서 정보)	- 정확도를 기반으로 신뢰성 - 출처 표시
	6. 가설 추가 / 관련 정보	- 다른 가설 정보 검색
규칙-지표 확인 / 문서 검색	7. 지표에 대한 심각성 및 상세 내용 표시	- 주요 지표를 색으로 표현 - 그래프 형식으로 심각성 정도 표시 - 가설과 연관된 지표만을 선택적으로 표시

	8. 기본 키워드, 추천 키워드	- 가설과 연관된 키워드 - 가설과 연관된 지표의 키워드도 반영
	9. 검색 결과 (문서 정보)	- 정확도를 기반으로 신뢰성 - 출처를 반드시 표시
이전 사례 탐색	10. 유사 사례	- 진단 결과 표시 - 과거 환자의 지표와 정보 - 유사한 정도에 따라 정렬 (판단 근거)
	11. 사용자 정보	- 신뢰성이 높은 사용자 자료보기
Knowledge Navigation	12. 가설 연관 지표	- 관련 규칙에 대한 그래프 표현 - 지표의 심각성을 색으로 표현
	13. 연관 가설 / 연관 지표	- 그래프 구조로서 지표와 가설 간의 탐색이 유용한 Knowledge Navigation 을 구축
	14. 가설 정보	- 가설 선택 시 가설에 대한 정보 출력
	15. 가설 추가	- 검색 후 가설 목록에 추가 (드래그 인)
가설 비교 (Decision Table)	16. 비교 가설 정보	- 가설 규칙 그래프 형식 표현
	17. 가설 비교 Table	- 환자 지표를 가설 비교 테이블에 표시 (사용자가 쉽게 파악할 수 있도록) - 색과 체크를 사용하여 표현
	18. 지표에 대한 심각성 및 상세 내용 표시	- 주요 지표를 Color 로 표현 - 그래프 형식으로 심각성 정도 표시 - 가설과 연관된 지표만을 선택적으로 표시
소견 편집	19. 병원 별 요구사항	- 요구사항이 있을 경우 표시를 통해 사용자에게 미리 알림.
	20. 기본 규칙 / 개인 규칙 비교, 개인 별 소견	- 가설 규칙 그래프 형식 - 개인 규칙의 소견을 리스트로 보여주고 쉽게 변경할 수 있도록 구성
	21. 환자 특이사항	- 반영이 필요 할 경우 표시 - 사용자 판단에 의해 소견 수정가능
	23. 소견 출력 여부	- 소견 출력하지 않을 경우 연하게 표시

표 3 시스템 기능별 필요 정보 특성 및 표현 방식

시스템 기능별 필요 정보 특성 및 표현 방식을 바탕으로 실제 인터페이스를 설계하였다. 그림 속의 번호는 표 3 의 필요 정보의 번호와 동일한 것으로 인터페이스에서 필요 정보들이 어떻게 표현되었는지 확인할 수 있다(그림 17-24).

복잡한 태스크를 수행하기 위해서 사용되는 핵심 인터페이스는 그림 19 이다. 어떤 정보들이 제공되는지 한눈에 알아 볼 수 있도록 분할 디자인 방식을 이용하여 총 6 가지의 분할로 구성되어 있다. 중앙 상단에 있는 가설들 중에 관심있는 가설을 사용자가 선택하였을 때 선택된 가설과 각 가설에 대한 소견이 출력되는 우측 상단의 해당하는 소견이 하이라이트 되고, 좌측 상단에는 관련 지표와 가설 생성 규칙이 표현된다. 이 좌-우 구성은 사용자의 태스크 흐름에 맞춘 것으로 지표와 규칙을 확인한 후 가설을 확인하고 마지막으로 소견을 수정하는 과정이다. 하단에는 Knowledge Navigation 과 이전 유사 사례, 문서 검색 부분이 있는데 확대해서 더 많은 정보를 볼 수 있도록 하였다. 문제가 있거나 다른 부분을 빠르게 파악하는 것이 중요하기 때문에 그런 부분들은 색깔을 이용하여 빨간색으로 강조하였고 특히 필요 정보 7 번에서는 연관 지표를 표와 더불어 수직선을 심각성을 표현하는데 사용하여 정상에서 약간 벗어났는지 많이 벗어나 위험한 정도인지 쉽게 알 수 있도록 하였다. 다른 필요 정보들도 글로 된 표현보다는 그래프 구조를 이용하여 빠르게 파악할 수 있도록 하였다.

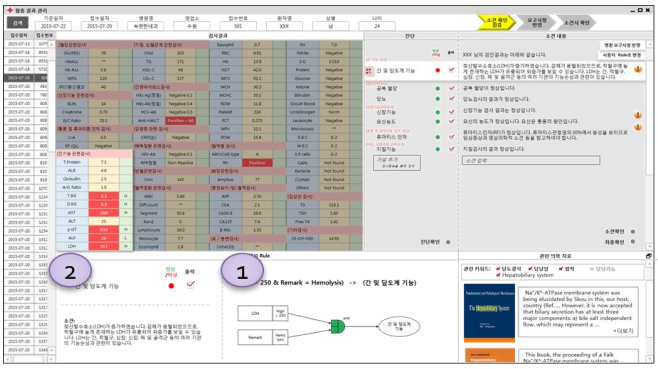


그림 17 인터페이스 - 기본 가설 평가

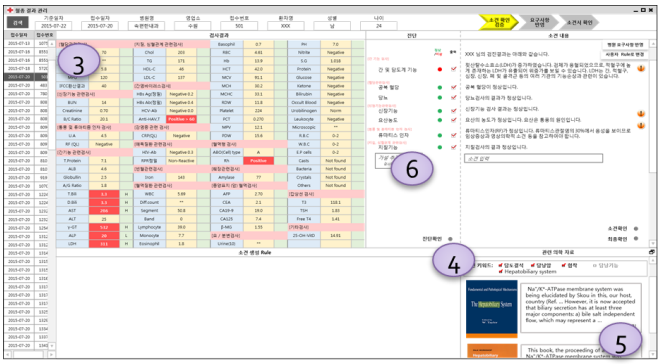


그림 18 인터페이스 - 검사결과 기반 가설 유추

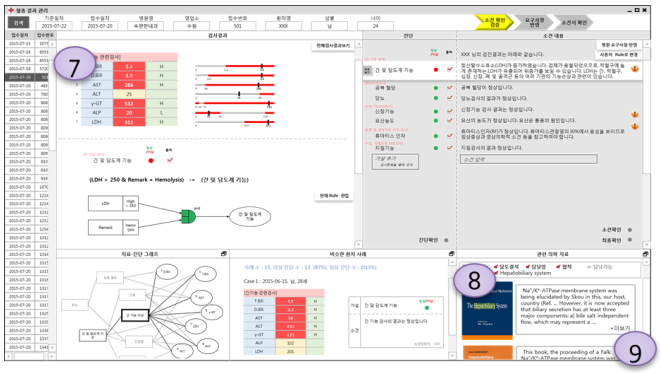


그림 19 인터페이스 - 규칙-지표 확인 / 문서검색

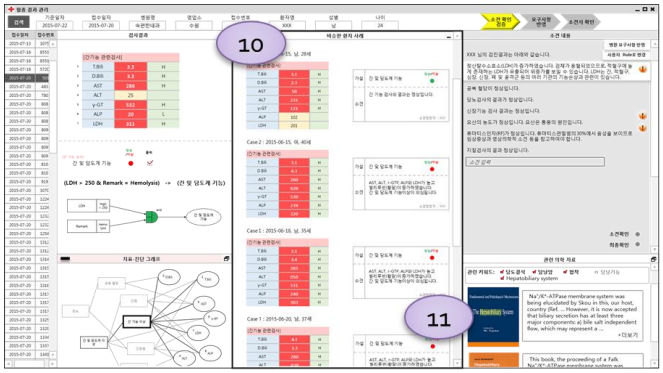


그림 20 인터페이스 - 이전 사례 검색

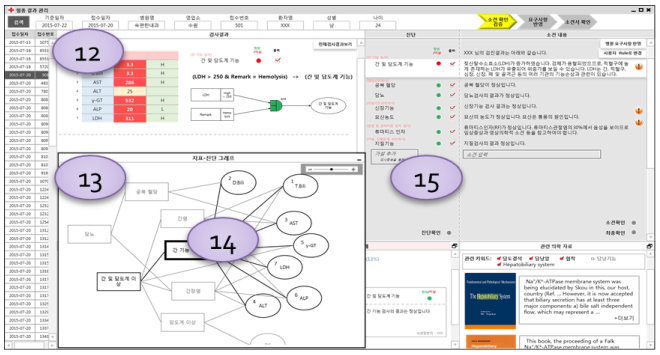


그림 21 인터페이스 - Knowledge Navigation

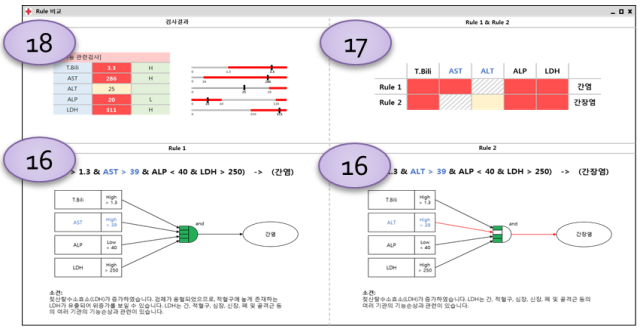


그림 22 인터페이스 - 가설 비교(Decision Table)

그림 23 인터페이스 - 소견 편집

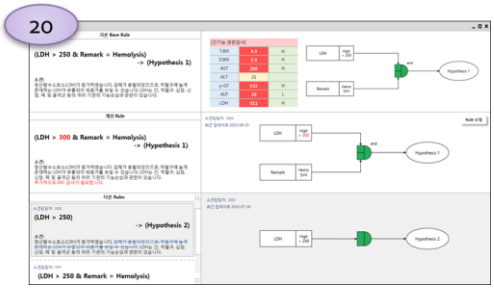


그림 24 인터페이스 - 소견 편집 - 개인 규칙

4. 평 가

시스템 인터페이스를 통해 제공되는 정보가 직무 수행에 필요한 정보인지, 표현방식이 적절한지에 대해 평가 받기 위해 평균 5 년 이상의 혈액검사 진단 경력을 가지고 있고 현재도 직무 수행중인 전문가 3 명을 대상으로 사용자 평가를 시행하였다. 전문가들이 기존에 사용하고 있던 시스템과 비교하여 7-point Likert scale 로 설문조사를 실시하였고 이후 인터뷰를 통해 정성적인 평가도 시행하였다.

4.1 정보 유용성 평가

서비스 품질 평가로 자주 사용되는 SERVPERF 모형을 사용하여 정보 유용성 평가를 세부정보 요소, 기능별로 시행하였고 종합적인 평가도 시행하였다.

세부요소 표현 방식 점수와 기능별 필요한 정보 유용성 및 표현 방식 점수는 각각 정보 요소에 하나에 대해서 제안된 인터페이스를 기존 인터페이스와 함께 제공하고 평가 받는 방법으로 6 개의 정보 구성 요소(표 4)와 7 개의 기능(표 5)에 대해 평가 받았다. 기존 인터페이스와 제안된 인터페이스의 차이 값이 3.35, 4.7 로 제안된 인터페이스의 높은 만족도를 나타낸다.

정보 요소	기존 인터페이스	제안된 인터페이스	차이 값
환자 지표	4	6.33	2.33
생성 규칙	4	6.44	3.33
진단 - 지표 연관 관계 표현	3	6	3
유사한 환자 사례	1.33	5	3.66
진단 관련 문서 제공	1.33	5.33	4
진단 비교 Table	1.33	6	4.66
평균	**2.5**	**5.85**	**3.35**

표 4 세부요소 표현 방식 점수 (SERVPERF)

정보 요소	기존 인터페이스	제안된 인터페이스	차이 값
기본 가설 평가	2.33	6.5	4.2
검사결과 기반 가설 유추	1.33	6	4.66
규칙-지표 확인 / 문서 검색	1.16	6.5	4.3
Knowledge Navigation	1.66	6.5	4.8
이전 사례 탐색	1.5	6	4.5
가설 비교(Decision Table)	1	6.66	5.66
소견 편집	2.66	6.33	4
평균	**1.7**	**6.4**	**4.7**

표 5 기능별 필요한 정보 유용성 및 표현 방식 점수 (SERVPERF)

피험자	기존 인터페이스	제안된 인터페이스	차이 값
A	1.35	6.15	4.8
B	2.1	6.5	4.4
C	2.05	6	3.95
평균	**1.83**	**6.21**	**4.38**

표 6 정보 유용성 평가 (SERVPERF)

피험자	기존 인터페이스	제안된 인터페이스	차이 값
A	1.39	6.31	4.92
B	2.05	5.97	3.92
C	2.16	6.61	4.45
평균	**1.87**	**6.03**	**4.16**

표 7 정보 유용성 평가 (W-SERVPERF)

종합적인 평가 결과는 기존인터페이스 1.83, 제안된 인터페이스 6.21, 차이 값이 4.38 으로 높은 만족도를 보였고 피험자들에게 각 기능에 대한 중요도를 조사한 후 W-SERVPERF 모형으로 계산했을 때에도 1.87, 6.03, 4.16 으로 높은 만족도를 보인다(표 6, 표 7).

4.2 인터뷰를 통한 정성적 평가

설문조사 후 각 평가항목과 전체적인 평가에 대해서 개별인터뷰를 진행하였고 이를 통해 제안된 인터페이스의 발전방향을 알아낼 수 있었다.

먼저 전체적으로 그래프 방식으로 표현한 것에 대한 반응은 “눈으로 보는 가시성이 더욱 중요한 분야인 임상과 조직에 적용하면 더 좋을 것 같다” 와 같이 긍정적인 반응이었지만 “추후 임상과 조직 등 적용 범위가 늘어날 경우 응용 가능성이 클 것으로 예상되지만, 진단 규칙의 영역을 정확하게 나누어 현재 인터페이스와 같은 깔끔한 규칙을 만들 수 있을지에 대한 의문이 존재한다” 와 같이 더 복잡한 자료를 바탕으로 했을 때 현재의 가시성을 유지할 수 있을지에 대한 우려를 얘기하였다. Knowledge Navigation 과 관해서는 좋은 기능이고 초보자의 교육용으로도 사용할 수 있을 것 같다고 하였지만 복잡함을 느껴서 실제 업무과정에서의 사용성에 대해서는 부정적이었다. 유사한 환자 사례와 관련 문서를 제공하는 것에 대해서는 보여지는 자료의 신뢰성을 강조하면서 많은 양을 보여주는 것보다 정말 필요한 자료를 제공하는 것이 중요하다고 하였고 진단 가설에 대해서는 추후 복잡한 분야에 적용될 때 진단에 대한 확률이 같이 나온다면 비교하는데 도움이 될 것이라는 의견을 제시하였다.

인터페이스에 대한 내용 외에 적응 및 교육시간에 대한 우려가 있었다. 적응 및 교육시간은 의사결정 지원 시스템이 성공하지 못하는 이유 중 하나에 속하는데 이런 문제를 해결하기 위해서 사용자와의 활발한 교류를 통해 직관적인 인터페이스를 개발하고 효과적인 가이드라인을 제시해 주는 것이 필요하다.

5. 결 론

효과적인 임상 의사결정 지원 시스템을 만들기 위해 실질적으로 사용자에게 도움이 될만한 체계적인 분석이 필요하다. 본 연구에서는 인지적인 접근을 통해 진단 직무, 진단 전략을 분석하고 그것을 통해 필요한 정보와 그것의 표현방식을 구성하여 인터페이

스에 반영하였다. 진단경력이 있는 전문가들과의 설문조사 및 인터뷰를 통해서 제안된 인터페이스의 필요성을 증명하였고 추후 발전방향에 대해서도 논의되었다.

현재 인터페이스는 임상 중 혈액검사를 바탕으로 구성되어 있다. 하지만 사람의 진단 과정이 유사하게 적용될 수 있고 실제로 임상 의사결정 시스템의 직무가 크게 다르지 않기 때문에 조직이나 임상 전체와 같이 복잡한 분야에 본 연구의 방법론이 적용 될 수 있을 것이다. 또한 의학 분야에서는 모든 자료의 신뢰성이 진단 실수와 이어질 수 있는 민감한 문제이기 때문에 인터페이스에도 제공된 정보의 신뢰도를 표시해줄 필요가 있다. 이 외에도 다양한 요구사항과 개선사항을 반영하면서 인터페이스를 설계한다면 효과적인 임상 의사결정 지원 시스템이 될 수 있을 것이다.

Acknowledgment

본 연구는 산업통상자원부 및 한국산업기술평가관리원의 산업핵심기술개발사업(지식서비스)의 일환으로 수행하였음. [10052955, 현장전문가의 경험지식 획득 및 활용을 위한 경험지식플랫폼 개발 연구]

참고문헌

[1] Belden, Jeffery L., Rebecca Grayson, and Janey Barnes (2009). “Defining and Testing EMR Usability: Principles and Proposed Methods of EMR Usability Evaluation and Rating.” *Technical Report. Healthcare Information and Management Systems Society* (HIMSS).

[2] Garg AX, Adhikari NJ, McDonald H, and et al (2005). “Effects of Computerized Clinical Decision Support Systems on Practitioner Performance and Patient Outcomes: A Systematic Review.” JAMA 293, no. 10 (March 9, 2005): 1223–38

[3] Mangiameli, Paul, David West, and Rohit Rampal(2004). “Model Selection for Medical Diagnosis Decision Support Systems.” Decision Support Systems 36, no. 3 (January 2004): 247–59.

[4] Paul, Mical, Steen Andreassen, Evelina Tacconelli, Anders D. Nielsen, Nadja Almanasreh, Uwe Frank, Roberto Cauda, Leonard Leibovici, and on behalf of the TREAT Study Group (2006). “Improving Empirical Antibiotic Treatment Using TREAT, a Computerized Decision Support System: Cluster Randomized Trial.” Journal of Antimicrobial Chemotherapy 58, no. 6 (December 1, 2006): 1238–45.

[5] Rasmussen, J (1986). “Information Processing and Human-Machine Interaction. An Approach to Cognitive Engineering”. North-Holland.

[6] Rasmussen, J., A. M. Pejtersen, and L. P (1994). Goodstein. Cognitive Systems Engineering. Wiley Series in Systems Engineering. Wiley.

[7] SHEPHERD, A. “HTA as a Framework for Task Analysis.” Ergonomics 41, no. 11 (November 1, 1998): 1537–52

[8] Sittig, Dean F., Adam Wright, Jerome A. Osheroff, Blackford Middleton, Jonathan M. Teich, Joan S. Ash, Emily Campbell, and David W. Bates (2008). “Grand Challenges in Clinical Decision Support.” Journal of Biomedical Informatics 41, no. 2 (April 2008): 387–92

[9] Tsopra, Rosy, Jean-Philippe Jais, Alain Venot, and Catherine Duclos (2014). “Comparison of Two Kinds of Interface, Based on Guided Navigation or Usability Principles, for Improving the Adoption of Computerized Decision Support Systems: Application to the Prescription of Antibiotics.” Journal of the American Medical Informatics Association 21, no. e1 (February 1, 2014): e107–16.

[10] van Wijk, Marc A.M., Johan van der Lei, Mees Mosseveld, Arthur M. Bohnen, and Jan H. van Bemmel (2001). “Assessment of Decision Support for Blood Test Ordering in Primary CareA Randomized Trial.” Annals of Internal Medicine 134, no. 4 (February 20, 2001): 274–81.

[11] Vicente, Kim J (1999). Cognitive Work Analysis: Toward Safe, Productive, and Healthy Computer-Based Work. CRC Press

[12] Wijk, Marc A. M. van, Johan van der Lei, Mees Mosseveld, Arthur M. Bohnen, and Jan H. van Bemmel (2002). “Compliance of General Practitioners with a Guideline-Based Decision Support System for Ordering Blood Tests.” Clinical Chemistry 48, no. 1 (January 1, 2002): 55–60.

혈액종합검사에 특화된 온톨로지 구축: 설계와 활용*

제현우, 박유경, 홍원의, 이문용

한국과학기술원 지식서비스공학과
대전광역시 유성구 대학로 291 305-701
jhw4824@kaist.ac.kr park60@kaist.ac.kr laftworld@kaist.ac.kr munyi@kaist.ac.kr

요약: 의료 종합 검진 분야의 전문 의료진들은 환자로부터 측정한 각종 검사 수치를 기반으로 소견문을 작성한다. 이때 전문가는 자신의 경험에 근거하여 여러 검사 수치 및 환자의 신상 기록, 자동 생성된 단편적인 소견 등을 복합적으로 검토하게 된다. 그러나 현재 검사 항목에 대한 기본적인 소견을 자동으로 생성해주는 Computer-aided Diagnosis 도구는 검사 결과 수치에 따른 단편적인 문구를 생성하는 수준에 그치고 있어 복잡한 사례에는 적용되기 어렵다. 또한 지식 재사용을 위한 지식 베이스(knowledge base)가 체계적으로 개발되어 있지 않아 전문의가 생성해온 상당량의 지식이 버려지고 있는 실정이다. 이 이외에도 전문의들의 문장 구사 습관에 따른 용어 정의가 표준화되지 않아 지식 공유의 문제도 부각되고 있다. 이러한 문제들을 해결하기 위하여 본 논문에서는 경험지식플랫폼(Experiential Knowledge Platform)을 소개하고, 해당 플랫폼에서 지식을 생성하는 메타데이터(meta data)로 활용되는 온톨로지(Ontology) 구축 과정을 제시한다. 특히, 방대한 의료분야에 대한 선행 연구로서 혈액종합검사 분야에 대하여 구축작업을 완료하였다. 본 연구에서 제시하는 혈액종합검사분야의 온톨로지는 현장에 적용되어 그 실효성을 검증받은 후, 타 분야로 확장되어 해당 분야의 전문 문서 검색 및 질의어 확장 모듈, 룰 생성 모듈을 생성하기 위한 지식베이스로 활용될 것을 기대한다.

핵심어: 온톨로지, 데이터베이스, 혈액종합검사, 경험지식플랫폼, 의학용어

1. 서론

Computer-aided Diagnosis (CAD)는 의료 검사 결과의 정확도를 높이는 동시에 검사 절차를 간소화하고 전문 의료인의 작업 부하를 줄이는 방향으로 진화하면서 의료 검사에서 주요한 분야로 자리잡았다. 의료 검사 및 진단 전문가들은 자동 생성된 소견을 참고하여 최종 진단 소견을 작성한다. 이와 같은 상황에서, 정보기술은 의사결정과정에 필요한 절차를 대리하여 전반적인 서비스 품질을 향상시킨다 [1]. 현재 의료분야 데이터베이스는 유전자부터 척수, 질환 등 세부분야로 나뉘어 연구되고 있으며, 이를 통합하는 연구 역시 진행되고 있다[2].

진단의학 과정에서는 검체 샘플을 검사하는 자동화 시스템을 활용해 빠른 시간에 판독을 내리지만, 최종적으로 전문의가 그 결과를 판단하고 의사결정을 하는 과정에서 의사결정지원시스템의 부재로 인해 의료진의 업무 부담이 가중되는 문제점이 있다. 본 연구팀을 포함한 국내외 유수 연구소 및 기업은 컨소시엄을 구성하여 현장전문가의 경험지식 획득 및 활용을 위한 경험지식플랫폼 개발 연구(이하, 경험지식플랫폼)를 추진하였다. 본 연구팀은 경험지식플랫폼에서 활용되는 온톨로지를 혈액종합검사 분야를 대상으로 개발하였다. 실제 혈액종합검사 분야에 사용되는 용어를 대상으로 검사항목, 검체, 질환과 같은 정보의 관계성을 정의하였으며, KOSTOM(Korea Standard Terminology of Medicine: 한국보건의료표준용어), UMLS(Unified Medical Language System), 그리고 KCD(Korean Standard Classification of Diseases)와 같은 기존 국내외 의료분야 데이터베이스 자료를 다수 수집해 상호 연계하였다. 산재된 용어들을 표준화된 형태로 표현하기 위해 동의어 중복 문제를 해소하였는데, 의미는 같지만 표현이 다른 개념들을 대표하는 대표용어를 유일하게 정의하고, 현장 전문의들의 온톨로지 검토과정을 모사한 구축

* 본 연구는 산업통상자원부 및 한국산업기술평가관리원의 산업핵심기술개발사업(지식서비스)의 일환으로 수행하였음.[10052955, 현장 전문가의 경험지식 획득 및 활용을 위한 경험지식플랫폼 개발 연구]

절차를 통해 결과의 신뢰성을 갖출 수 있었다.

현장전문가의 의사결정을 돕기 위한 경험지식플랫폼 개발연구에서는 혈액종합검사 영역의 논리구조 및 관계와 의학용어를 수록하는 온톨로지 구축이 필수적이다. 해당 온톨로지는 서로 다르게 해석될 수 있는 각종 의료 용어들을 표준화된 형태로 정리하는 역할뿐 아니라 혈액종합검사의 관계성을 내포하고 있기 때문에 그림 1 과 같이 경험지식플랫폼 내부의 질의어 확장 모듈과 룰 생성을 보조하는 메타지식으로 활용된다. 질의어 확장 모듈에서는 최종사용자가 사용자 인터페이스에서 만들어내는 질의어를 받아서 질의어와 관련된 내용을 의학 교과서 및 저널에서 자동으로 수집하는 것이 목적이다. 이러한 작업을 위해서는 적절한 질의어를 선정하는 것이 중요하다. 그렇기에 사용자가 부분적인 질의어를 입력하더라도, 해당 질의어가 어떤 의미를 갖고 있는지에 대한 정보와 관련 동의어를 수록한 온톨로지의 지원을 받아서, 더욱 정확한 관련 문서를 찾을 수 있다. 그리고 경험지식플랫폼 시스템 상에서 사례를 판단하는 것과 같이, 노하우 형태로 습득되는 절차적 지식이라는 일종의 룰을 생성하기 위해서 참조가 필요한 메타지식인 온톨로지가 필요하다.

본 연구팀은 혈액종합검사 온톨로지 구축 연구를 위해 국내외에서 연구된 사례가 있는 의료분야 데이터베이스를 수집하였다. 국외 사례로는 UMLS [1], MeSH [2] (Medical Subject Headings), SNOMED [3] (Systematized Nomenclature of Medicine)가 대표적이며, 국내에서는 KOSTOM, KCD 가 대표적인 의학 용어 데이터베이스이다. 하지만 이러한 국내외

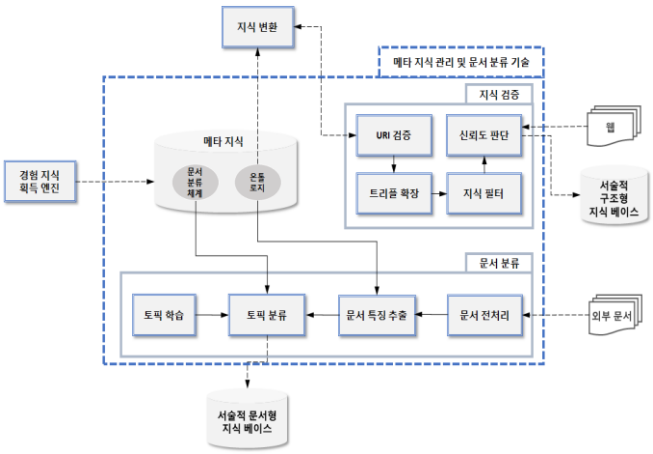


그림 1. 메타 지식 관리 및 문서 분류 기술 세부 동작도

[1] https://www.nlm.nih.gov/research/umls/

[2] https://www.nlm.nih.gov/mesh/

[3] https://www.nlm.nih.gov/research/umls/Snomed/snomed_main.html

데이터베이스 자료를 조합하여 경험지식플랫폼을 위한 메타지식으로 즉시 활용하기에는 몇 가지 문제점이 있다.

하나, 혈액종합검사라는 영역에 특화된 온톨로지가 부재한다. 다른 분야의 온톨로지와 상이하게 혈액종합검사 분야의 경우, 해당 분야의 특성으로 인해 검사항목, 검체, 질환, 검사항목 측정치 수준과 같은 개념 사이의 관계 정의가 필요하다.

둘, 표 1 의 예시와 같이 KOSTOM 온톨로지에는 불필요한 데이터 중복 문제가 존재한다. Abdominal pain 이라는 영문명을 복부통증, 배통증, 배앓이, 복통 등으로 다양한 한글명으로 표현할 수 있는데, 이러한 동의어 관계를 KOSTOM 에서는 한/영문 동의어를 한 쌍으로 묶어 ID(용어코드)를 부여하는 방식을 채택했기 때문에 불필요한 데이터 중복문제가 발생하게 된다.

표 1. 혈액종합검사 동의어 예시

용어코드	개념코드	영문명	한글명	UMLS
H00003487	H00003511	Abdominal pain	복부통증	C0000737
H00003492	H00003511	Abdominal pain	배통증	C0000737
H00003506	H00003511	Abdominal pain	배앓이	C0000737
H00003511	H00003511	Abdominal pain	복통	C0000737

셋, 혈액종합검사 온톨로지를 구축 시, 실제 의료현장에서 해당 구조를 활용 가능하도록 설계하는 것이 중요하다. 그렇기에 실제 의료기관에서 사용중인 시스템과의 연동을 통해 기존의 시스템이 소유하고 있는 경험지식을 포괄할 수 있는 온톨로지 구축의 필요성이 있었다. 이러한 이유로 실제 현장에서 활용되는 시스템에 내포되어있는 현장전문가의 경험지식을 획득할 수 있는 유연한 시스템을 구축하기 위해, 국내 유수 법인인 씨젠의료재단의 전산시스템과의 연동을 추진하였다.

위와 같은 이유로 현장에서 의료진이 사용하는 특수용어를 반영하여 확장할 수 있는 온톨로지 체계가 요구되었고, 본 연구팀은 혈액종합검사 온톨로지를 만들기 위한 기초 자료로 KOSTOM 데이터를 선정하였다. KOSTOM 은 2004 년부터 보건복지부 산하 보건의료정보표준화위원회가 장기적으로 구축해온 표준용어체계로 진단, 의료행위, 임상검사, 방사선의학, 치과, 보건, 간호, 기타의 8 개 분야 서브셋(subset) 테이블을 포함하는 통합용어테이블이다[3]. KOSTOM 을 혈액종합검사 온톨로지 구축의 기초자료로 선택한 것은 정제된 의학용어가 수록됨으로써 데이터의 신뢰성을 보장할 뿐 아니라, 외부 온톨로지 참조를 위한 각종 코드분류체계도 수록되어 있어 활용이 용이했기 때문이다. 또한 KOSTOM 에서 검사항목과 관련

된 용어를 보완하기 위해서 대한진단검사의학회[4]에 있는 의학용어와 각종 유의어를 수집해 반영하였다.

본 연구팀은 이러한 리소스 선정과정을 통해서 국내외의 온톨로지를 상호 연결할 수 있는 구조로, KOSTOM 데이터를 토대로 혈액종합검사를 위한 온톨로지 구축에 대한 연구를 진행하였다. 본 연구팀이 제안하는 혈액종합검사 온톨로지는 현장에서 구축된 데이터베이스를 활용함과 동시에 실제 종합검진 시스템과의 연동성을 고려하여 설계함으로써 실용성을 강화하였다. 실제 의료현장에서 발생하는 소견문에 등장하는 용어 역시 현장전문가의 경험지식으로써 반영하였다. 또한, 한/영 동의어 세트를 독립적으로 구축함으로써 문서검색에 활용되는 질의어 확장 모듈의 지식베이스로 활용할 수 있을 뿐 아니라 불필요한 질의어 생성 가능성을 사전에 방지함으로써 검색 시스템의 효율성을 증대시켰다. 추가로 의료기관의 협조를 통해 온톨로지 관계성 및 용어목록의 신뢰성을 갖출 수 있었다. 마지막으로 대표용어를 선정하는 구체적인 과정을 전문가의 업무 수행 절차를 모사하여 수행하였다.

2. 관련 연구

혈액종합검사 온톨로지를 구축하기 위해서는 먼저 의료분야에서 이뤄진 온톨로지, 데이터베이스, 시소러스(Thesaurus), 표준용어체계의 선행연구 검토와 그 차이점을 파악하는 작업이 선행될 필요가 있다.

온톨로지와 데이터베이스는 설계 목적에서 차이가 있다. 데이터베이스는 데이터들이 통합되고 저장되어있는 데이터의 집합인 것인 반면, 온톨로지는 사물을 개념과 개념간의 관계 정의로 표현해 놓은 것이다. 또한, 데이터베이스의 목표는 효율적인 데이터 관리 및 저장인 반면, 온톨로지는 실세계의 사물을 제약사항 및 개념간의 관계를 상세하고 체계적으로 표현하여 정의한 것이다.

시소러스는 특정한 도메인에서 사용되는 용어와 그들 사이의 관계를 제시한 색인어휘집이다. 시소러스는 주로 색인과 검색 과정에서 디스크립터와 검색어를 선정하기 위한 도구로 사용된다[4]. 그렇기에 정보검색 보조도구로서의 시소러스 개발을 위한 많은 연구와 이를 실현하기 위한 연구들이 진행되고 있다. 온톨로지와 시소러스의 공통점은 개념 간의 의미 관계를 기반으로 지식의 구조를 규정한 방법론이라는 것이다. 온톨로지와 시소러스는 모두 지식을 표현하는 방법이다. 지식 형성은 개념이 구조화된 것으로 생각할 수 있다. 즉, 각기 다른 개념이 정의되고 그 개념간 관계를 정의함으로써 지식은 규정된다. 그렇기에 지식을 형성하기 위한 방법론으로써는 온톨로지뿐만 아니라 시소러스 역시 가능하다.

온톨로지와 시소러스의 차이점으로는 온톨로지는 시소러스와 비교하여 개념 관계를 세분화할 수 있는 표현력을 가진다는 것이다[4]. 다시 말해, 각 개념의 정의와 용어간 관계를 다양한 정의 속성들을 활용함으로써 구조화 시킬 수 있다. 그 결과 시소러스와 비교하여 온톨로지는 인간이 지식을 이해하는 형상과 더욱 비슷한 구조를 나타낸다. 그렇기에 인간은 개념과 개념 사이의 관계성을 기반으로 지식을 파악하는 심적구조를 갖고 있기 때문에, 온톨로지를 활용한다면 개념 간의 관계를 구조화함으로써 부분적인 지식구조 영역의 특성을 더욱 분명하고 세밀하게 표현 가능하다.

온톨로지 언어가 시소러스와 구별되는 또 다른 특성은 일반화 혹은 상호운영성의 규칙을 적용함으로써 구조화된 지식으로부터 새로운 지식을 추론할 수 있다는 점이다[4]. 이는 온톨로지 설계자가 정의한 각기 다른 개념 및 속성 그리고 관계정의를 토대로 추론을 통해 추가로 나타나는 지식을 의미한다. 추가적으로 생성되는 지식 또는 룰은 메타지식을 통한 인공지능과 같은 역할을 시스템 속에서 담당한다.

국민 건강 증진과 의료서비스 분야의 가치 창출을 목적으로 세계 각국에서는 다양한 의료분야 온톨로지를 구축하고 있다. 그 중 대표적인 온톨로지는 UMLS, MeSH, SNOMED 이며, 국내에서는 KOSTOM, KCD 등이 있다. 의료분야 내 국소적인 온톨로지 구축에 대한 노력도 계속 진행되고 있는데, 그 중 대표적인 온톨로지는 Gene Ontology 로써 유전자 정보에 대한 개념과 의미 관계망를 OWL 언어로 표현한 것이다[5]. 그뿐만 아니라 인체의 전반적인 인체자원은행을 온톨로지로 설계하기도 하며, 세부영역인 척추정보만을 도메인으로 한 온톨로지를 연구하기도 하였다[6,7]. 또한 한국 및 중국 문화권에서는 한의학을 대상으로 한 온톨로지 구축에 대한 연구도 지속적으로 이뤄지고 있다[8]. 이러한 국소적인 분야를 다루는 온톨로지는 하나의 모듈로써 단일한 복합 온톨로지로 통합하려는 연구가 이뤄지고 있다[2].

국외에서의 대표적인 의료분야 온톨로지는 UMLS, MeSH, SNOMED 등이 있다. UMLS 는 미국 국립의학도서관(National Library of Medicine, 이하 NLM)에서 1986 년부터 시작한 장기 연구개발 프로젝트로써 다양한 생명과학정보를 통합하고 검색하기 위해 개발된 온톨로지다. UMLS 를 이용함으로써 기존의 동일 개념을 표현하는 데 사용되는 용어의 다양성으로 인한 정보검색의 효율저하 및 통합검색 문제를 해소할 수 있었다[9]. UMLS 는 개념 및 용어 데이터베이스인 메타시소러스, 개념들을 분류하기 위한 의미망, 자연어처리 작업을 위한 전문가사전으로 구성된다[9]. 메타시소러스는 생물의학적 개념과 동의어뿐만 아니

[4] http://www.labtestsonline.kr/

라 개념간 관계성에 대한 정보를 포함한다. 의미망은 모든 개념들을 일괄적으로 목록화한 시멘틱 관계(semantic relationship)으로 구분된다. 전문가사전은 자연어처리를 위한 사전정보를 제공하기 위한 도구이다.

MeSH는 미국 국립의학도서관에서 구축한 의학분야의 시소러스로써 의학용어를 유의어, 상위어, 하위어, 관련어 등으로 정의했다. MeSH 는 1960년 “Index Medicus(New Series)” 발간과 함께 4,400 개의 용어를 수록한 “Medical Subject Headings: Main headings, Subheadings, and Cross references used in the Index Medicus and the National Library of Medicine Catalog”란 서명으로 초판이 발행되었다[10]. 컴퓨터의 발전 및 보편화로 인해, MeSH는 의료현장에서 데이터베이스 색인과 검색에 사용되고 있다. MeSH 표목이 27,000여개로 늘어났으며, 어휘의 상호참조, 기입어를 포함하면 123,000 여개에 이른다[10]. 현재 NLM 에서는 MeSH 를 의료분야의 여러 하위 데이터베이스와 NLM 에서 소장하는 목록의 주제색인을 위해 사용하고 있다.

SNOMED 는 전자의무기록을 위한 용어시스템으로써 CAP(College of American Pathologists)에 의해서 발전, 유지, 배포되고 있다[11]. 또한 질병, 임상소견, 처치를 표현하는 포괄적 임상용어로 질병을 분류하기 위해 개발한 것이다. SNOMED 는 다양한 전문분야와 진료 장소에서 임상자료를 색인, 저장, 검색 및 통합하는 것을 허용한다. 또한 SNOMED의 가장 큰 특징은 특정 질병을 질병이름뿐만 아니라 환자의 상태를 묘사하는 방식으로도 코딩이 가능하다는 점이다[11].

국내에서 대표적인 의료분야 용어 및 분류체계 연구는 KOSTOM 과 KCD 가 있다. KOSTOM 은 2004 년부터 보건복지부 산하 보건의료정보표준화위원회가 한국보건의료표준용어체계를 구축한 것이다[3]. KOSTOM 구축은 용어표준화를 위한, 보건의료 및 한의학에 사용되는 모든 용어를 개념단위로 군집화 하고 각 군집마다 유일한 코드를 부여함으로써 개념과 동의어 처리를 마칠 수 있었다. 이러한 활동은 향후 보건의료기관 HER(Electronic Health Record: HER)시스템의 상호운용성을 보장하기 위한 기초이다[3]. 표준화위원회에서는 보건의료정보 표준화연구를 통하여 KOSTOM에 포함되어있는 KCD 한국어용어를 UMLS에 등재를 함으로써 국외 자료와 연결을 도모했다. 각 분야별 보건의료용어에서는 해당 분야에 맞는 외부의 기본용어체계를 참조함으로써 설계의 타당성을 높였다. 또한 WHO 용어작업에 합류하는 등 국제기구와의 협력을 도모하여 UMLS 와 LOINC(Logical Observation Identifiers Names and Codes)에 등재하였다.

국가간 통일된 의학용어체계의 필요성으로 ICD(international classification of diseases)를 사용하고 있다. 우리나라도 ICD-10 을 번역하여 국내실정에 맞도록 질병을 추가하고 상세 분류한 KCD-6(Korean classification of diseases)을 사용하고 있다[12]. 이러한 국제표준 분류체계를 사용함으로써 생기는 장점은 다음과 같다. 하나, 의료기관뿐만 아니라 지역, 국가간 상호 운용성을 확보할 수 있고, 국가간 통계 산출 및 다양한 분야간의 데이터 비교가 가능하다[12]. 이처럼 국가간 통일된 의학용어체계 구축은 의무기록의 전산화 통일뿐만 아니라 국가간 의료서비스의 양과 질적 측면 비교에도 활용할 수 있다.

이처럼 UMLS, MeSH, SNOMED, KOSTOM, 그리고 KCD와 같이, 국내외에서 의학분야 온톨로지 구축에 대한 연구가 지속되고 있다. 하지만 이러한 선행연구결과를 혈액종합검사 온톨로지를 구축하는데 기초리소스로 바로 적용하기에는 여러 문제점이 존재했다. 먼저, UMLS 는 의학분야에 대한 용어를 정리한 데이터베이스인 메타시소러스뿐만 아니라 개념간의 관계를 정의 및 분류하여 구조적으로 정의를 내린 의미망을 갖고 있지만, 혈액종합검사를 위한 개념간의 의미관계 설정이 되어있지 않았다. UMLS 는 단순 개념들간의 상하 구조적 관계를 포함하고 비슷한 개념을 갖고 있는 용어들을 묶는 작업을 중점으로 하였다. 또한, MeSH 역시 의학용어를 유의어, 상위어, 하위어, 관련어로 정의하는 작업을 하였을 뿐, 혈액종합검사를 위한 개념간의 관계설정에 대한 연구는 없었다. 이와 마찬가지로 SNOMED 와 KOSTOM, 그리고 KCD 는 의학용어를 표준화하여 의학 용어를 개념화시켜 정리한 작업이다. 그렇기에 이러한 선행연구들은 의학용어 표준화 작업 및 개념화 작업에서 큰 기여를 하였지만, 혈액종합검사 온톨로지 구축을 위한 개념간 관계 설정이 되어있지 않은 한계가 있었다. 본 연구팀은 이러한 선행연구의 리소스를 활용해서 혈액종합검사라는 특수한 영역의 온톨로지 구축을 시도했다.

본 연구팀은 혈액종합검사 온톨로지를 구축하기 위해서 단순히 의학용어를 개념화시키는 작업뿐만 아니라, 혈액종합검사를 위해 필요한 특수 개념들의 유기적인 관계정의 작업을 시도했다. 그렇기에 본 연구팀은 혈액종합검사를 위해서 검사항목과 검체, 그리고 질환과의 유기적인 의미 관계망을 정의했다. 또한, 특정 검사항목의 수치 수준에 따라 상이한 질환을 의심할 수 있기 때문에 검사수치 수준에 대한 정보의 연결을 시도했다. 즉, 혈액종합검사 온톨로지 구축을 위해서는 검사항목, 검체, 질환, 검사항목 측정치 수준과 같은 4 가지 상위 개념간의 의미관계 설정이 필요했다. 이러한 이유로 인해, 본 연구팀은 앞서 언급한 문제점을 해결할 수 있는 혈액종합검사에 특화된 온톨로지 구축에 대한 연구를 진행하였다.

3. 혈액종합검사 온톨로지 설계

본 연구팀이 제안하는 혈액종합검사 온톨로지는 관계형데이터베이스 기반으로 설계하였다. 의학용어 표준화 작업을 위해 만들어진 KOSTOM은 데이터가 테이블 형식으로 구조화되어 있었다. 그렇기에 본 연구팀은 관계형데이터베이스를 활용하여 검사항목, 검체, 질환, 측정치 수준과 같은 개념간의 관계를 정의함으로써 KOSTOM 데이터를 확장 및 재구조화 시키는 것이 혈액종합검사 온톨로지를 표현하기에 가장 적합하다고 판단했다. 왜냐하면 기존 KOSTOM 에 수록된 의학용어와 구조를 반영할 수 있을 뿐만 아니라, KOSTOM에서 발생하는 불필요한 데이터 중복을 최소화하는 스키마를 설계함으로써 정규화 문제를 해소할 수 있다는 장점이 있기 때문이다.

본 연구팀은 혈액종합검사 온톨로지 구축에 필요한 의학용어를 수록하기 위해 각종 리소스 목록을 조사하여 표 2 와 같은 최종 리소스를 선정하였다. 이를 토대로 혈액종합검사 영역에서 발생하는 개념간의 관계를 정의하는 혈액종합검사 스키마를 설계했다. 추가로 데이터 정제 및 대표용어 선정 프로세스를 고안하였으며, KOSTOM 데이터가 반영하지 못하는 검사항목 용어와 같은 추가적인 정보를 외부 리소스를 참조하여 온톨로지에 수록하였다.

표 2. 혈액종합검사 온톨로지 리소스

자료명	형태	참조 데이터	목적
2015 씨젠의료재단 검사안내	책	검사항목(Component)	용어, 의미관계 추출
씨젠 Handbook of Laboratory tests	책	관련질환(Disease)	용어, 의미관계 추출
씨젠 혈액종합검사 종목 (전산)	엑셀	검사항목(Component) 검체(System)	용어 추출
씨젠 검사동의어 (전산)	엑셀	검사항목(Component)	동의어 추출
KOSTOM	엑셀	검사항목(Component) 검체(System) 관련질환(Disease)	동의어 추출 KOSTOM 코드 UMLS코드 기타코드(KCD)
Lab Tests Online	웹	검사항목(Component)	동의어 추출
씨젠 전산 시스템 소견문	엑셀	소견문 내 특이용어	용어 추출

본 연구팀이 혈액종합검사 온톨로지 구축을 위해 선정한 리소스는 표 2와 같다. 경험지식플랫폼의 모듈로 활용될 혈액종합검사 온톨로지를 구축하기에 기초가 되는 리소스로 가장 적합한 것은 KOSTOM이라고 생각하였다. 왜냐하면 의학용어 표준화를 위해 국내외 보건의료에 사용되는 모든 의학용어를 개념단위로 정리하였고, 국제용어체계와의 연결을 위한 작업이 선행된 리소스이기 때문이다.

하지만 KOSTOM에 존재하는 데이터를 혈액종합검사 온톨로지에 포함되는 데이터로 바로 활용할 수는 없었다. 왜냐하면 하나, KOSTOM에는 혈액종합검사뿐만 아니라 의료 전 분야에 대한 용어표준화 작업이 진행된 데이터가 포함되어있기 때문이다. 그렇기에 본 연구팀은 혈액종합검사에 관련된 의학용어를 추출할 필요가 있었다. 둘, KOSTOM 구조를 혈액종합검사 온톨로지에 적용시키는 것은 데이터의 중복문제를 심화시키는 단점이 있었다. KOSTOM 데이터는 표 1 에서 언급한 것과 같은 구조로써 발생하는 동의어 중복문제가 있었다. 이러한 구조를 그대로 활용하여 검사항목, 검체, 질환, 측정치 수준을 나타내는 속성 정보를 포함하도록 온톨로지를 설계한다면 데이터 중복문제가 심각해진다는 단점이 있었다. 그렇기에 대표용어를 선정함으로써 KOSTOM 에 있는 동의어 중복문제를 해소하는, 데이터베이스 정규화 작업이 필요했다. 셋, KOSTOM 에는 혈액종합검사를 위한 특수한 영역에서 발생하는 검사항목에 대한 용어를 모두 다루고 있지 않았다. 그렇기에 본 연구팀은 대한진단검사의학회의 자료를 참조하여 KOSTOM에서 수록되어 있지 않은 혈액종합검사 검사항목에 대한 용어 및 동의어를 온톨로지에 반영했다. 대한진단검사의학회에서는 보건 의료에서 중요한 역할을 하고 있는 진단검사의학 분야의 각종 검사에 대해 이해하기 쉽게 기초적 정보를 환자 및 그 보호자들에게 제공하고 있을 뿐만 아니라 직접 모든 자료를 제작 및 감수를 시행함으로써 데이터의 신뢰성을 높였다. 또한, 미국 AACC(American Association for Clinical Chemistry)와의 협력에 의해 구축된 범세계적 사이트이기에 온톨로지 데이터 수록에 대한 신뢰성을 갖춘 리소스라고 판단했다. 넷, KOSTOM 은 혈액종합검사 온톨로지 구축을 위한 개념간의 관계정의 정보가 부재했다. 그렇기에 본 연구팀은 표 2 의 리소스를 참조하여 혈액종합검사를 위한 검사항목, 검체, 질환, 그리고 검사항목의 측정치 수준과 같은 개념들 간의 관계를 정의했다. 그 후에, 혈액종합검사에 필요한 개념들 간의 관계를 정의한 구조를 나타내는 스키마 설계를 시도했다.

현장전문가의 경험지식을 획득하는 것에 중점을 두고 있는 경험지식플랫폼 연구 특성을 고려하여 실제 사용중인 전산시스템 소견문에서 의료진이 사용한 용어를 추가로 온톨로지에 수록하였다. 이러한

작업을 통해서, 온톨로지는 국내외 표준에 맞는 의학용어체계뿐만 아니라 의료현장에서 실제로 사용되고 있는 용어를 상호 연결함으로써, 시스템 자체의 유연성을 높였다.

현장에서 요구하는 혈액종합검사에 대한 세부적 검사항목 종류와 그에 따른 검사수치는 병,의원마다 차이가 있을 수 있다. 이러한 이유로, 본 연구팀은 씨젠의료재단에서 사용하고 있는 검사항목 및 검사수치에 따른 질환의 의심정도를 판단하는 기준에 맞추어 온톨로지를 구축하였고, 추후 사용자의 요구에 따라 변경(customization) 및 확장 가능하도록 하였다.

본 연구팀이 제안하는 혈액종합검사 온톨로지의 스키마는 그림 2 와 같다. 혈액종합검사 온톨로지에서 중심이 되는 테이블은 KEYWORD 테이블이다. 해당 테이블은 대표용어를 수록하기 위해 고안되었다. 해당 테이블을 구성하는 속성은 3 가지이다. EKP_CODE 는 대표용어와 매핑되는 주 키(primary key)를 뜻한다. 해당 코드을 활용함으로써 대표용어와 관련된 동의어 집합을 탐색할 수 있다. 또한 TYPE 속성으로 해당 대표용어가 어떤 카테고리에 묶이는지에 대한 정보를 포함한다. 현재 혈액종합검사에서 대표용어는 검사항목, 검체, 질환, 카테고리, 기타와 같은 5 가지 타입(type)으로 구분된다. 대표용어를 지정하는 테이블과 동의어 테이블을 구분함으로써, 기존 KOSTOM 데이터베이스에서 발생하는 데이터 중복을 최소화하는 스키마 설계를 고안하여 정규화 문제를 해결할 수 있었다.

동의어 사전 역할을 할 수 있는 테이블은 KR_SYN, EN_SYN 과 같이 두 테이블로 분할하였다. 왜냐하면, 온톨로지를 활용해서 문서를 검색할 경우, 한/영 동의어를 구분 없이 모아두면 검색에 불필요한 질의어를 만들어 낼 가능성이 존재하기 때문이다. 이러한 이유로 한/영 동의어 테이블을 따로 설계함으로써 질의어를 확장하는 작업에서의 검색 정확도 및 검색 효율을 온톨로지 구조 자체에서 강화시켰다.

KEYWORD_CODE_MATCHING 테이블에는 외부 온톨로지와의 연계를 위해 각종 코드정보를 수록하였다. 그렇기에 특정 대표용어와 관련된 UMLS, KOSTOM, KCD 코드를 참조하여 외부 온톨로지와 연동 및 탐색이 가능할 수 있도록 하였다. 추가로 UMLS, 및 KOSTOM 정보가 갱신될 때, 해당 코드를 참조하여 혈액종합

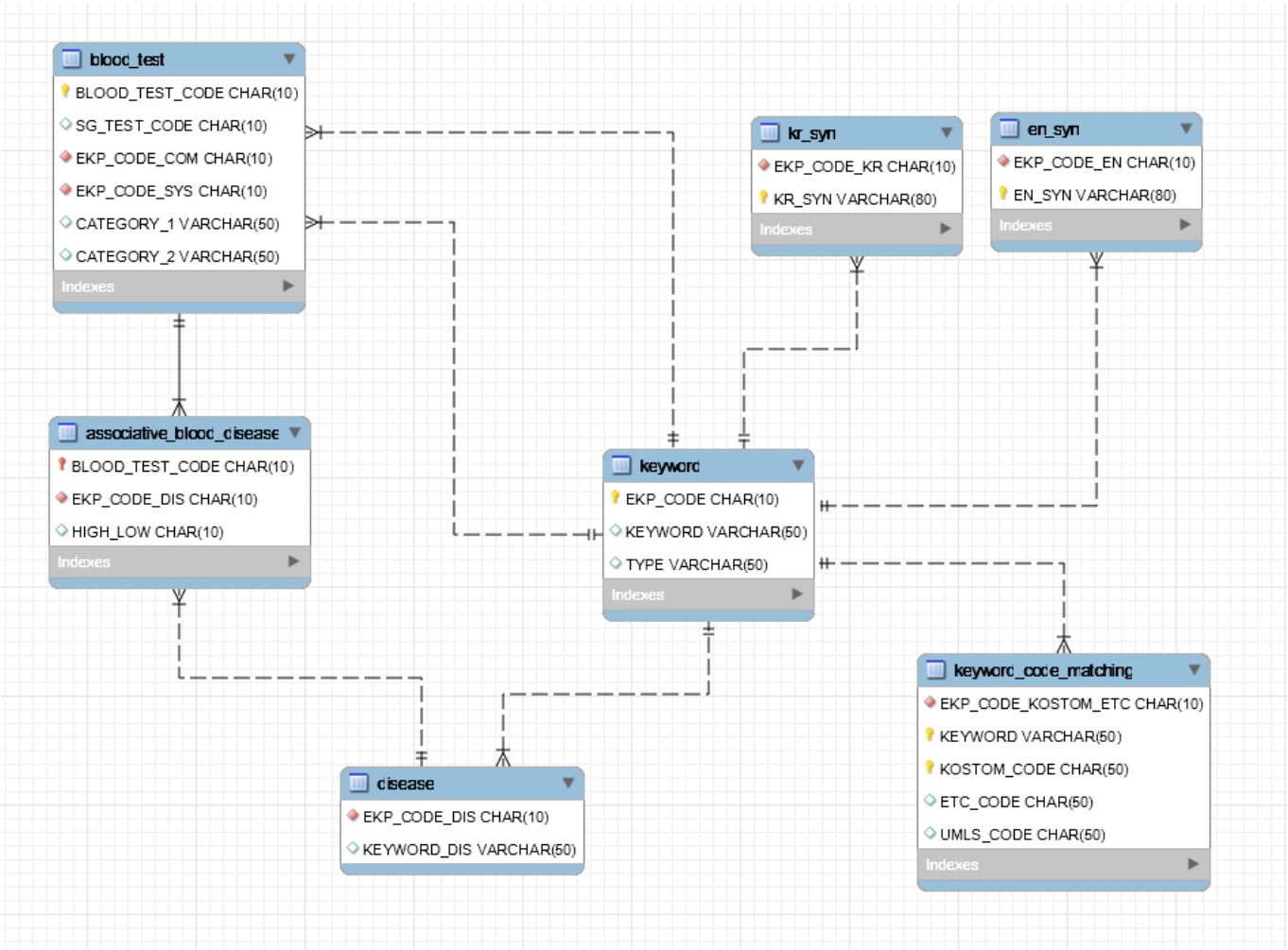


그림 2. 혈액종합검사 온톨로지 스키마

검사 온톨로지의 수정사항 반영을 용이하도록 설계하였다.

혈액종합검사의 전체적인 논리구조 및 의미관계를 포함하는 테이블은 BLOODTEST 이다. 혈액종합검사 구체적 하위 항목은 병,의원마다 다소간 차이가 있다. 그렇기에 본 혈액종합검사 온톨로지는 씨젠의료재단의 혈액종합검사 기준을 따랐다. 이런 이유로, 혈액종합검사 항목과 관련된 질환 정보와 측정치 수준 정보를 담고 있는 ASSOCIATIVE_BLOOD_DISEASE 테이블과 BLOODTEST 테이블을 분리하여 병,의원마다 발생하는 검사항목의 차이를 BLOODTEST 테이블에 유연하게 반영시킬 수 있도록 스키마를 설계하였다. 또한, 기존에 운영되고 있는 전산시스템의 의료정보 체계를 반영하기 위해서 SG_TEST_CODE 를 부여하여 기존 시스템과의 연동을 도모했다. 검사항목과 검체의 코드정보를 담고 있는 EKP_CODE_COM 와 EKP_CODE_SYS 의 관계성을 BLOOD_TEST_CODE 라는 주 키를 활용하여 수록했다. 또한, 해당 항목이 실제 진단을 목적으로 하는 병,의원 현장에서 쓰이는 분류 용어와 의학 교과서에서 쓰이는 분류 용어와의 불일치로 속성값 CATEGORY_1, CATEGORY_2 를 구분하여 해당 정보를 따로 수록하였다. 실제 현장에서 사용되고 있는 분류체계와 학술정보를 탐색할 수 있는 정보를 모두 포함함으로써, 해당 온톨로지를 활용하여 문서검색의 효율성을 높일 수 있었다.

ASSOCIATIVE_BLOOD_DSEASE 테이블에는 검사항목과 검체의 관계 정보를 담고 있는 BLOOD_TEST_CODE 와 질환을 나타내는 코드 정보인 EKP_CODE_DIS 그리고 검사항목 측정치의 수준 정보를 나타내는 HIGH_LOW 속성값이 있다. EKP_CODE_DIS 와 같은 코드정보를 활용한 이유는 동의어 중복문제를 해결하기 위함이다. 다양하게 표현될 수 있는 동의어들을 모두 수록하여 불필요한 데이터 중복을 막기 위해서, 각 동의어 세트를 대표하는 대표용어의 코드를 활용하였다. 또한, 구체적인 측정치는 병,의원마다 차이가 있는 것을 고려하여 메타지식인 온톨로지에서는 단순히 특정 검사 수치 수준이 높은지 또는 낮은지에 대한 정보만 HIGH_LOW 속성을 통해서 수록하였다.

끝으로 DISEASE 테이블은 질환 사이의 계층정보의 반영을 고려해서, KEYWORD 테이블과 독립성을 유지하였다.

이로써 혈액종합검사 온톨로지는 경험지식플랫폼 내부에서 활용되는 메타지식의 역할을 할 뿐만 아니라, 혈액종합검사라는 의료서비스 세부분야 온톨로지를 설계 및 구조화하는 방법을 제시하였다.

본 연구팀이 활용한 의료용어 데이터는 KOSTOM 을 사용하였다. 그 이유는 KOSTOM 은 보건의료정보표준화위원회가 한국보건의료표준용어체계를 구축한 것으로써, 의학용어를 개념을 토대로 구성하고 UMLS, KCD 와 같은 국내외 대표적인 온톨로지와의 호환성을 보장하는 코드 정보를 수록하고 있기 때문이다.

KOSTOM 데이터를 기반으로 앞서 본 연구팀이 설계한 스키마에 맞도록 1 차적으로 용어를 분류하고, 삭제 및 수정 또는 추가적인 검토가 필요한 용어들은 2 차 전문의 검토를 위하여 남겨두었다. 2 차 전문의 검토를 마친 데이터는 본 연구팀이 재확인 후, 해당 결과를 온톨로지에 반영하였다. 지식공학자들의 불충분한 의학 도메인 지식은 부정확한 데이터베이스 설계 및 구축을 야기할 수 있기 때문에 정확한 용어 분류 및 정제가 필요했다. 그렇기에, 본 연구팀은 실제 현장에서 근무하는 전문의들의 협조를 받아 실제 스키마 구조와 데이터 정확성 및 분류체계의 검토를 받은 후, 해당 내용을 본 온톨로지에 반영함으로써 스키마와 데이터의 신뢰성을 갖출 수 있었다. 이러한 접근법을 활용한 이유는 의료용어를 분류 및 정제하는 과정에서 오류가 발생한다면, 혈액종합검사 온톨로지의 오염을 일으킬 뿐만 아니라 경험지식 플랫폼에서 온톨로지를 활용하는 모듈의 성능에도 악영향을 끼칠 수 있기에 2 차 검토를 기반으로 온톨로지 데이터를 수록하였다.

본 연구팀은 각 동의어 묶음들을 대표하는 대표용어를 정의하기 위해 KOSTOM 개별코드 및 대표코드를 활용했다. KOSTOM 개별코드는 각 항목별로 부여된 주 키의 역할을 하고 있다. KOSTOM 대표코드는 비슷한 용어를 하나의 개념으로 묶는 역할을 한다. 본 연구팀은 해당 코드를 그대로 활용할 수 없었다. 왜냐하면 혈액종합검사란 특수 분야에 관련된 의학용어가 KOSTOM 에 모두 수록되어 있지 않았기에, 추가로 외부 리소스에서 용어를 수록할 필요가 있었기 때문이다. 또한, 실제 현장에서 사용하는 전산시스템에서 발생하는 의학용어를 수록하기 위해서 추가적인 코드 체계가 필요했다.

이러한 이유로 본 연구팀이 제안한 혈액종합검사 온톨로지에서는 EKP_CODE 를 활용함으로써 추가적인 데이터 확장을 가능하게 했으며, KOSTOM 의 코드 정보를 독립적으로 수록함으로써 기존 KOSTOM 의 원래 구조와도 연동이 가능하도록 하였다. 본 연구팀은 EKP_CODE 를 활용하여 대표용어와 매핑을 시켰다. 대표용어가 명확히 지정되지 않은 경우에는 데이터 정제 프로세스에서 전문의의 협조를 통해 동의어 목록을 검토한 후, 각 용어를 가장 잘 표현하고, 사용빈도가 높은 단어를 대표용어로 선정하여 EKP_CODE 를 부여하였다.

본 연구팀이 혈액종합검사 온톨로지를 설계하며 고려한 부분 중 하나는 기존에 사용중인 전산시스템과의 연계성 보장과 외부 리소스를 자유롭게 추가할 수 있는 온톨로지의 확장가능성이었다. 경험지식플랫폼 개발연구에서 중점을 두고 있는 사항 중 하나는 현장전문가의 경험지식을 획득하여 활용하는 것이었다. 그렇기에 기존에 현장에서 사용하고 있는 전산시스템에 저장되어 있는 정보를 온톨로지에 수록하려고 시도했다.

현장전문가인 의료진이 혈액종합검사 결과치를 보고 실제로 어떠한 의료용어 및 표현을 소견서에 작성하는지에 대한 정보를 온톨로지에 반영함으로써, 현장 및 기존 전산시스템이 갖고 있는 경험지식을 획득할 수 있다고 판단했다. 이를 위해 현장에서 사용된 소견문을 전처리하여 일반명사를 추출했다. 소견문에서 등장하는 용어 중 온톨로지에 이미 수록된 용어들을 제거한 후, 본 연구팀과 현장 전문의들의 이중 검토를 거쳐 소견문에서 발생하는 현장용어를 추가로 온톨로지에 반영하였다.

KOSTOM 은 의학용어를 표준화하면서 수많은 종류의 질환을 개념화하고 국내외 의료 온톨로지에 대한 코드를 수록한 장점이 있었지만, 혈액종합검사를 위한 검사항목에 대한 용어 정리에 대해서는 부족함이 있었다. 그렇기에 본 연구팀은 이러한 한계를 보완하기 위해서 대한진단검사의학회에 나타나는 검사항목 정보를 웹크롤링을 통하여 검사항목 용어 및 동의어를 추출하였다. 그리고 한글명, 영문명과 같이 독립적으로 분류한 후, 해당 데이터를 온톨로지에 반영하였다.

본 연구팀은 혈액종합검사 온톨로지를 만들기 위한 각종 리소스 목록을 탐색하여, 혈액종합검사와 관련된 의학용어를 수록하기에 가장 적합한 기초 리소스로 KOSTOM 을 선정하고, 혈액종합검사 스키마를 설계했다. 또한, 데이터 정제 및 대표용어 선정 프로세스를 고안하였으며 KOSTOM 데이터가 반영하지 못하는 정보를 외부 리소스를 참조하여 추가함으로써 온톨로지의 신뢰도 및 활용성을 높였다.

4. 결과

본 연구팀이 혈액종합검사를 위한 온톨로지 구축에 사용한 데이터는 표 3 과 같다. KEYWORD 테이블은 검사항목 105 개, 질환 221 개, 검체 3 개, 그리고 검사구분 22 개에 대한 대표용어를 수록한다. 또한, EN_SYN, KR_SYN 테이블에서는 대표용어들을 다르게 부를 수 있는 영문 동의어 1195 개 한글 동의어 1044 개가 수록되어 있다. 또한, 혈액종합검사 온톨로지를 구성하는 개념간 관계를 정의한 테이블은 BLOODTEST 와 ASSOCIATIVE_BLOOD_DISEASE 으로써 각각 105 개 377 개의 관계정보를 수록하고 있다. 그리고 각 의료용어에 대응되는 외부 온톨로지의 코드 정보는 약 694 쌍이 존재한다.

본 연구팀이 구축한 혈액종합검사 온톨로지는 다양한 측면에서 활용 및 응용이 가능하다. 먼저 본 온톨로지는 경험지식플랫폼의 하위 모듈인 문서 분류 및 처리작업에서 활용될 수 있다. 적절한 문서를 찾기 위해서는 적절한 질의어가 필요하다. 그렇기에 올바르게 구조화되고 정제된 용어사전이 필요하다. 더 나아가, 질의어를 만들기 위한 단순 동의어 사전

표 3. 혈액종합검사 온톨로지 데이터 확보 현황

자료명	개수	참조 테이블
Component (검사항목: 대표용어)	105 개	KEYWORD
Disease (질환: 대표용어)	221 개	KEYWORD
System (검체: 대표용어)	3 개	KEYWORD
Category (검사구분: 대표용어)	22 개	KEYWORD
English synonym(영문 동의어)	1195 개	EN_SYN
Korean synonym(한글 동의어)	1044 개	KR_SYN
검사항목-검체 관계	105 개	BLOODTEST
검사항목-검체-질환-측정치 관계	377 개	ASSOCIATIVE_BLOOD_DISEASE
외부 코드(KOSTOM, UMLS, KCD)	694 쌍	KEYWORD_CODE_MATCHING

역할뿐만 아니라 해당 용어와 관련되어 나타나는 다양한 의미관계 정보(검체, 검사항목, 질환, 수준)가 포함되어 있어, 더욱 정확한 검색 결과를 기대할 수 있도록 한다.

또한, 혈액종합검사 온톨로지는 의학용어 분야를 도메인으로 다룸으로써 발생할 수 있는 잘못된 데이터 분류 및 정제 작업의 문제점에 주목하였다. 이러한 문제점을 예방하기 위해 구체적인 용어 정제 절차를 고안하고, 이를 반영한 결과를 온톨로지에 수록하였다. 또한, 전문의 협조를 통해 온톨로지에 수록된 용어목록과 관계성을 전수 검토함으로써 데이터 신뢰성을 갖출 수 있었다.

혈액종합검사 온톨로지를 구성하는데 토대가 된 KOSTOM 의 의학용어를 수록하는데서 그치지 않고, UMLS, KCD 와 같은 외부 온톨로지와의 확장적 연계를 고려하여 관련 코드를 수록함으로써 의학용어 국제표준화와도 연계를 도모했다.

끝으로 본 연구는 경험지식플랫폼 개발연구의 하위 모듈로 활용될 혈액종합검사 온톨로지 구축 연구로써, 경험지식플랫폼 내부에서 사례판단을 위한 룰을 생성하는 메타지식으로 활용 가능하다. 그리고 실제 의료기관의 전산서비스 체계와의 연동을 도모함으로써 유기적인 온톨로지 시스템 확장을 가능하게 한다.

5. 결론

본 연구팀은 경험지식플랫폼 개발연구에서 활용될 하위모듈인 혈액종합검사 온톨로지를 구축했다. 기존에 UMLS, MeSH, SNOMED, KOSTOM, 그리고 KCD 와 같은 의학 분야의 온톨로지가 존재했지만, 이러한 연구결과를 바로 본 연구에 적용할 수 없는 문제가 있었다. 왜냐하면 혈액종합검사를 위한 검사항목, 검체, 질환, 그리고 측정치 수준과 같은 개념 사이의 관계 정의가 없었기 때문이다. 이러한 문제로 인해 본 연구팀은 KOSTOM 리소스를 기초로 활용해서, 부족한 의학용어를 외부 리소스를 활용하여 보강한 후, 혈액종합검사에 필요한 개념간의 관계를 관계형데이터베이스로 정의함으로써, 혈액종합검사 온톨로지 구축을 시도했다.

본 연구팀이 제안한 혈액종합검사 온톨로지는 다음과 같은 장점을 지닌다. 하나, 본 연구는 경험지식플랫폼의 하위 모듈 개발에 대한 연구일 뿐만 아니라, 혈액종합검사를 위한 개념간 관계성 정보를 데이터베이스를 활용하여 구축함으로써 다양한 현장에서 사용될 수 있는 혈액종합검사 온톨로지를 구축하였다. 둘, 대표용어 및 동의어 목록 구축으로 질의어 확장과 같은 문서 검색을 위한 지식베이스로 활용 가능하다. 셋, 체계적인 용어 검토 프로세스를 고안하고 반영함으로써 혈액종합검사 온톨로지에 대한 신뢰성을 갖추었다. 넷, KOSTOM, UMLS, KCD 와 같은 다양한 외부 온톨로지에 대한 코드정보를 수록함으로써 외부정보 참조 및 확장을 용이하도록 만들었다.

본 연구에서의 한계점으로는 의료현장에서 기존에 사용하고 있던 전산시스템 내부의 소견문 데이터에 등장하는 현장용어를 사전에 한번만 추출하여 온톨로지 반영한 부분이다. 이로써 추가로 발생하는 용어를 지속적으로 온톨로지에 반영하지 못한다는 한계가 있다. 무조건적으로 소견문에 나오는 용어를 모두 온톨로지에 수록하는 것은 온톨로지의 오염을 초래하는 문제를 야기할 수도 있으나, 소견문에 등장하는 용어에 대한 지속적 모니터링 기능을 개발한다면, 소견문에서 발생하는 유사단어를 판별하여 자동적으로 속성에 맞게 분류하고 온톨로지를 확장하는 세부 모듈 개발을 연구할 수 있을 것이다.

본 연구팀은 다음과 같은 후속 추가 연구를 제안한다. 하나, 지식경험플랫폼의 범위가 혈액종합검사뿐만 아니라 장기적으로는 조직검사로 확장될 필요가 있다. 그렇기에 해당 범위를 커버하기 위해 온톨로지 역시 확장될 필요가 있다. 조직검사 온톨로지가 구축된다면, 다양한 질환과 검사항목들 간의 관계성에 대해 구조적인 지식체계를 갖춤으로써 조직검사를 대상으로 하는 의사결정지원시스템에 적용할 수 있을 것이다. 둘, 현재 혈액종합검사 온톨로지에는 검사항목, 검체, 질환, 검사항목 측정치 수준에 대한 개념 사이의 관계성을 포함하고 있다. 추후에는 더욱 정교한 관계설정을 위해 사람의 나이, 성별 등 혈액종합검사에 영향을 미칠 수 있는 인적 요인의 관계정보를 추가할 필요가 있다. 끝으로 의료진이 입력하는 소견문에서 새로운 의학용어가 발견되면, 자동적으로 동의어를 추출하여 온톨로지를 확장하는 기능에 대한 개발연구 필요성이 존재한다. 이러한 후속 연구가 진행된다면 의료진단검사 영역에서 유용하게 활용될 수 있는 메타지식을 구축하는데 기여를 할 수 있을 것이다.

본 연구팀이 수행한 혈액종합검사 온톨로지 구축을 위한 연구는 경험지식플랫폼의 모듈로 활용되는 메타지식을 구축하기 위한 연구일 뿐만 아니라, 혈액종합검사 온톨로지 구축이란 세분화된 의료영역의 온톨로지 구축에 대한 방법을 제시한다. 그렇기에, 본 연구팀의 혈액종합검사 온톨로지 구축 연구는 혈액종합검사 스키마 설계에 대한 방향성을 제시하고 있다. 또한, 실제 의료시스템과의 연동이라는 실용적인 측면을 강조함으로써, 본 연구팀이 제안한 온톨로지가 실제 혈액종합검사를 위한 전산시스템에 적용 가능하도록 설계하였다. 끝으로, 본 연구는 혈액종합검사라는 부분적인 진단검사 영역에 대한 온톨로지 구축 연구를 했지만, 본 연구팀이 제안한 온톨로지 구조와 구축 방법론은 추후 조직진단 검사 등 확장된 분야에서의 후속연구 방향을 제시할 수 있을 것이다.

참고문헌

[1] Doi, Kunio. "Computer-aided diagnosis in medical imaging: historical review, current status and future potential." *Computerized medical imaging and graphics* 31.4 (2007): 198-211

[2] 최호섭, et al. "온톨로지 구축 방법과 사례." 정보과학회지 24.4 (2006): 31-44.

[3] 안선주, et al. "의료정보의 의미적 상호운용성 보장을 위한 임상콘텐츠 모델." 정보과학회논문지: 소프트웨어 및 응용 37.12 (2010): 871-881.

[4] 고영만. "시소러스 기반 온톨로지에 관한 연구." 성균관대학교, 정보관리 5 (2006).

[5] Gene Ontology Consortium. "The Gene Ontology (GO) database and informatics resource." Nucleic acids research 32.suppl 1 (2004): D258-D261.

[6] 이상민, 김화선, and 조훈. "인체자원은행의 효율적인 운영을 위한 OWL 기반의 데이터베이스 구축에 관한 연구." (2015).

[7] 이승우, et al. "노령 척추 정보 데이터베이스 및 관리 도구 설계." 한국콘텐츠학회 2010 춘계 종합학술대회 (2010): 539-541.

[8] 김철, 김상균, and 송미영. "논문분석과 구축사례 조사를 통한 한의학 온톨로지 연구동향 분석." 한국한의학연구원 논문집 14.2 (2008): 121-129.

[9] Bodenreider, Olivier. "The unified medical language system (UMLS): integrating biomedical terminology." Nucleic acids research 32.suppl 1 (2004): D267-D270.

[10] 정소나, and 이춘실. "NLM Medical Text Indexer를 활용한 우리나라 의학문헌의 MeSH Semi Indexing 방안." 제 17 회 한국정보관리학회 학술대회 논문집 (2010): 21-28.

[11] Stearns, Michael Q., et al. "SNOMED clinical terms: overview of the development process and project status." Proceedings of the AMIA Symposium. American Medical Informatics Association, 2001.

[12] 김지영. "보건의료정보 용어표준의 국제동향." 전자공학회지 40.1 (2013): 30-38.

사용자의 서비스 사용 성향을 고려한 클라우드 서비스 추천

최 비 오*, 박 준 석§, 황 제 승*, 김 웅 수**, 윤 동 규**, 염 근 혁***

*부산대학교 전기전자컴퓨터공학과, §부산대학교 물류혁신네트워킹연구소,
**부산대학교 정보컴퓨터공학부, ***부산대학교 전기컴퓨터공학부
부산 금정구 장전동 산 30
{ckstkwldls, pjs50, hjs0790, kus9010, lodestar692, yeom}@pusan.ac.kr

요약: 다양한 클라우드 서비스가 생겨나면서 사용자들이 자신이 원하는 서비스를 찾는데 불편함이 발생하였다. 이에 따라 사용자 대신 클라우드 서비스를 찾아서 추천해주는 클라우드 서비스 브로커가 등장하였다. 클라우드 서비스 브로커는 사용자의 기능적, 비기능적 요구사항만을 분석하여 서비스를 추천해주고 있다. 그러나 기능, 비기능 중심의 서비스 추천방식을 탈피하여 잠재적인 사용자를 확보하고 클라우드 서비스 브로커를 더욱 활성화 시킬 필요가 있다. 이를 위해 사용자와 서비스 사용 성향이 비슷한 사용자를 찾아내고, 그들이 사용한 서비스를 추천하는 방법을 제안한다.

핵심어: 클라우드 서비스 브로커, 서비스 추천, 유사 사용자 검출, 클라우드 서비스

1. 서론

IT 기술이 발전하면서 여러 가지의 클라우드 서비스가 생겨났다. 클라우드 서비스는 사용자들에게 편의를 제공하며 일부 사람들에게는 없으면 불편함을 느낄 정도까지 성장하였다. 그와 동시에 클라우드 서비스의 종류와 수가 급격히 증가하면서 사용자들은 자신이 원하는 서비스를 찾는데 불편함이 생겨났다. 이를 해소시킬 필요성이 대두되면서 나타난 방법이 클라우드 서비스 브로커이다.

클라우드 서비스 브로커란 클라우드 서비스 제공자와 클라우드 서비스 사용자 사이에서 중개인 역할을 수행한다. 즉, 클라우드 서비스 사용자의 기능적, 비기능적 요구사항을 수집하고 이들을 대신하여 사용자가 원하는 서비스를 대신 찾아주고 추천해주는 시스템이다[1, 2]. 현재 학계에서는 클라우드 서비스 브로커에 대한 연구가 지속적으로 이루어지고 있으며 IT 전문 업체인 Sky Computing Alliance[3] 에서는 클라우드 서비스 브로커의 구조를 그림 1 과 같이 정의하였다.

그림과 같이 클라우드 서비스를 제공하는 제공자는 SaaS(Software as a Service), PaaS(Platform as a Service), IaaS(Infrastructure as a Service) 세 가지 형태의 서비스를 제공한다.

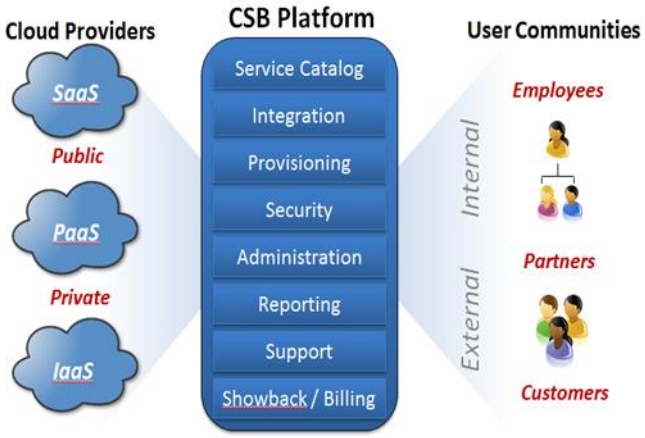


그림 1 클라우드 서비스 브로커 구조

클라우드 서비스 브로커에 대한 관심이 높아지고 이에 대한 연구가 활발히 진행되면서 클라우드 서비스 브로커의 본연의 기능인 사용자의 서비스에 대한 요구를 수집하여 서비스를 추천해 주는 것 이외에 추가적인 서비스를 제공할 필요성이 대두되었다. 대부분의 클라우드 서비스 브로커는 사용자의 요구에 충실한 서비스만 추천해주고 있는 상태이다. 따라서 클라우드 서비스 브로커의 잠재적인 고객을 확보하여 더욱 많은 사용자들이 서비스를 사용할 수 있도록 더 효과적인 서비스를 제공해 주어야 한다[4, 5]. 또한, 사용자의 요구에 충실한 서비스를 추천할 뿐만 아니라 사용자가 관심을 가지고 사용할 가능성이 있는 서비스를 추천하여 사용자의 서비스 선택에 대한 폭을 넓혀주어야 한다.

이를 위하여 본 논문에서는 서비스를 추천 받는 사용자와, 유사한 서비스 사용 성향을 띠는 사용자를 찾아내어 이들이 과거에 사용한 기록이 있거나 현재 사용중인 서비스를 분석하여 유사한 사용자들이 중복하여 많이 사용한 서비스를 추천해 주는 방법을

*** 교신 저자

"이 논문은 2016 년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. NRF-2013R1A2A2A01068256)"

제시한다.
본 논문은 2 장에서 클라우드 서비스 추천 관련연구에 대하여 설명하고 3 장에서는 논문에서 제시하는 유사 사용자를 검출하는 방법과 서비스 추천 방법에 대하여 설명한다. 4 장에서는 진행한 사례연구에 대하여 설명하고 5 장에서는 결론 및 향후 연구에 대하여 설명한다.

2. 관련 연구

2.1 클라우드 서비스 브로커의 서비스 추천 방법

기존의 기능만을 고려한 서비스 추천 방법 이외에 선호도와 프로파일 등의 요소를 고려하여 서비스를 추천하는 연구들이 존재한다.

H. Ma[4] 등의 연구에서는 유사한 사용자를 검출하여 검출된 사용자들이 사용했던 서비스를 추천해주는 방법을 적용하였다. 해당 연구에서는 유사한 사용자를 검출할 때 나이, 교육환경, 직업적 배경, 산업적 배경, 지리적 요소를 고려하였다. 먼저 이들의 지리적인 측면을 고려하여 유사한 사용자를 필터링 하고 이후 네 가지 요소에 대하여 AHP(Analytic hierarchy Process, AHP)[6] 방법을 사용하여 요소들의 유사성을 구한 후 유사 사용자를 검출하였다. 이와 같은 방법으로 서비스를 추천하게 될 경우, 사용자의 서비스 사용 성향을 파악하지 못하고 프로파일 요소만을 반영하기 때문에 추천된 서비스를 사용할 가능성이 낮아진다.

G. Zou[5] 등의 연구에서는 사용자를 모델링하고 사용자의 선호도와 프로파일 정보를 이용하여 유사한 사용자를 검출하였다. 이후 이들이 사용한 서비스를 추천해주었다. 해당 연구에서는 사용자에게 선호도를 수집하기 때문에 관심사가 비슷한 사용자를 검출할 수 있다. 하지만 관심사가 비슷하다고 서비스를 사용하는 성향이 비슷하다고는 보기 힘들기 때문에 추천된 서비스에 관심을 가질 확률도 낮다고 볼 수 있다.

G. Kumar[7] 등의 연구에서는 사용자의 프로파일 정보를 분석하여 유사한 정보를 지닌 사용자를 검출한다. 유사한 사용자로 검출된 사용자들이 사용했던 서비스 목록을 도출한 후, 도출된 서비스들의 QoS(Quality of Service)를 비교하여 높은 QoS 를 지닌 서비스를 추천해주는 방법을 제안하였다. 해당 방법을 이용하여 서비스를 추천하면 분명 품질이 좋은 서비스를 추천할 수 있다. 하지만 서비스의 품질이 좋다고 해서 사용자가 서비스에 만족할 것이라고 주장하기에는 힘들다.

Y. Wang[8] 등의 연구는 사용자의 선호도를 수집하여 서비스를 추천하는 방법을 제안하였다. 선호도로 수집하는 요소는 QoS 이며 서비스를 추천할 때에는 지리적인 요소도 고려하여 서비스를 추천한다. 사용자의 지리적 요소를 고려한 후 특정 위치에서 많이 사용한 서비스를 우선 선별한다. 우선 선별된 서비스의 QoS 와, 사용자의 선호도로 입력 받은 QoS 를 비교하여 가장 적합한 QoS 를 지닌 서비스를 추천해주는 방식이다. 본 방법은 사용자가 원하는 QoS 를 갖춘 서비스를 추천 받을 수 있지만 서비스의 종류가 일관성 없이 전혀 다른 종류의 서비스가 추천될 수도 있다.

이와 같이 클라우드 서비스 브로커를 발전시키기 위한 서비스 추천 연구들이 존재하지만 사용자의 서비스 사용 성향을 분석하여 그에 적합한 서비스를 추천하는 것은 아직 부족한 상태이다.

2.2 Q 방법론

Q 방법론은 영국의 심리학자 윌리엄 스티븐슨이 창안한 연구 기법으로 특정 대상 개인의 주관적 관점, 견해 및 의견, 신념, 태도를 고찰하기 위한 체계적인 토대를 제공한다[9]. 또한 사람과 사람간의 상관 또는 유사성을 규명하는데 주로 사용된다. Q 방법론의 진행 절차는 그림 2 와 같다.

그림 2 Q 방법론 진행 절차

먼저 조사하고자 하는 문제를 명확하게 정의한 후 조사하고자 하는 목적의 특징이 드러나도록 설문지

를 작성한다. 설문 문항은 30 ~ 50 개가 가장 적당하다. 각 설문의 점수는 -4 ~ +4 점으로 응답하며 점수에 대한 가이드라인을 제시한다. 설문이 끝난 후 PQMethod 프로그램을 이용하여 사람간의 유사성을 가지는 요인분석, 해석 작업을 진행하여 결론을 도출한다.

장석용[10] 등의 연구에서는 Q 방법론을 이용하여 운전자의 성향을 분석한 후 어떤 성향을 띠는 운전자가 교통사고를 일으킬 확률이 높은지를 판단하는 연구를 진행하였다. 연구의 결과로 교통사고를 유발하는 운전자들은 6 가지의 공통적인 성향을 가지는 것으로 분류하였고, 교통 법규를 위반하는 운전자들은 5 가지의 공통적인 성향을 가지는 것으로 분류하였다.

이문주[11] 등의 연구는 비즈니스 목적의 여행자가 호텔 객실을 이용할 때 선택하는 객실 속성을 분석하고 계약 만족도와 재계약에 미치는 속성은 무엇인지 분석하는 연구를 진행하였다. 연구 결과로 계약 만족도에 가장 큰 영향을 미치는 속성은 무료서비스였으며, 재계약에 가장 큰 영향을 미치는 요소는 요금 및 예약으로 제시하였다.

유형숙[12]의 연구는 국내의 와인 마니아들을 대상으로 이들이 선호하는 와인시장의 유형을 구분하였다. 이 연구는 PQMethod 프로그램을 사용하였으며 5 가지 유형의 와인 시장이 선호되는 것으로 구분되었다.

박경일[13] 등의 연구는 대학생들이 교수의 유형에 따라 긍정적인 인식을 가지는 유형과 부정적인 인식을 가지는 유형을 구분하는 연구를 진행하였다. 연구는 Q 방법론의 절차대로 진행하였으며 각 각 4 가지 유형을 가지는 것으로 나타났다.

기존의 많은 연구들이 사용자의 유사성을 규명하는데 Q 방법론을 많이 사용하였으며 본 연구에서는 Q 방법론의 일부를 이용하여 유사한 사용자를 검출하고 그들이 중복하여 가장 많이 사용한 서비스를 추천해주는 방법을 제시한다.

3. 유사 사용자 검출 및 서비스 추천

본 장에서는 클라우드 서비스 사용자의 서비스 사용 성향을 분석하여 그와 유사한 성향을 띠는 사용자를 찾아낸 후 중복적으로 많이 사용한 서비스를 추천하는 방법을 제시하며 이 과정은 다음과 같다.

1. 프로파일 구성 요소 기반의 유사 사용자 검출
2. 서비스 사용 성향 기반 유사 사용자 검출
3. 서비스 추천

3.1 절에서 프로파일 구성 요소 기반의 유사 사용자를 검출하는 방법에 대하여 설명하고, 3.2 절에서 서비스 사용 성향 기반의 유사 사용자를 검출한 후 서비스를 추천하는 방법에 대하여 설명한다.

3.1 프로파일 구성 요소 기반의 유사 사용자 검출

유사 사용자를 검출하기 위하여 사용자의 프로파일을 구성하는 feature 요소를 수집한다. 수집하는 feature 요소는 나이, 성별, 직업, 취미, 재정 5 가지 중 하나이며 feature 요소 중 사용자가 선택한 정보의 가중치를 높게 두고 유사 사용자를 검출한다.

직업, 성별, 취미에 대한 유사도 측정 방법은 직업군과 취미 군에 따라서 유사도를 측정한다. 나이, 재정에 대한 유사도 측정 방법은 유클리디안 거리공식을 활용하여 측정하며 수식 1 과 같다.

$$d = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \quad (1)$$

서비스를 추천 받기 원하는 사용자를 x, 유사도 측정 대상 사용자를 y 라고 가정한다면 다음 수식 2 와 같이 적용하여 거리를 측정할 수 있다.

$$d = \sqrt{(x_{age} - y_{age})^2 + (x_{finance} - y_{finance})^2} \quad (2)$$

유클리디안 공식에 의하여 측정된 값 d 는 0 에 가까울수록 두 요소가 가깝다고 판단한다. 또한 d 의 값이 항상 0 과 1 사이의 값을 갖도록 수식 3 과 같이 정규화 한다.

$$\text{normalization } D = \frac{1}{1+d} \quad (3)$$

예를 들어 x 의 나이가 28 이고 재정 21, y 의 나이가 27 이고 재정이 26 이라고 가정한다면 아래와 같이 수식에 대입하여 값을 측정할 수 있다.

$$d = \sqrt{(28-27)^2 + (21-26)^2} = 2.45$$

도출된 유클리디안 거리 값 2.45 를 수식 3 에 대입시키면 0.29 의 값을 구할 수 있다.

위의 과정을 거쳐 직업, 나이, 성별, 취미, 재정에 대한 유사도를 측정하여 1 차 유사 사용자를 검출한 후 다음 단계에서 이들을 대상으로 서비스 사용 성향에 대한 유사성을 측정한다.

3.2 서비스 사용 성향 기반 유사 사용자 검출 및 서비스 추천

클라우드 서비스 브로커의 사용자는 회원가입을 할 때 프로파일 정보와 함께 클라우드 서비스 사용 성향에 대한 설문을 진행한다. 설문은 클라우드 별

표 1 SaaS, PaaS, IaaS 특징

서비스 유형	특징
SaaS (Software as a Service	- 사용자는 클라우드 인프라를 통하여 응용 프로그램을 사용할 수 있다. - 응용 프로그램은 웹 브라우저 (웹 기반 이메일 등)와 같은 씬 클라이언트 인터페이스를 통하여 다양한 클라이언트 기기로부터 접근이 가능하다. - 사용자는 응용 프로그램을 구성 및 설정하기 위한 네트워크, 운영 시스템, 스토리지 또는 개별 응용 프로그램을 필요로 하지 않는다.
PaaS (Platform as a Service)	- 사용자는 클라우드 인프라에 배포된 서비스를 사용하여 응용 프로그램을 창조할 수 있다. - 사용자에게 응용 프로그램 개발을 위한 프로그래밍 언어, 라이브러리, 서비스를 제공한다. - 사용자는 네트워크, 서버, 운영체제, 스토리지를 포함한 클라우드 인프라를 관리하지 않아도 된다.
IaaS (Infrastructure as a Service)	- 사용자에게 저장 공간, 네트워크와 같은 컴퓨팅 자원을 제공한다. - 사용자가 원하는 운영체제, 스토리지에 맞게 컨트롤 할 수 있다.

표 2 설문지 형태

질문		부정적			중립			긍정적		
번호	질문내용	-4	-3	-2	-1	0	+1	+2	+3	+4
1	질문 1 내용	O								
2	질문 2 내용		O							
3	질문 3 내용					O				
..	..	..								
30	질문 30 내용							O		

서비스 사용 성향을 파악하기 위하여 SaaS, PaaS, IaaS 의 특징이 드러나는 질문을 작성하여 각 서비스 타입 10 개 문항, 총 30 개의 질문을 작성하였다.

질문을 작성하기 위하여 각 서비스 타입 특징과 정의를 조사 및 NIST(National Institute of Standards and Technology)[14]를 참고하여 표 1 과 같이 서비스 유형별 특징을 정의하였다.

사용자는 설문의 문항에 긍정적으로 생각하면 +2, +3, +4 에 표시하고 부정적으로 생각하면 -2, -3, -4 에 표시한다. 긍정적이지도 부정적이지도 않으면 -1, 0, +1 에 표시하며 설문지의 형태는 표 2 와 같다.

표 3 은 IaaS, PaaS, SaaS 에 대한 특징을 조사하여 각 타입의 특징이 드러나는 질문을 작성한 것이다. 번호 1 ~ 10 은 IaaS 에 대한 특징이 드러나도록 작성하였고, 번호 11 ~ 20 은 PaaS, 번호 21 ~ 30 은 SaaS 의 특징이 나타나도록 작성하였다. 사용자는 각 질문에 – 4 ~ +4 점의 점수로 응답하여 클라우드 서비스 브로커는 사용자가 어떤 유형의 서비스에 관심이 있는지 파악할 수 있다.

클라우드 서비스 브로커는 사용자의 프로파일 중 요도를 수집하여 선정된 유사 사용자들을 대상으로 설문 응답 점수를 비교한 후 최종 유사 사용자를 검출한다. 이때는 수식 4 에 제시한 상관계수 공식을 이용하여 유사도를 측정한다.

표 3 사용자 서비스 성향 파악 질문 내용

번호	질문 내용
1	인터넷 공간에 파일을 자주 저장한다.
2	많은 양의 저장 공간이 필요하다.
3	개인의 목적을 위한 서버 환경이 필요하다.
	…
11	자신이 원하는 응용 프로그램을 직접 만들기 원한다.
12	프로그램을 만들기 위한 기능을 제공받기를 원한다.
13	자신이 만든 프로그램을 다른 사람과 함께 사용하고 싶다.
	…
21	인터넷을 이용하여 간편하게 응용 프로그램을 사용하기를 원한다.
22	특정 컴퓨터에서 사용하는 프로그램을 다른 컴퓨터에서도 사용하기를 원한다.
23	로컬 환경에서 진행한 작업을 인터넷 환경에서 이어서 작업 하기를 원한다.
	…

$$\mathrm{Corr}(x,y) = \frac{Cov(x,y)}{S_x S_y} \quad (4)$$

상관계수는 x 와 y 의 공분산 값에 x 의 표준편차와 y 의 표준편차의 곱을 나눈 값으로 측정된다. 공분산은 둘 이상의 변량(x, y)이 서로 관련성을 가지며 분포하는 모양을 전체적으로 나타내는 분산이며, 표준편차는 통계집단의 단위의 계량적 특성 값에 관한 산포도를 나타내는 도수 특성 값을 말한다. 공분산은 수식 5 에 의하여 측정된다.

$$\mathrm{Cov}(x,y) = \mathrm{E}(xy) - \mathrm{E}(x)\mathrm{E}(y) \quad (5)$$

수식에 표시된 E(x)는 x 의 평균값을 나타내고 E(y)는 y 의 평균값을 나타낸다. 그리고 E(xy)는 x 와 y 의 곱의 평균을 나타내며 E(x)E(y)는 x 의 평균값 E(x)와 y 의 평균값 E(y)의 곱을 나타낸다

본 수식 5 를 이용하여 x 와 y 의 공분산을 측정한 후 표준편차 값을 나누면 상관계수가 측정된다. 표준편차는 수식 6 을 이용하여 측정한다.

$$S = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \quad (6)$$

표준편차는 구하고자 하는 집합 원소의 개수 N 과 각 각의 값 x 를 이용하며 $\overline{x}$ 는 x 의 평균값을 의미한다. 본 논문에서는 질문의 개수가 N 이 되어 N 은 30 이 되고, x 의 값은 각 설문에 대한 응답 점수가 대입된다. 본 수식을 이용하면 표준편차 값이 측정되고 앞서 구한 공분산 값과 표준편차 값을 상관계수를 구하는 수식에 대입하면 최종적인 서비스 사용 성향에 대한 상관계수가 측정된다.

측정된 상관계수는 항상 -1.0 ~ +1.0 사이의 값을 가진다. 측정된 상관계수 값이 -1.0 에 가까울수록 음의 선형관계에 있고, +1.0 에 가까울수록 양의 선형관계에 있다. 이를 판별하는 방법은 표 4 의 상관계수 기준표[15]에 따라 상관관계를 결정한다.

표 4 상관관계 기준표

상관계수 값	상관계수에 따른 유사성
-1.0 ~ -0.7	강한 음의 선형관계
-0.7 ~ -0.3	뚜렷한 음의 선형관계
-0.3 ~ -0.1	약한 음의 선형관계
-0.1 ~ +0.1	거의 무시될 수 있는 선형관계
+0.1 ~ +0.3	약한 양의 선형관계
+0.3 ~ +0.7	뚜렷한 양의 선형관계
+0.7 ~ +1.0	강한 양의 선형관계

본 논문에서는 상관관계 기준표에 따라 측정된 상관계수 값이 양의 선형관계를 가지는 0.1 이상이 측정될 경우 유사 사용자로 판단한다. 이 과정에서 검출된 사용자는 사용자의 선호도와 서비스 사용 성향이 반영된 최종 유사 사용자이다.

이후 최종 유사 사용자들이 중복하여 많이 사용한 서비스를 추천 받는 사용자에게 추천해주면 서비스 추천이 완료된다.

4. 사례 연구

본 장에서는 3 장에서 제시한 방법에 대한 사례 연구를 진행한다. 사례연구를 진행하기 위하여 각 서비스 타입의 특징이 드러나는 질문을 작성하였으며 설문을 진행하여 데이터를 수집하였다. 설문은 구글에서 제공하는 설문지를 사용하였고 설문지 URL 을 E-mail, Message, SNS 등 각종 정보통신 매체를 이용하여 배포하고 정보를 수집하였다

사용자는 그림 3 과 같은 인터페이스를 통하여 설문에 참여하였다.

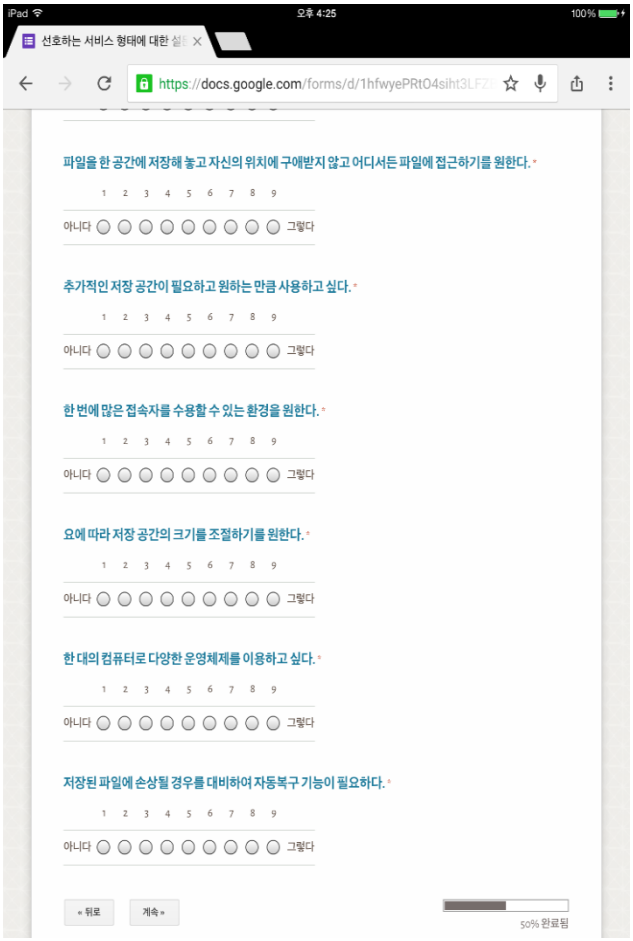


그림 3 설문지 인터페이스

설문의 응답은 GoogleDocs 에서 확인하였고 3 주간 (2015. 10. 22 ~ 2015. 11. 05) 진행하였으며 총 287 명의 인원이 설문에 참여하였다.

	C	D	E	F	G
1	Job	Hobby	Finance	UsedService	Q1
2	전문직 (공학전문가, 보건전문가 등)	미디어활동 (음악듣기, 게임, SNS 등)	19	Ndrive	4
3	전문직 (공학전문가, 보건전문가 등)	미디어활동 (음악듣기, 게임, SNS 등)	17	Ndrive	1
4	관리직 (관리자 등)	오락활동 (카드놀이, 바둑, 카지노, 복권 등)	34	없음	2
5	관리직 (관리자 등)	여행 (등산, 드라이브, 해외여행 등)	22	엔드라이브, 아이폰 클라우드 서비스	2
6	사무종사자 (경영, 회계 등)	여행 (등산, 드라이브, 해외여행 등)	18		4
7	전문직 (공학전문가, 보건전문가 등)	여행 (등산, 드라이브, 해외여행 등)	23	Google Docs, Ndrive, Dropbox	4
8	전문직 (공학전문가, 보건전문가 등)	스포츠활동	22	NDrive, GoogleDrive	0
9	전문직 (공학전문가, 보건전문가 등)	오락활동 (카드놀이, 바둑, 카지노, 복권 등)	18	없음	-4
10	관리직 (관리자 등)	여행 (등산, 드라이브, 해외여행 등)	26	Ndrive	2
11	서비스종사자 (음식, 조리, 혼례관련 등)	관람, 감상 (영화관람, 미술괄람 등)	24	Ndrive	0
12	관리직 (관리자 등)	미디어활동 (음악듣기, 게임, SNS 등)	18		-3
13	서비스종사자 (음식, 조리, 혼례관련 등)	스포츠활동	17	n drive	1
14	단순노무종사자 (건설, 청소, 경비 등)	관람, 감상 (영화관람, 미술괄람 등)	16	Ndrive	0
15	전문직 (공학전문가, 보건전문가 등)	가사활동 (요리, 청소, 세탁 등)	23	naver office ndrive	-1
16	전문직 (공학전문가, 보건전문가 등)	여행 (등산, 드라이브, 해외여행 등)	25	Ndrive	0
17	관리직 (관리자 등)	미디어활동 (음악듣기, 게임, SNS 등)	18	Ndrive	-3
18	사무종사자 (경영, 회계 등)	여행 (등산, 드라이브, 해외여행 등)	22	ndrive, icloud	1
19	전문직 (공학전문가, 보건전문가 등)	여행 (등산, 드라이브, 해외여행 등)	24	네이버	0
20	전문직 (공학전문가, 보건전문가 등)	관람, 감상 (영화관람, 미술괄람 등)	19	엔드라이브, 다음클라우드, 구글독스, 구글 드라이브	2

그림 4 수집된 사용자 정보 및 서비스 사용 성향 설문 점수

Num	Gender	Age	Job	Hobby	Finance	UsedService
A	남자	27	관리직	미디어활동	22	Ndrive
B	남자	30	관리직	미디어활동	24	NDrive, DaumCloud, GoogleDocs, GoogleDrive
C	남자	30	관리직	미디어활동	23	NDrive, GoogleDrive
D	남자	30	관리직	미디어활동	23	NDrive, Dropbox, NaverOffice, GoogleDocs, GoogleDrive
E	남자	30	관리직	미디어활동	24	NDrive
F	남자	31	관리직	미디어활동	23	Dropbox
G	남자	27	관리직	미디어활동	22	Ndrive, GoogleDocs, NaverOffice
H	남자	28	관리직	미디어활동	21	NDRIVE, GoogleDocs
I	남자	27	관리직	미디어활동	23	googledrive, daumcloud
J	남자	29	관리직	미디어활동	21	Ndrive, GoogleDocs
K	남자	29	관리직	미디어활동	24	Ndrive

그림 5 검출된 유사 사용자

설문 응답 중 한 가지의 점수로 응답한 사용자의 데이터는 삭제하고 265 명의 데이터를 사용하였다. 수집된 데이터는 그림 4 와 같이 CSV 파일로 저장하였다.

사례 연구를 진행하기 위하여 다음 표 5 와 같이 서비스를 추천 받는 사용자의 데이터를 사용하여 유사 사용자를 검출하고 서비스를 추천 받기까지의 과정을 제시한다.

표 5 사용자 정보

성별	나이	직업	취미	재정
남자	29	관리직	미디어활동	2,200,000

사용자의 정보가 표 5 와 같고 사용자가 가장 중요하게 생각하는 정보가 직업이라고 선택하였을 때, 265 명의 데이터 중 직업이 관리직인 사용자들을 모두 검출한다. 이후 이들을 대상으로 성별이 남자이면서 취미가 미디어활동인 사용자들을 검출한다.

위 과정을 거쳐 추출된 사용자들을 대상으로 나이와 재정에 대한 유클리디안 거리를 측정하여 유사 사용자를 검출한다. 재정은 효율적인 계산을 위하여 십만 단위는 없애고 계산한다. 이때 3 장에서 정의한 수식 (1)의 유클리디안 거리공식을 이용한다.

도출된 유클리디안 거리 값을 수식 (2)를 이용하여 정규화시킨다. 정규화된 값은 항상 0 과 1 사이의 값을 가지므로 1 과 가까운 수를 가진 사용자들을 1 차 유사 사용자로 검출한다.

본 과정을 수행하여 그림 5 와 같은 총 11 명의 유사 사용자가 검출되었다. 검출된 11 명의 1 차 유사 사용자를 대상으로 서비스 사용 성향에 대한 유사성을 측정하는 과정을 진행하였다. 상관계수 측정 공식에 따라 공분산과 표준편차를 구하였다. 서비스를 추천 받기 원하는 사용자의 설문 점수는 표 6 과 같으며 이 점수와 11 명의 설문점수에 대한 상관계수를 측정한다.

상관계수를 측정하기 위하여 공분산과 표준편차 값을 측정한다. 먼저 3 장에서 정의한 공분산을 구하는 수식 (5)를 이용한다.

서비스를 추천 받는 사용자를 x 라 두고 유사 사용자 대상 11 명을 하나씩 y 로 두고 수식에 대입하여 값을 구한다. 수식에 따라 x 의 설문점수 평균을 구

하면 E(x) = 0.63 의 값이 도출된다. 나머지 11 명에 대하여 평균을 구한 후 측정된 공분산 값은 표 6 과 같다.

표 6 사용자 설문 점수

설문 번호	응답 점수	설문 번호	응답 점수	설문 번호	응답점수
1	2	11	1	21	2
2	2	12	-1	22	2
3	1	13	2	23	2
4	2	14	-1	24	2
5	3	15	1	25	3
6	-2	16	-1	26	2
7	0	17	0	27	1
8	1	18	1	28	-2
9	-4	19	-1	29	-1
10	3	20	-3	30	2

공분산 값을 도출한 후 각 사용자들의 설문 응답 점수에 대한 표준편차 값을 구한다. 마찬가지로 3 장에서 정의한 수식 (6)에 따라 사용자의 표준편차를 측정하면 1.8 이 도출된다. 같은 방법으로 11 명의 사용자에 대한 표준편차 값을 측정하면 표 7 과 같은 값이 측정된다.

표 7 공분산 수치

User	Cov	User	Cov
A	0.1	G	0.7
B	2.2	H	0.4
C	1.6	I	1.5
D	2.6	J	0.9
E	2.3	K	0.4
F	1.4		

측정된 공분산 수치와 표준편차 수치를 수식 (4)의 상관계수 공식에 대입하면 사용자와 유사 사용자간의 상관계수가 측정된다. 측정된 상관계수 값은 표 8 과 같다.

표 8 유사 사용자의 표준편차 수치

User	STD	User	STD
A	1.8	G	2.4
B	3.0	H	1.7
C	2.2	I	3.0
D	2.7	J	1.5
E	3.2	K	1.7
F	2.4		

측정된 상관계수 값은 표 3 의 상관계수 기준표에 따라 상관관계를 판단한다. 사용자 A 는 상관계수의 값이 0.0 으로 거의 무시될 수 있는 선형관계에 있으므로 유사 사용자 리스트에서 제외시킨다

표 9 사용자간의 상관계수 수치

User	Corr	User	Corr
A	0.0	G	0.2
B	0.4	H	0.1
C	0.4	I	0.3
D	0.5	J	0.3
E	0.4	K	0.1
F	0.3		

사용자 F, G, H, I, J, K 는 약한 양의 선형관계에 있고, 사용자 B, C, D 는 뚜렷한 양의 선형관계에 있으므로 사용자 A 를 제외한 B~K 의 사용자를 최종 유사 사용자로 검출한다.

마지막으로 이들이 중복하여 많이 사용한 서비스를 사용자에게 추천해주면 서비스 추천이 종료된다. 그림 5 에 제시된 검출된 유사 사용자의 UsedService 목록을 참고하여 이들이 중복하여 가장 많이 사용한 NDrive 서비스를 추천한다.

5. 결론 및 향후 연구

본 논문에서는 사용자의 프로파일 구성 요소와 서비스 사용 성향을 고려하여 유사한 사용자를 검출하고 그들이 중복하여 많이 사용한 서비스를 추천하는 방법에 대하여 제시하였다.

유사한 사용자를 찾기 위하여 프로파일 정보를 기반으로 사용자의 주관적인 의견을 반영한 사용자를 우선 선별하였으며, 이후 이들의 서비스 사용 성향 설문 점수를 이용하여 사용자와 유사 사용자간의 서비스 사용 성향에 대한 상관관계를 측정하는 방법을 정의하였다.

기존 연구들에서는 유사 사용자를 검출하는 과정에서 사용자의 프로파일 정보만을 이용하여 유사 사용자를 검출하는 방법을 사용하였다.

이는 단순히 사용자의 정보만 비슷한 사용자를 검출할 뿐이지 사용자의 클라우드 서비스에 대한 사용 성향은 고려하지 못한다는 문제가 있었다.

본 논문은 기존의 클라우드 서비스 브로커에서 제공하는 사용자의 요구에 맞는 서비스를 추천하는 것 이외에 사용자의 서비스 사용 성향을 고려하여 서비스를 추천하기 때문에 사용자가 추천된 서비스에 좀 더 관심을 가질 수 있다.

향후 연구로는 사용자의 서비스 사용 성향을 파악함에 따라 축적되는 데이터를 분석하여 사용자 유형별 서비스 사용 패턴을 정의하는 방법에 대하여 연구할 것이다. 이는 추후 클라우드 서비스 브로커에 새로운 사용자가 유입되었을 때 서비스를 더욱 정확하게 찾아주는 방법이 될 것이다.

참고문헌

[1] Gartner, http://www.gartner.com/it-glossary/cloud-servicces-brokeragecsb

[2] R. Bohn, J. Messina, F. Liu, J. Tong, and J. Mao, “NIST Cloud Computing Reference Architecture”, NIST Special Publication, Jul. 2011

[3] Sky Computing Alliance, http://www.skycomputingalliance.com/wp/index.php/cloud-services-brokeragescsbs/, (Visited on 6 Nov. 2015)

[4] H. Ma, Z. Hu L. Yang, and T. Song, "User Feature-Aware Trustworthiness Measurement of Cloud Services via Evidence Synthesis for Potential Users", Journal of Visual Languages and Computing, pp. Vol. 25, Num. 6, 791-799, Dec. 2014

[5] G. Zou, Y. Gan, J. Zheng and B. Zhang, "Service Composition and user Modeling for Personalized Recommendation in Cloud Computing", International Conference on Computing, Communication and Networking Technologies, pp. 1-7, July, 2014

[6] T. L. Saaty and L. G. Vargas, "Models, Methods, Concepts & Applications of the Analytic Hierarchy Process", Springer Science & Business Media, 2001

[7] G. Kumar and K. Morarjee, "Ranking Prediction for Cloud Services from The Past Usages", International Journal of Computer Sciences and Engineering, Vol. 2, Num. 9, pp. 22-25, Sep. 2014

[8] Y. Wang, J. Zhou and H. Tan, “CC-PSM: A Preference-Aware Selection Model for Cloud Service Based on Consumer Community”, Mathematical Problems in Engineering, July, 2015

[9] 사이먼 와츠, 폴 스테너 “Q 방법론 연구의 실행, 이론 · 방법 · 해석”, 커뮤니케이션북스, 2014

[10] 장석용, 정헌영, 이원규, 고상선, “Q 분석 방법을 이용한 운전자 운전성향별 유형화에 관한 연구”, 대한교통학회지 제 26 권 제 1 호, pp. 165-180, 2008

[11] 이문주, 한진수, “BTL 중심의 브랜드 접촉점에 따른 외식프랜차이즈기업 소비자 유형 분류: Q 방법론의 적용”, 한국호텔외식경영학회 제 20 권 제 6 호, pp. 58-63, 2011

[12] 유형숙, “와인 마니아의 국내 와인시장에 관한 인지도 분석 - Q 방법론을 사용하여 -”, 동북아관광학회 제 9 권 제 1 호, pp. 123-145, 2 월, 2013

[13] 박경일, 주재훈, 이은희, 오원옥, 염현이, 권진아, “Q 방법론을 활용한 교수자 유형 탐색 연구”, Korean Journal of Social Welfare Education, Vol.20, pp. 230-259, 2012

[14] NIST, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, (Visited on 7 Nov. 2015)

[15] 김석우, 기초통계학, 학지사, 2007

이기종 시맨틱 온톨로지 시스템의 통합 검색

황상원, 남영광

연세대학교 컴퓨터정보통신공학부
강원도 원주시 연세대길 1
{arsenal, yknam}@yonsei.ac.kr

요약: 본 논문에서는 서로 다른 스키마를 가진 두 개의 분산 온톨로지 시스템의 N-Triple 데이터를 통합 검색하는 방법을 제안한다. 사용자는 통합 온톨로지 스키마를 활용하여 질의를 생성하며, 통합 시스템이 해당 질의를 분석하여 각 분산 시스템에 질의를 전송하고 결과를 제공받기 때문에 마치 하나의 시스템에서 검색하는 것과 동일한 효과를 제공받는다. 그리고 이종의 데이터를 서로에게 적합한 형태로 변환하지 않고 통합 검색을 수행하기 때문에, 물리적인 통합 시스템 구축 비용의 절감과 통합 검색 시스템 구축이 용이하다는 장점이 있다.

핵심어:통합정보검색,온톨로지,질의재생성,SPARQL

1. 연구배경

대부분의 시멘틱 웹 시스템은 전통적인 RDB 기반 시스템 혹은 각 검색 시스템에 저장되어 있는 데이터를 기반으로 자신들만의 고유한 온톨로지 스키마를 구성하여 N-Triple[1] 데이터로 가공 후 구축하였다. 시간이 지나면서 이러한 시스템들에 대해 의미가 같지만 표현 방법이 다른 공통 데이터를 기반으로 시스템 통합 검색의 필요성이 생겨났다. 이와 같은 이기종의 시스템에서 일부 같은 데이터를 활용하여 서로 보완적인 데이터를 검색 할 수 있도록 시스템을 구축하면 사용자에게 의미 있는 결과를 제공해줄 수 있다.[2] 그러나 이기종 RDB 시스템을 물리적으로 통합하는 것은 쉬운 일이 아니며, 시스템 별 고유 스키마로 구축된 시맨틱웹의 통합 검색 시스템 구축 또한 많은 비용이 든다.

본 연구에서는 이와 같은 비용 발생을 최소화 하고자 의미적으로 연계가 가능한 이기종 온톨로지 시스템에 대한 통합 방법을 제안한다. 이 시스템은 구조가 서로 다른 시스템간에 대해 통합 질의를 통하여 각 시스템에 접근하여 SPARQL 질의를 수행하며, 사용자에게는 분산 시스템의 존재에 대하여 투명성을 제공한다. 이와 같은 방법은 시스템을 물리적으로 통합하지 않기 때문에 시스템 통합 비용을 줄일 수 있다.

2. 통합 검색 시스템

본 논문에서 제안하는 시스템은 분산되어 존재하는 두 개의 다른 시스템에 도메인이 다른 두 개의 스키마를 가진 온톨로지에 대하여 사용자가 통합 스키마를 참조하여 통합 질의를 수행하고 그 결과를 단일 시스템에서 검색하는 것과 동일한 결과를 얻도록 하는 기능을 제공한다.

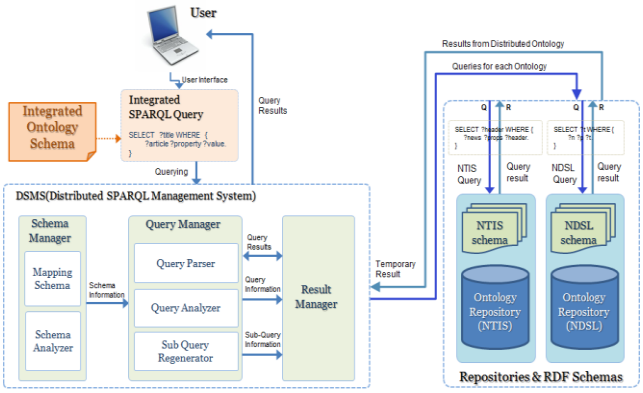


그림 1 통합 검색 시스템 구조도

이 시스템에서 사용하는 통합 스키마는 분산 시스템에 가지고 있는 온톨로지 스키마 중 의미는 같지만 표현이 다른 클래스/속성을 연결하여 생성한다. 매핑된 스키마 구조는 아래 그림과 같다.

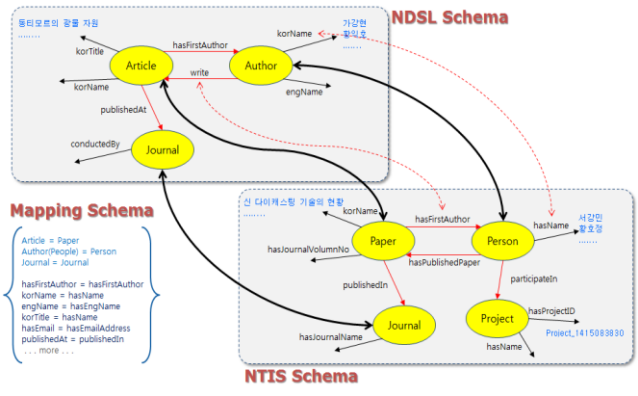


그림 2 매핑된 NDSL, NTIS 스키마

시스템은 스키마 관리기, 질의 관리기, 결과 관리기, 분산 데이터 저장소로 구성된다. 스키마 관리기는 각 시스템의 스키마와 통합 스키마를 관리한다. 질의 관리기는 각 분산 스키마와 통합 스키마를 기반으로 질의 분석 후 서브질의 생성기로 정보를 전달하여 통합 질의를 분산된 각 시스템에 유효한 질의로 변경한다. 결과 관리기는 서브 질의를 수행하여 얻은 결과를 질의 유형에 따라 타 시스템에 대한 FILTER 절로 사용되거나 조인 등의 처리를 수행하여 최종 질의 결과를 사용자에게 보여줄 수 있도록 데이터를 처리한다. 아래 표 1 은 질의 유형을 분석한 것으로 SELECT, FILTER 구문에 포함된 변수에 따라 질의 유형을 분류하였다.

표 1 통합 질의 유형

유형		SELECT 구문 변수			FILTER 절 변수		
대	중	모두 존재	시스템1에만 존재(NDSL)	시스템2에만 존재(NTIS)	모두 존재	시스템1에만 존재(NDSL)	시스템2에만 존재(NTIS)
1	1		O			O	
				O			O
	2		O		O	O	
				O	O		O
2		O			O		
3	1	O				O	
		O					O
	2	O			O	O	
		O			O		O
4	1		O		O		
				O	O		
	2	O	O		O		
		O		O	O		
5	1		O				O
				O		O	
	2		O		O		O
				O	O	O	
	3	O	O		O		O
		O		O	O	O	

유형 분류 방법은 통합 질의가 하나의 시스템에서 결과를 추출하는지 혹은 두 시스템에서 추출하는지, 조합을 해야 하는지 등의 조건에 따라 구성된다.. 그리고 분류된 유형에 따라 각 분산 시스템에서 수행될 서브 질의 재생성하고 질의를 수행한다.

3. 실험 및 결과

제안된 시스템에 1-1 ~ 5-3 까지 유형의 질의를 입력하고 각 분산 시스템에서 추출되는 트리플 수와 수행 시간, 통합 질의 결과 트리플 수와 수행 시간을 추출하였다. 아래 표는 유형 5-1 의 통합 질의(좌)와 개별 시스템 1(NDSL, 중), 개별 시스템 2(NTIS, 우)로 변경되어 전송 될 질의의 예제다. 아래 그림은 유형 5-1 의 질의 수행 시나리오다.

표 2 질의 유형 5-1 예

Integrated query	NDSL query	NTIS query
SELECT ?o_articleName WHERE { ?outstandingArticle rdf:type ys:OutstandingArticle. ?outstandingArticle ys:**hasFirstAuthor** ?author. ?outstandingArticle ys:korTitle ?o_articleName. ?project rdf:type ys:Project. ?project ys:hasName ?projName. ?author ys:participateIn ?project. **?author ys:korName ?authorName.** FILTER (regex(str(?projName),"통신방송 융합형 광가입자 시스템 개발")) }	SELECT ?o_articleName WHERE { ?outstandingArticle rdf:type ndsl:OutstandingArticle. ?outstandingArticle ndsl:hasFirstAuthor ?author. ?outstandingArticle ndsl:korTitle ?o_articleName. ?author ndsl:korName ?authorName. FILTER (regex(str(?authorName), "김성무") || regex(str(?authorName), "김태호") || regex(str(?authorName), "장경준") || … regex(str(?authorName), "이재훈")) }	SELECT [NO_SELECT_VAR] WHERE {?project rdf:type ntis:Project. ?project ntis:hasName ?projName. ?author ntis:participateIn ?project. ?author ntis:hasName ?authorName. FILTER (regex (str (?projName) , "통신방송 융합형 광가입자 시스템 개발")) SUB_FILTER }

표 3 질의 유형 5 수행 결과

Query #	Type 5					
	#of NDSL result tuple	execution time (ms)	# of NTIS result tuple	execution time (ms)	# of answer tuple	total execution time (ms)
1(5-1)	1	179	39	502	1	802
2(5-1)	1	35	3	537	3	651
3(5-1)	1	22	24	495	1	548
4(5-2)	1	53	12	530	1	611
5(5-2)	1	43	6	521	1	594
6(5-2)	5	23	54	517	32	575
7(5-3)	1	33	32	490	1	550
8(5-3)	0	36	54	520	0	600
9(5-3)	2	27	456	514	2	552
10(5-3)	3	44	1752	556	3	632

4. 결 론

본 연구에서는 서로 다른 두 개의 온톨로지 시스템의 데이터 변환 및 물리적 통합을 하지 않고 통합 스키마를 이용하여 검색할 수 있는 시스템을 제안하였다. 분리된 시스템에 대해 의미는 같지만 표현 형태가 다른 스키마들을 연결하여 통합 스키마를 생성하고, 사용자는 관리자가 생성한 스키마를 활용하여 통합 SPARQL 질의를 생성함으로써 사용자는 각 개별 온톨로지에 대한 인지 없이 질의를 수행하여 시스템 재 구축에 대한 추가 비용 발생을 줄일 수 있다.

향후에는 결과 통합 알고리즘의 성능 개선과, aggregation 함수와 단일 질의 내에 포함되어 있는 서브 질의에 대한 분석을 통하여 질의 유형 확장에 대한 추가적인 연구를 수행하면 더욱 활용도가 높은 시스템이 될 것이라 판단된다.

참고문헌

[1] N-Triple, http://www.w3.org/TR/n-triples/

[2] Q. Abir, A. Dimitre, H. Jeff, "ISENS: A System for Information Integration, Exploration, and Querying of Multi-Ontology Data Sources," Proceedings of the 2009 IEEE International Conference on Semantic Computing, pp. 330-335, 2009.

소프트웨어 성능 가시화를 위한 툴 체인 개발

강건희*, 박보경, 장우성, 황준순, 권하은, 이한솔, 이현준, 김영철

*홍익대학교 소프트웨어공학연구실
세종특별자치시 조치원읍 세종로 2639
{kang, park, jang, hwang, kwon, hslee, hjlee, bob}@selab.hongik.ac.kr

요약: 오늘날의 소프트웨어 산업은 점점 커지며 고품질에 대한 이슈가 대두되고 있다. 하지만 커지는 산업에 비해 시장 출하 기간의 단축상황으로 산업현장에서는 빠른 개발을 위한 코드 중심의 개발을 하게 된다. 그 결과 저 품질의 소프트웨어가 양산된다. 경쟁력을 가진 고품질의 소프트웨어 생산을 위한 인력과 비용이 우리나라는 부족하다. 그래서 소프트웨어 가시화가 필요하다. 본 논문에서는 소프트웨어의 성능(반응속도)가시화를 위한 도구의 구성과 가시화 방법을 소개한다. 예를 들면, 프로파일러를 통해 소프트웨어 성능정보, 그리고 기존 소프트웨어 가시화의 도구에서 추출하는 소프트웨어 구조정보를 가지고 소프트웨어의 가시화한다. 제안한 방법은 개발자뿐 아니라 다양한 관계자들이 소프트웨어의 성능에 대한 이해가 쉬울 것이다.

핵심어: 소프트웨어 성능, 소프트웨어 가시화, 역 공학

1. 서론

한국의 소프트웨어 시장규모는 전년 대비 5.8% 성장한 110 억 달러로 파악되며, 2017 년까지도 성장의 지속이 전망된다[1]. 그러나 우리나라 소프트웨어 규모는 세계 소프트웨어 시장에서 차지하는 비중이 약 1%에 불과하다. IT 강국이지만 하드웨어 쪽으로 치우치는 기현상이 보인다. 또한 시가총액 상위 ICT 기업에서 Google, Microsoft, Facebook, Amazon 등 외국 소프트웨어 기업이 주도하며, 반면 국내 소프트웨어 기업의 글로벌 경쟁력은 아직 부족하다. 그리고 국내 대기업의 소프트웨어 개발은 SW 공학적 접근의 개발을 하고 있다. 하지만 중소기업은 SW 공학적용을 위한 전문 인력과 개발시간, 자본 등의 부족으로 구현 중심의 개발을 한다. 이러한 소프트웨어 산업의 대부분인 중소기업의 현실을 개선 위해서 프로젝트의 시작부터 소프트웨어 공학과 다양한 소프트웨어 개발 프로세스의 적용과 이를 위한 인력을 배치가 필수다. 하지만 이미 개발이 시작된 프로젝트 중간에 적용하기란 쉽지 않고, 기 개발된 소프트웨어에 적용은 소용이 없다. 기존연구[2-4]에서 역 공학을 통해 기존의 소프트웨어의 소스코드를 통해 품질지표의 적용과 가시화에 초점을 뒀다. 기존 소프트웨어의 가시화를 통해 소프트웨어의 구조를 쉽게 파악하고 소프트웨어 품질지표도 함께 적용하여 소프트웨어의 품질판단과 어느 부분이 품질상의 문제점이 있는지 파악 가능하다. 그래서 프로파일러를 통해서 소프트웨어 성능정보를 얻고 이를 가시화한다.

본 논문의 구성은 다음과 같다. 2 장 관련연구는 소프트웨어 가시화와 소프트웨어 성능에 대해 설명한다. 3 장은 소프트웨어 성능 측정 위한 도구에 대해 기술한다. 마지막으로 4 장에서는 결론을 언급한다.

2. 관련 연구

2.1 소프트웨어 성능

전통적인 알고리즘에서 성능은 중요한 명령어의 수행 횟수 및 메모리 공간의 점유율로 정의한다. 하지만 컴퓨터 사이언스 중 하드웨어의 성능은 처리속도, 채널 용량, 지연 속도, 대역폭, 처리량, 전력 소모 등이 있고 소프트웨어 성능은 신뢰성, 코드의 크기 및 무게, 반응속도 등도 있다[5,6].

소프트웨어성능에서 소프트웨어 가용성은 평균고장발생간격(MTBF)에서 평균 수리시간을 뺀 뒤 평균 수리시간으로 나눈 값의 100 을 곱하여 소프트웨어가 구동되는 전체 시간에서 실제 기능을 하는 시간을 백분율로 계산한다. 평균고장발생간격(MTBF)은 오류가 발생할 때까지 응용프로그램이 실행되는 평균시간이다. 그리고 평균복구시간(MTTR)은 오류가 발생한 후 서비스를 복구 및 복원하는데 필요한 평균시간을 의미한다. 그리고 코드의 크기와 무게는 모바일 시스템과 같은 임베디드 시스템에서 자원이 제한되어있기 때문에 소프트웨어의 크기 가 성능에 많은 영향을 끼친다. 소프트웨어 반응속도는 시스템 혹은

† 이 논문은 2015 년 교육부와 한국연구재단의 지역혁신창의인력양성사업(NRF-2015H1C1A1035548)과 2015 년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2013R1A1A2011601).

실행단위에 입력이 주어진 뒤 반응까지 걸린 시간을 의미한다. 반응속도가 길어질수록 시스템의 속도는 느려진다.

본 연구에서는 다양한 소프트웨어 프로젝트의 이해관계자가 쉽게 이해되는 소프트웨어 메소드 단위의 소프트웨어 반응속도를 프로파일러를 통해 측정 및 가시화 한다.

2.2 소프트웨어 가시화

성공적인 소프트웨어 개발 및 관리를 위해 소프트웨어 개발, 프로세스, 테스트 자동화, 그리고 품질인증 등은 필수적이다. 하지만 이러한 일을 수행하는 자원과 전문인원이 너무나 부족하다. 소프트웨어 가시화는 유지보수와 품질관리의 효율성을 향상하는 기법이다. 소프트웨어 가시화에는 시각화와 문서화가 있다[6].

첫째, 시각화는 소프트웨어 개발의 가장 어려운 점인 소프트웨어 비가시성을 극복하고 전체 소프트웨어 개발 과정을 파악하여 품질관리를 수행하는 방안이다. 둘째, 문서화는 기업 혹은 단체의 개발 노하우 관리 및 내부 인력간의 업무 이해도 향상과 특정 상황에서 외부와의 의사소통을 위한 방안이다.

본 연구는 기존연구[2-4]와 다르게 코드의 복잡도(결합도, 응집도)뿐만 아니라 소프트웨어의 성능 정보를 가시화 한다.

3. 소프트웨어 성능 측정을 위한 도구

그림 1 은 소프트웨어 성능 가시화를 위한 도구의 구성도이다. 소프트웨어성능 가시화를 위한 툴체인에는 기존 소프트웨어 가시화를 위한 툴체인[2]에 소프트웨어 성능의 좋지 않은 패턴을 추출하기 위한 RuleChecker(PMD[7])와 소프트웨어의 동적분석을 위한 Profiler(Hprof[8]), Profiler 에서 추출된 데이터를 Xml 데이터로 정제하기 위한 HprofDataExtractor 를 추가하였다. 소프트웨어 성능 가시화를 위해서는 소프트웨어의 정보 추출-> DB 저장-> 품질지표정의->가시화 총 4 단계를 거치게 된다.

- 1 단계: 소프트웨어의 정보추출을 위해서 소프트웨어 구조정보, RuleChecker 를 통한 성능저해요소 정보, Profiler 를 통한 성능 추출이 수행 한다.
  - ■소프트웨어 구조정보: 소프트웨어 구조정보를 추출하기 위해서는 SourceNavigator 에 소스코드를 입력하여 SNDB 파일을 추출한 뒤 바이너리로 되어있는 정보를 DBdump.exe 로 볼 수 있는 문자열로 변경, 추출한다.
  - ■소프트웨어 성능저해요소정보: 성능저해요소의 대한 패턴을 XPath 로 정의하고 RuleChecker(PMD)에 적용하여 XML 정보로 추출한다.
  - ■Profiler 를 통한 성능정보: 실제 소프트웨어를 프로파일러와 함께 구동하여 소프트웨어 메소드 단위의 구동시간과 호출횟수정보를 추출한다. 하지만 추출된 정보가 규칙이 없는 문자열 값이기 때문에 HprofDataExtrator 프로그램을 통해서 Xml 형식의 정제화된 정보로 추출한다.

- 2 단계: CreateDB 프로그램을 통해서 DBdump 를 통한 SNDB 의 정보를 해당하는 테이블에 저장한다. 그리고 RuleChecker 와 Profiler 에서 추출된 성능에 대한 정보도 DB 에 입력한다.

- 3 단계: 품질지표 적용단계에서는 소프트웨어 구조정보를 통해서 각 모듈(클래스) 간의 결합도를 측정한다.

- 4 단계: 가시화 단계에서는 GenerateDotContents 프로그램을 사용하여 쿼리문으로 추출된 정보로 코드에 대한 아키텍처와 정량화된 품질지표 수치를 DOT 스크립트로 생성하고 graphViz 도구를 통해서 그래프로 그린다.

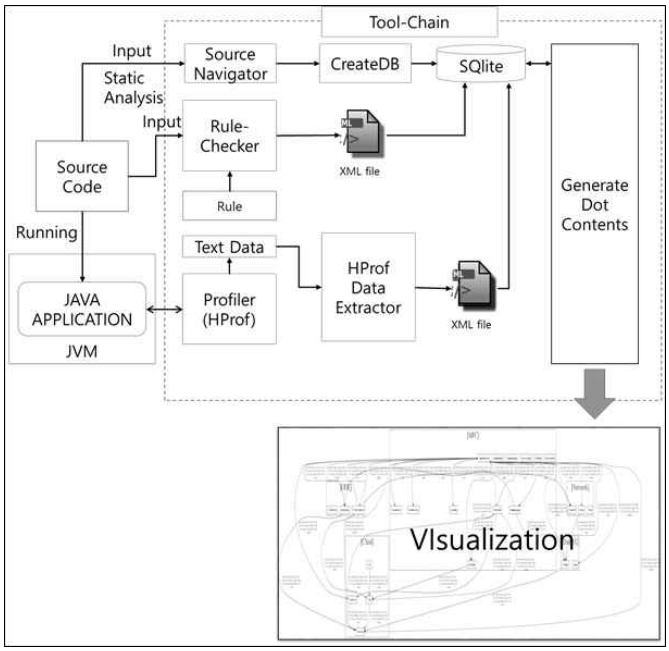


그림 1 소프트웨어 성능 가시화 도구 구성도

그림 2 는 소프트웨어 성능가시화도구에 결합도(자료, 스탬프, 제어, 외부, 공유, 내용)와 성능 저해요소(Loop Unrolling, Loop DownCounter, 반복문 내부의 불필요한 조건문, 다중 조건문)가 포함된 소스코드를 입력 후 구동한 가시화 이미지이다.

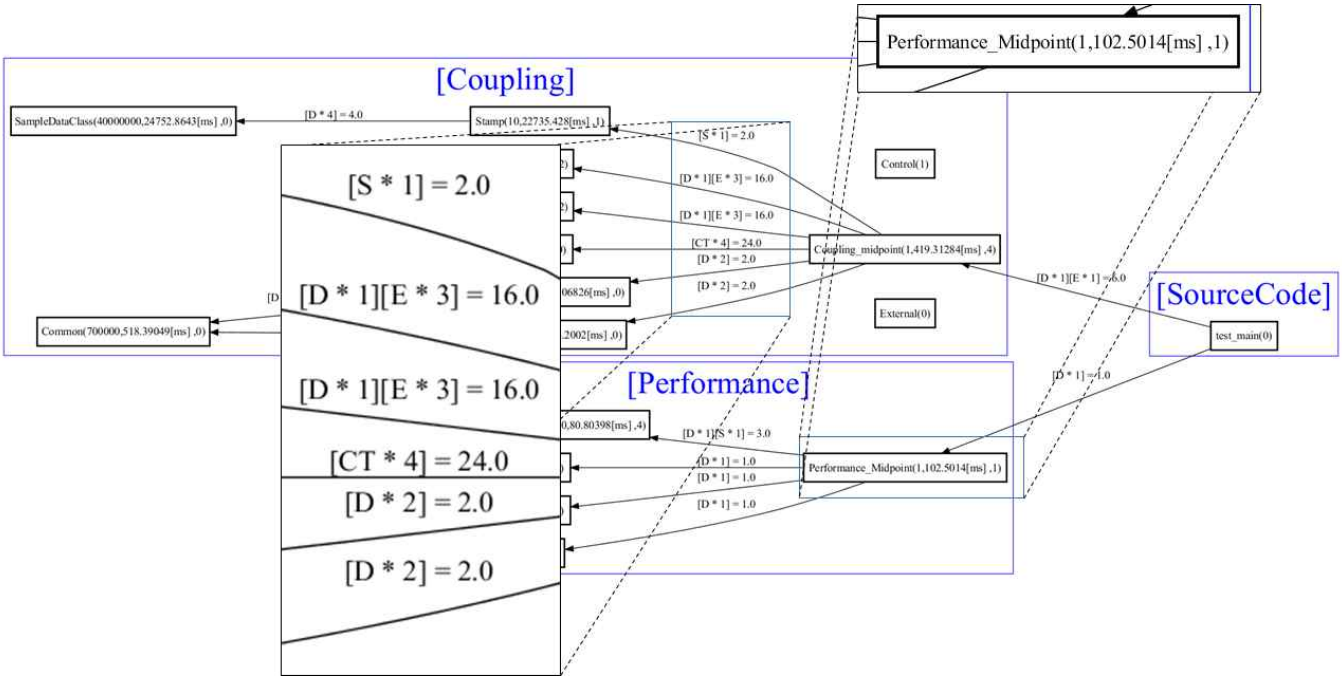


그림 2 추출된 소프트웨어 성능 가시화 이미지

그림 2 를 보면 네모로 표시되는 모듈에서는 클래스의 실제 호출 수, 클래스 전체의 수행시간, 성능저하요소 순으로 데이터를 표기한다. 그리고 모듈 간의 화살표에는 모듈 간의 결합도(D: 자료, S: 스탬프, C: 제어, E: 외부, CM: 공유, CT: 내용)의 개수와 결합도 총합점수를 표기한다. 이렇게 소프트웨어의 구조와 함께 성능지표, 결합도 점수를 같이 표현하여 쉽게 소프트웨어의 성능에 대한 평가가 가능하다.

4. 결 론

현재 우리나라의 거대해지는 소프트웨어 산업에 의해서 소프트웨어 고품질에 대한 이슈가 대두되고 있다. 하지만 그에 비해 단발성의 소프트웨어 개발, 시장 출하기간의 단축 등으로 인한 구현 중심의 개발 프로세스와 잦은 개발 인력의 이동으로 개발 문서의 부재, 잦은 패칭으로 소프트웨어의 품질저하의 악순환이 지속되고 있다. 그래서 본 논문에서는 소프트웨어의 고품질화를 위한 소프트웨어의 성능 측정 및 개선을 위한 역 공학 기반의 자동화 메커니즘 제안과 개발한다. 우리는 역공학과 정적 분석을 통해 소프트웨어의 재 사용성을 높이는 결합도와 소프트웨어성능에 위배되는 성능저하요소를 추출 했다. 그와 동시에 자바 성능측정도구(HProf)를 통한 동적 분석을 수행하여 정적 분석에서 얻을 수 없는 메소드의 수행속도, 실제 수행 횟수의 정보를 얻어 이전과는 다른 새로운 소프트웨어 가시화 했다. 이를 통해 실제 개발자뿐 아니라 많은 이해관계자들도 소프트웨어의 품질 지표와 구조를 쉽게 이해가 가능하다. 그리고 실제 소프트웨어가 실행 될 때 제일 많이 수행시간을 소모하는 부분을 보다 쉽게 찾는 것이다. 그래서 그 부분의 문제를 발견하고 리펙토링 한다면 성능적 품질의 개선이 가능하다.

현재 Java 언어에만 프로파일러가 적용되어, 향후 C, C++의 성능 가시화를 적용할 예정이다. 또한 다양한 언어의 의존된 성능문제를 찾아 패턴화 적용할 예정이다. 그리고 임베디드 시스템의 성능 가시화중 프로파일러를 통한 속도 측정뿐 아니라 실제 디바이스를 통한 사용전력을 측정하여 전력이 많이 소모되는 소스코드의 패턴을 찾아 가시화에 적용할 예정이다.

참고문헌

[1] Gartner “Gartner Market Databook, 2Q15 Update, 2015.06

[2] 강건희, 이근상, 김동호, 황준순, 김영수, 박용범, 김영철, “절차식 언어 기반의 코드 정적 분석을 위한 툴 체인 사례 연구,” 한국정보과학회 2014 한국 컴퓨터 종합학술대회 논문집, pp. 559-561, 2014

[3] 강건희, 이근상, 김동호, 황준순, 김영수, 박용범, 김영철, “SW 가시화 기반 리펙토링 기법 적용을 통한 정적 코드 복잡도 개선,” 2014 한국정보처리학회 추계학술대회, 제 21 권, 제 2 호, pp. 646-649, 2014

[4] 강건희, 이근상, 김영수, 박용범, 손현승,

김영철, “Open Source 기반 툴 체인화를 통한 코드 정적 분석 연구, 한국 정보과학회 컴퓨팅의 실제 논문지, 제 21 권, 제 2 호, pp. 148-153, 2015
[5] Henry H. Liu, “Software Performance and Scalability, A quantitative Approach,” Wiley, 2009
[6] Henry H. Liu, “Java Performance and Scalability, A quantitative Approach,” Wiley, 2013
[7] PMD, https://pmd.github.io/
[8] HPROF: A Heap/CPU Profiling Tool, https://docs.oracle.com/javase/7/docs/technotes/samples/hprof.html

PageRank 와 토픽 모델링을 이용한 소프트웨어 재사용 도메인 토픽 추출 시스템 개발*

이용석, 남영광, 황상원

연세대학교 전산학과
26493 강원도 원주시 연세대길 1
asapfast@icloud.com, yknam@yonsei.ac.kr, arsenal@yonsei.ac.kr

요약: 소프트웨어의 유지보수와 재사용에 있어서 소프트웨어의 도메인 토픽을 이해하는 것은 매우 중요하다. 소프트웨어가 발전함에 따라 소프트웨어의 크기는 더 방대해지고 있으며 이러한 소프트웨어의 도메인 토픽을 이해하는 것은 매우 어렵다.

최근에는 LSI(Latent Semantic Indexing), LDA(Latent Dirichlet Allocation[3] 등의 정보 검색 기술을 사용하여 소프트웨어의 특징, 기능 및 도메인 토픽을 추출하는 연구가 증가하고 있으며, 특히 소스 코드의 구조적 정보를 제외한 식별자 및 주석과 같은 비구조적 정보를 활용하여 LDA 를 적용하는 연구가 매우 활발하다..

본 논문에서는 구조적, 비구조적 정보 모두를 활용하며 자바로 작성된 소프트웨어를 대상으로 한다. 소스 코드의 구조적 정보와 비구조적 정보를 활용하여 소프트웨어로부터 문서 집합을 생성하고, 문서 집합에서 토픽을 추출하기 위해 LDA 알고리즘에 기초한 토픽 모델링을 사용함으로써 소프트웨어의 도메인 토픽을 추출하는 JTopic(Java Topic) 시스템을 개발 및 제안한다. JTopic 시스템은 문서 집합 생성 시 소스 코드 내의 식별자간의 호출 관계를 통해 그래프를 생성하고 PageRank 알고리즘[12]을 적용하여 식별자에 더 큰 중요도 값을 가지게 한다.

본 연구에서는 소스 코드 내 식별자의 PageRank 알고리즘에 의한 점수를 활용하여 기존 시스템들보다 정확한 토픽들을 제시함으로써 도메인 토픽을 추출하는 것을 목적으로 한다. JTopic 시스템은 오픈 소스 프로젝트를 대상으로 MALLET 토픽 모델링 시스템의 결과와 비교하여 평가하였다.

핵심어: LDA, 잠재 디리클레 할당, 토픽 모델링, 페이지 랭크, 소프트웨어 재사용

1. 연구배경 및 연구 목적

소프트웨어의 유지 보수 및 재사용을 위해 프로젝트에 대한 이해가 반드시 필요하지만, 프로젝트의 규모가 커지고 적용 시스템이 다양화 됨에 따라 구성 소프트웨어의 구조 역시 복잡해지고 있다.

소프트웨어에 대한 도메인 토픽은 해당 소프트웨어의 상위 수준의 기능적 특징을 의미하며 시스템을 이해하는데 중요한 정보다. 개발자가 모든 소스 코드를 직접 확인하거나 호출 그래프, 제어 흐름도, 데이터 흐름도와 같은 프로그램 분석 기술을 활용[14]하여 도메인 토픽을 추출하는 방법이 존재하지만 이러한 방법들은 매우 비효율적이며 결과 역시 좋지 않다. 반면, 소스코드의 주석과 식별자 등의 비구조적인 데이터의 분석은 프로젝트를 이해하는데 더 정확하고 효율적인 방법이다[5]. 최근에는 소스 코드의 비구조적인 데이터에 LSI 또는 LDA 알고리즘을 적용하여 프로젝트에 대한 여러 가지 토픽을 추출하는 기술들이 소개되었다[6, 9, 11, 15]. 이 방법들은 주로 프로젝트에 대한 여러 개의 단어 집합(하나의 단어 집합은 하나의 토픽을 의미)과 각 단어 집합에 프로젝트가 포함될 가능성(또는 토픽 분포)를 출력해줌으로써 사용자가 각 토픽들을 명명 하여 도메인 토픽을 추측하도록 한다. 하지만, 토픽 분포 값은 매우 작고 각 토픽들의 연관성이 크지 않기 때문에 사용자가 프로젝트의 도메인 토픽을 추측하는 것이 어렵다.

소프트웨어 내의 각 소스 코드는 하나의 문서라고 할 수 있고 소스 코드들로 구성된 하나의 소프트웨어는 문서 집합이라고 할 수 있으며, 문서 집합에는 토픽 모델링(Topic Modeling)을 적용함으로써 개발자가 소프트웨어의 도메인 토픽을 파악하는데 도움이 되는 결과를 내어줄 수 있다. 한편, 소스 코드의 클래스로부터 추출되는 단어는 다른 단어들에 비해 토

* 이 논문은 2014 년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보• 컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No.NRF-2014M3C4A7030505)

픽으로서의 의미가 더 크다.

위 사실에 기초하여, 본 논문에서 제안하는 JTopic 시스템은 PageRank 알고리즘을 통해 각 식별자에 중요도 값을 할당하여 분석 대상 문서를 생성하는데 활용한다. 그리고 생성된 문서에 LDA 기반 토픽 모델링을 적용하여 시스템의 결과가 도메인 토픽을 추출하기에 용이하도록 하는 것을 목적으로 한다.

2. 관련 연구 및 배경 지식

이번 장에서는 소프트웨어로부터 토픽을 추출하기 위해 진행되었던 연구를 소개하고, 본 논문에서 제안하는 방법에 요구되는 기술들에 대해 기술한다.

2.1 상위- 수준 개념 추출 연구[8]

소스 코드의 비구조적인 정보만을 분석하고 LDA를 적용함으로써 토픽을 추출하는 연구로 시스템 구조는 그림 1 과 같다. 시스템은 프로젝트를 입력으로 받아 전처리 과정을 통하여 문서 집합을 생성한다. 전처리 과정은 우선 키워드 필터링 과정을 통하여 식별자와 주석을 추출한다. 그리고 추출된 키워드에서 프로그래밍 언어 형태로 구성된 단어들을 선별하여 Camel case, Underscored 된 단어 분리 및 불용어 제거 후 최종 분석 대상 단어집을 생성한다.

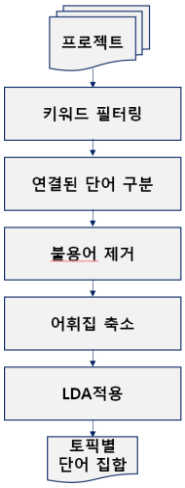


그림 1 상위-수준 개념 추출 시스템 구조

그림 2 는 전처리를 통해 소스 코드로부터 비구조적 정보를 추출한 결과를 나타낸다. 전처리 과정이 끝나면 단어집에 LDA 를 적용하여 토픽별 단어 그룹을 출력한다.

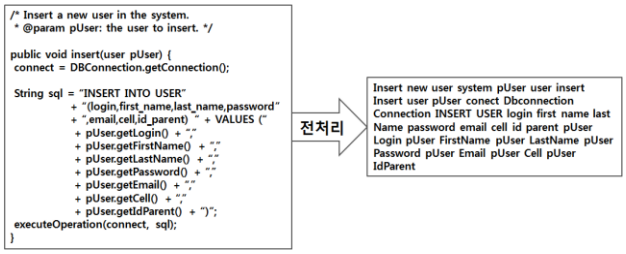


그림 2 전처리를 통한 비구조적 정보 추출

표 1 은 UML 기반 모델링 도구로써 UML 다이어그램 작성을 지원하는 프로젝트인 ArgoUML 을 대상으로 상위-수준 개념 추출 시스템의 결과를 나타낸다. 사용자는 추출된 단어 집합을 보고 수동적으로 토픽을 명명함으로써 토픽을 얻는다.

해당 연구는 비구조적 정보와 LDA 를 활용하여 소프트웨어의 도메인 토픽을 추출할 수 있는 가능성을 확인할 수 있는 연구 사례다.

표 1 상위-수준 개념 추출 시스템 결과

대상 시스템	명명된 토픽	추출 단어
ArgoUML	Figure	g, diagram, setting, node, edge, width, bound, param
	Files	le, project, name, pro le, label, name, panel, list
	Events	event, action, target, model, item, namespace, element, handle

2.2 토픽 마이닝[5]

LDA 를 적용하여 소스 코드로부터 토픽들을 마이닝하는 시스템을 제안한다. 그림 3 는 토픽 마이닝 시스템 구조를 나타낸다.

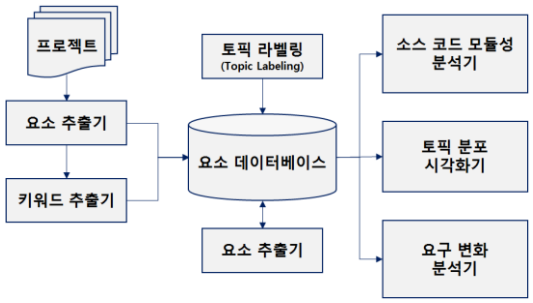


그림 3 토픽 마이닝 시스템 구조

토픽 마이닝 시스템은 먼저, 소스 코드를 파싱하고 파일, 메서드, 메서드 종속성, 데이터 구조 등의 메타

데이터를 추출한다. 이어서 식별자들을 의미 있는 단어들로 구분하여 스테밍 과정을 거친 후 불용어를 제거하는 과정을 거친다. 그리고 추출된 정보들을 활용하여 문서-단어 매트릭스를 구성하여 LDA 를 적용함으로써 토픽 모델링을 통해 출력되는 결과에 사용자가 직접 명명 한다. 표 2 는 토픽 마이닝 시스템의 결과를 나타낸다.

이 연구에서는 문서-단어 매트릭스를 구성할 때, 클래스, 메서드, 속성 수준에 따라서 해당 수준으로 출현한 단어의 횟수에 각각 2, 1, 1 의 특정 값을 곱해 줌으로써 셀 값을 할당한다. 예를 들어, 그림 4 에서 단어 'order'에 대한 문서-단어 매트릭스의 셀 값은 7 이 된다.

해당 연구는 비구조적 정보뿐만 아니라 구조적 정보를 통해 일반적인 단어에 비해 상대적으로 큰 의미를 가질 가능성이 있는 클래스 이름 등에 추가적인 값을 할당해 줌으로써 소스 코드에 대한 토픽 모델링의 결과를 향상 시켰다.

표 2 Apache 소스 코드의 토픽 추출

키워드	확률
ssl	0.373722
expr	0.042501
init	0.033207
engine	0.026447
var	0.022222
ctx	0.023067
ptemp	0.017153
mctx	0.013773
lookup	0.011238
Modssl	0.011238
ca	0.009548
명명 결과	
SSL	

키워드	확률
log	0.141733
request	0.036017
mod	0.031100
config	0.029871
name	0.023725
headers	0.021266
autoindex	0.020037
format	0.017578
cmd	0.015120
header	0.013891
add	0.012661
명명 결과	
Logging	

```
public class OrderDetails implements java.io.Serializable {
    private String orderId;
    private String userId;
    private String orderDate;
    private float orderValue;
    private String orderStatus;

    public String getOrderStatus() {
        return(orderStatus);
    }
    ...
    ...
}
```

1*2 + 4*1 + 1*1 = 7

그림 4 문서-단어 매트릭스의 셀 값 할당

2.3 LDA(Latent Dirichlet Allocation)

LDA 는 Blei & A. Ng & M.Jordan[3]에 의해 제안된 알고리즘이다. LDA 는 자연어로 구성된 텍스트 문서 집합으로부터 생성 확률 모델(Generative Probabilistic Model)을 통해 확률 토픽 모델을 유도하는 알고리즘으로 각 문서에 어떤 토픽들이 존재하는지에 대한 확률 모델이다.

LDA 에서의 문서는 여러 개의 토픽을 포함하고 있으며, 문서 내에 등장하는 단어의 순서에 상관하지 않고 단어의 출현 횟수만을 고려한다. 토픽별 단어 수의 분포를 기반으로 각 문서에서 출현하는 단어 수의 분포를 분석함으로써 해당 문서가 어떤 토픽들을 다루고 있을지를 예측하는 것이 LDA 의 목적이다. 그림 5 는 LDA 적용 결과를 나타낸다. 아래 부분은 특정 문서이고 위에 부분은 토픽별 단어 집합이며 토픽은 개발자가 직접 명명 한다.

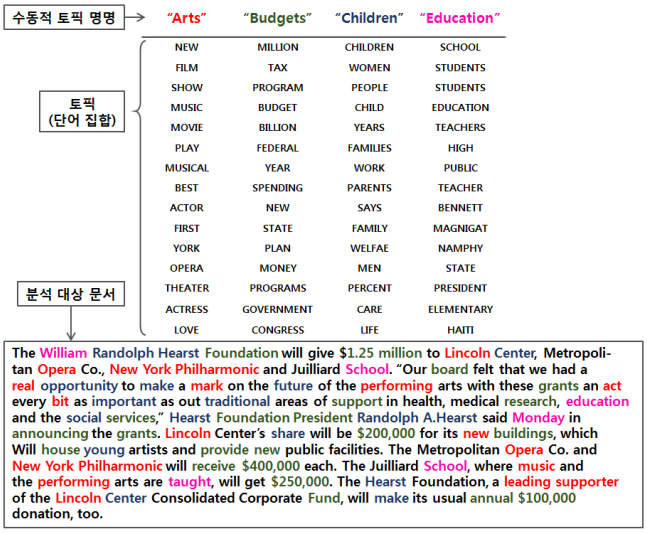


그림 5 LDA 적용 결과

LDA 에서 사용되는 모델은 다음과 같다.

M 개의 문서로 구성된 코퍼스(D)가 주어졌을 때, 각 문서(d)는 k 개의 주제(z)에 속한다. 각 문서는 구성하고 있는 N 개의 단어(w)로 나타내며, $d=(w_1,w_2,w_3,\ldots,w_N)$ 으로 표기한다. 코퍼스는 $D=\{d_1,d_2,\ldots,d_M\}$ 으로 표기한다. 단어집은 코퍼스를 구성하는 모든 단어로 구성되며 단어 수만큼의 크기를 갖는다. 단어집의 크기를 V 라 하면, 각 단어는 인덱스 $v\in\{1,\ldots,V\}$로 대응된다. LDA 는 이러한 모델로부터 다음과 같은 생성 과정을 따른다.

1. $N-Posion(\xi)$ 선택
2. $\theta-Dirichlet(\alpha)$ 선택
3. 각 문서내의 단어 에 대하여:
   (a) $z_n-Multinomial(\theta)$ 선택
   (b) z_n 이 주어졌을 때 $p(w_n\mid z_n,\beta)$ 로부터 w_n 선택

α 는 k 차원 디리클레 분포의 매개변수이고 θ 는

k 차원 벡터이며, θ_i 는 해당 문서가 i 번째 주제에 속할 확률 분포를 나타낸다. z_n 은 k 차원 벡터이고 z_n^i 는 단어 w_n 이 i 번째 주제에 속할 확률 분포를 나타낸다. β 는 $k \times V$ 크기의 행렬 매개변수로 β_{ij} 는 i 번째 주제가 단어집의 j 번째 단어를 생성할 확률을 나타낸다. LDA 의 매개변수를 추정하기 위한 방법으로는 변분 추론(Variational Inference) 방법과 깁스 샘플링(Gibbs Sampling)방법이 있다. 본 연구에서는 깁스 샘플링 방법을 사용한다. 추가로, LDA 는 다양한 분야에서 활용될 수 있으며, 소프트웨어 공학 측면에서는 소프트웨어의 분류 연구[8], 소프트웨어 추적성(Traceability) 연구[7], 버그 처리(localization) 연구[13], 통계적 디버깅 연구[2] 등에 사용되고 있다.

2.4 PageRank

PageRank 는 Sergey Brin & Lawrence Page[12]에 의해 제안된 알고리즘으로 월드 와이드 웹(World Wide Web)과 같은 링크 구조를 가진 문서 집합에 대하여, 각각의 문서에 가중치를 부여하여 중요도에 따라 문서들의 순위를 매기는 알고리즘이다.

그림 6 은 웹 페이지 그래프에 PageRank 알고리즘을 적용한 결과를 나타낸다.

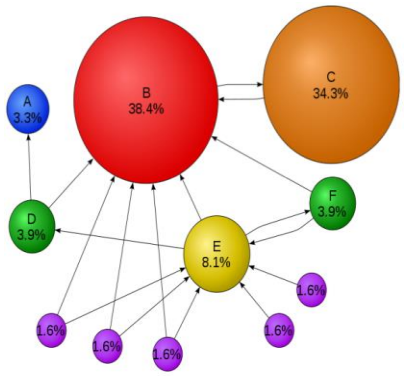


그림 6 PageRank 및 관계 도식화

각 노드(Nodes)는 하나의 웹 페이지에 대응되고 간선(Edges)은 하나의 웹페이지가 다른 웹페이지에 대한 링크를 포함한다는 것을 의미한다. PageRank 알고리즘에서는 노드를 향하는 링크 수를 정규화(Normalize)하는 방식을 사용하며 아래와 같은 수식을 사용한다.

$$PR(A) = (1-d)/N + d(PR(T_1)/C(T_1) + \cdots + PR(T_n)/C(T_n))$$

PR 은 PageRank 점수를 의미하고, PR(A)는 웹 페이지 'A'의 PageRank 점수를 의미한다. 'Damping Factor'라 부르는 d 는 0 과 1 의 사이의 값으로 정해지며, 사용자가 해당 페이지가 가진 링크를 통해 다른 페이지로 이동할 확률을 의미한다. d 의 값은 실험을 통해 정해지는데 일반적으로 0.85 로 설정한다. N 은 전체 페이지 수를 의미한다. T_n 은 웹 페이지 'A'를 가리키는 페이지를 의미하며 $C(T_n)$ 은 T_n 이 가지고 있는 링크의 수를 의미한다.

즉, 특정 페이지의 PageRank 점수는 해당 페이지를 인용 또는 참조하고 있는 다른 페이지들의 PageRank 점수를 정규화한 값의 합이 되며, 각 페이지는 PageRank 점수를 할당 받게 된다.

3. JTopic 시스템 구현

이번 장에서는 본 논문에서 제안하는 JTopic 시스템의 구조와 모듈을 설명한다.

그림 7 은 JTopic 시스템의 매개변수 설정부분을 나타내며 그림 8 은 JTopic 시스템의 구조를 나타낸다. 시스템의 최초 입력은 모델링 대상 소프트웨어의 소스 코드 및 jar 파일이고 사용자 정의 토픽 수, 하나의 토픽에 대해 출력할 단어 수, 그리고 깁스 샘플링 횟수를 설정할 수 있다.

그림 7 JTopic 시스템의 매개변수 설정

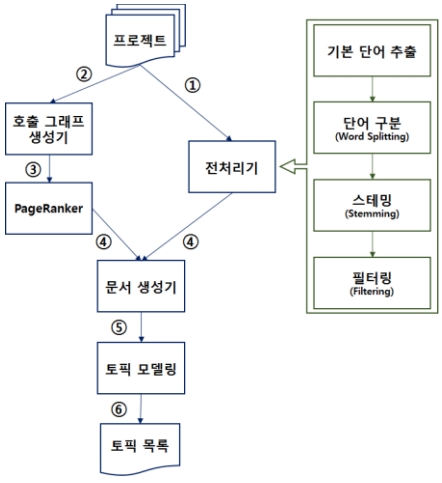


그림 8 JTopic 시스템 구조

JTopic 시스템은 먼저, 전처리를 통해 프로젝트의 소스 코드를 분석하여 소스 코드로부터 의미 있는 단어 집합을 추출하는 과정 및 jar 파일로부터 식별자간 동적 호출 관계를 추출하여 호출 그래프를 생성하는 과정을 거친다. 호출 그래프가 생성되면

PageRank 알고리즘을 호출 그래프에 적용함으로써 각 식별자에 PageRank 점수를 할당한다. 다음으로, 전처리에서 추출된 단어 집합에 PageRank 점수를 활용하여 최종 분석 대상 문서를 생성한다. 마지막으로, 생성된 문서에 LDA 기반 토픽 모델링을 적용하여 사용자에게 토픽별 단어 집합을 보여주고 토픽을 구성함으로써 프로젝트의 도메인 토픽을 얻을 수 있도록 한다. LDA 기반 토픽 모델링 적용하기 위하여 MALLET Topic Modeling Toolkit [1] 을 사용하였다. MALLET Topic Modeling Toolkit 은 명명 되지 않은 텍스트 집합을 분석하는데 사용되는 도구로써 토픽 모델링을 위해 LDA, Pachinko Allocation[17] 및 HLDA(Hierarchical LDA)[4]를 구현해 놓았으며, 본 연구에서는 MALLET 의 LDA 기반 토픽 모델링 방법을 사용하였다.

다음은 각 단계에 대한 설명이다.

3.1 전처리기

LDA 기반 토픽 모델링을 적용하기 위해서는 분석 대상 문서 집합이 요구된다. 프로젝트 내의 소스 코드 파일들은 각각이 하나의 문서로 대응될 수 있으며 프로젝트는 문서 집합으로 대응될 수 있다. 하지만 소스 코드 파일은 일반적인 문서와는 다르게 구조적으로 구성된 문서이기 때문에 그 자체를 토픽 모델링의 대상으로 사용하기에 적합하지 않다. 소스 코드가 자체적으로 분석 대상 문서로써 사용되기에 부적합한 이유는 다음과 같은 특징 때문이다.

- 프로그래밍 언어의 키워드 및 예약어 : 토픽과 관련 없는 프로그래밍 언어의 예약어 및 키워드 public, private, for, if, else 등이 존재한다.
- 주석 내 예약어 : 주석이 JavaDoc 과 같은 형식으로 작성되었을 경우, 토픽과 거의 관련 없는 단어가 등장한다. JavaDoc 에서 지정된 예약어인 param, return, author, exception, see, serial 등을 예로들 수 있다.
- 합성어 : 주석과 식별자에는 getDrawingView, getConnectedTextTool 와 같은 합성어가 존재한다.
- 불용어 : 관사, 전치사, 조사 등 단어로써 의미가 없는 단어가 존재한다.

따라서, 소스 코드를 토픽 모델링의 대상으로 사용하기 위해서는 위와 같은 특징을 처리하여 단어 집합을 생성하며, 문서 생성기는 생성된 단어 집합을 활용하여 문서를 생성한다.

3.1.1 기본 단어 추출

기본 단어 추출 단계는 소스 코드를 파싱하여 토픽과 관련 없는 프로그래밍 언어의 예약어 및 키워드를 제외한 단어들 즉, 시스템에서 사용될 문서를 구성하기 위해 필요한 모든 주석 및 식별자를 추출하는 단계이다. 그림 9 는 프로젝트 JHotDraw 의 소스 코드 중 하나인 'PertFigureCreationTool.java' 에 기본 단어 추출 단계를 적용한 결과를 나타낸다. JTopic 시스템에서는 소스 코드를 파싱하기 위해 버전 1.8 의 자바 파서(JavaParser-1.8)[2] 를 사용하였다.

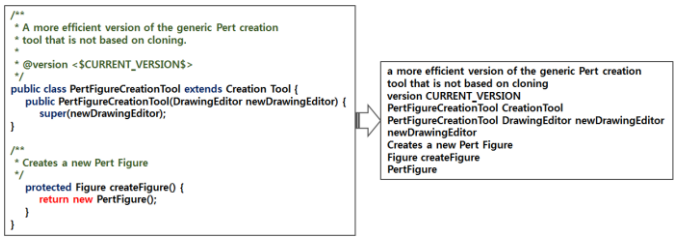


그림 9 기본 단어 추출 결과

3.1.2 단어 구분

소스 코드에서 사용되는 식별자 및 주석 내 단어는 Camel Case, Underscore 등의 프로그래밍 언어의 명명 규칙에 따라 작성되기 때문에 많은 합성어가 존재한다. 이러한 합성어는 한 개 이상의 의미 있는 단어들로 이루어지므로 단어별로 분리될 필요가 있으며, 합성어 자체로도 중요한 의미를 나타낼 수 있으므로 원본 단어 역시 추출된 단어 그대로 존재하도록 한다. 명명 규칙에 따른 단어 구분 예는 아래와 같으며, 그림 10 은 기본 단어 추출 단계의 결과(그림 9)에 단어 구분 단계를 적용한 결과를 나타낸다.

- Camel Case : 클래스 명이 'DrawApplication' 일 경우, 이를 세 개의 단어 'Draw', 'Application', 'DrawApplication'으로 구분한다.
- Underscore : 메서드 명이 'draw_sample_application'일 경우, 이를 '_' 문자로 구분하여 'draw', 'sample', 'application', 'draw_sample_application'으로 구분한다.

[1] http://www.mallet.cs.umass.edu/topics.php

[2] http://mvnrepository.com/artifact/com.google.code.javaparser/javaparser/1.0.8

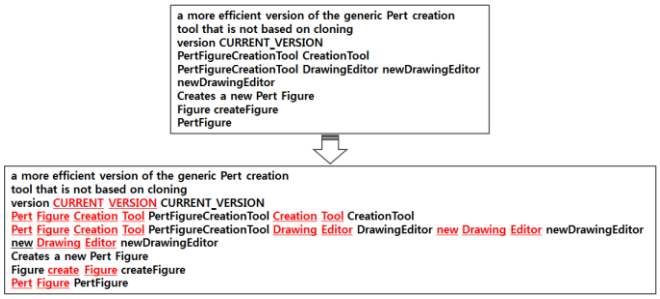


그림 10 단어 구분 결과

3.1.3 스테밍(Stemming)

일반적인 문서뿐만 아니라 소스 코드 내의 단어들은 여러 가지 형태로 사용된다. 따라서 단어를 스테밍하여 하나의 어근으로 표현되도록 한다. 예를 들면, 'Draw', 'Draws', 'Drawing'란 단어는 'Draw'라는 하나의 어근으로 표현된다.

본 시스템의 스테밍 단계에서는 Porter 스테밍 알고리즘(Porter's Stemming Algorithm)[10]을 사용하였으며 원본 단어를 포함하는 어근-단어 사전을 생성하여 어근과 원본 단어 모두가 출력이 가능하도록 하였다. 그림 11 은 단어 구분 결과(그림 10)에 스테밍을 적용한 결과를 나타내고 표 3 은 생성된 어근-단어 사전의 일부를 나타낸다.

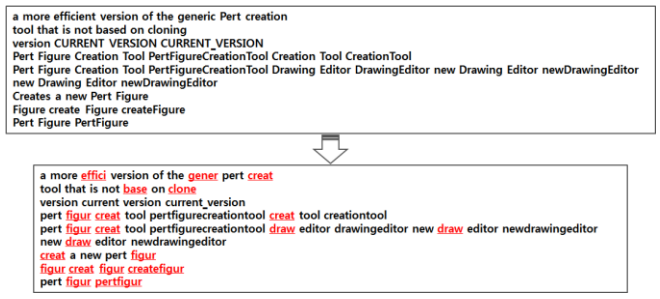


그림 11 스테밍 결과

표 3 스테밍 단계의 어근-단어 사전 생성

어근	원본 단어
a	a
more	more
effici	efficient
version	version
of	of
the	the
gener	generic
pert	pert
creat	creation, creates
clone	cloning
drawingeditor	drawingeditor
new	new
newdrawingeditor	newdrawingeditor
pertfigur	pertfigure

3.1.4. 필터링(Filtering)

필터링 단계에서는 불용어를 제거한다. 이전 단계들에 의해 얻어진 모든 단어가 토픽으로 추천될 가능성이 있는 것이 아니다. 텍스트 문서와 주석 내에서 흔히 사용되는 관사, 조사, 전치사 등 단어로서 의미가 없는 단어와 소스 코드의 식별자에서 흔히 사용되는 'get', 'set' 등의 상대적 중요성이 매우 작은 단어를 단어 집합과 어근-단어 사전에서 제거 한다. MALLET 에서 사용하는 불용어 목록에 본 시스템에서 정의한 단어들을 추가하여 필터링을 적용하였으며, 그림 12 와 표 4 는 스테밍 단계의 결과(그림 12, 표 4 에 필터링을 적용한 결과를 나타낸다.

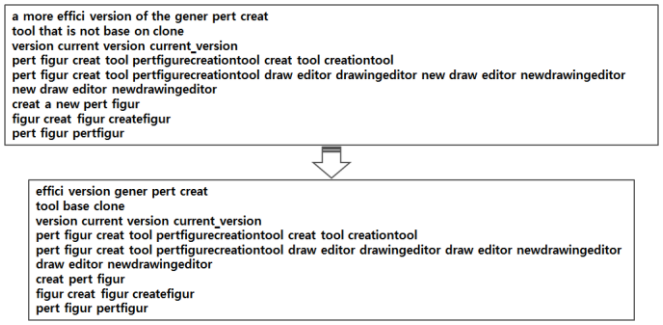


그림 12 필터링 결과

표 4 필터링 단계의 어근-단어 사전 변화

어근	원본 단어
~~a~~	~~a~~
~~more~~	~~more~~
effici	efficient
version	version
~~of~~	~~of~~
~~the~~	~~the~~
gener	generic
pert	pert
creat	creation, creates
clone	cloning
drawingeditor	drawingeditor
~~new~~	~~new~~
newdrawingeditor	newdrawingeditor
pertfigur	pertfigure

3.2 호출 그래프 생성기

자바 프로젝트는 많은 수의 클래스로 구성되어있으며 클래스는 생성자 또는 클래스가 포함하고 있는 메서드를 통해 다른 클래스들과 연결된다. 프로젝트를 웹드 와이드 웹에 대응시킨다면, 클래스는 웹 페이지로 대응되고 메서드 호출 관계는 웹 페이지의 링크로서 대응시킬 수 있다.

따라서, 프로젝트에 PageRank 알고리즘을 적용할 수 있으며 호출 그래프 생성기는 PageRank 알고리즘을 적용하기 위해서 프로젝트의 호출 그래프를 생성한다.

호출 그래프 생성기에서는 먼저, 표 5 와 같이 프로젝트로부터 호출, 피호출 관계에 있는 클래스 및 호출 횟수를 추출한다. 그리고 클래스를 노드(node)로 하고 클래스 내에 포함된 메서드 호출 관계를 간선(edge)로 하는 그래프를 생성한다. 간선은 방향과 호출 횟수를 속성으로 갖는다. 그림 13 는 JHotDraw 프로젝트를 대상으로 그래프를 생성한 결과의 예를 나타낸다. 클래스 PertFigureCreationTool 은 세 개의 클래스 PertApplet, PertApplication, PertFigureCreation 으로부터 각각 2, 2, 1 회 호출되고 있다는 사실을 확인할 수 있다.

호출 그래프 생성기를 통해 만들어진 그래프는 PageRank 알고리즘의 입력으로서 사용된다.

표 5 추출된 호출, 피호출 클래스 및 호출 횟수

피호출자(클래스)	호출자(클래스)	호출 횟수 (회)
FollowURLTool	JavaDrawViwer	2
	FollowURLTool	2
PertFigureCreationTool	PertApplet	2
	PertApplication	2
	PertFigureCreationTool	2
CustomSelectionTool	CutomSelectionTool	1

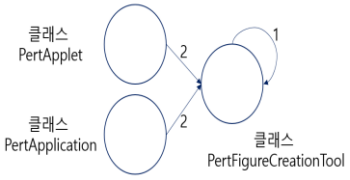


그림 13 PertFigureCreationTool 클래스의 호출 그래프 예

3.3 PageRanker

앞서 생성된 호출 그래프를 대상으로 PageRank 알고리즘을 적용한다. 그림 14 는 그래프에 PageRank 알고리즘을 적용한 결과를 나타낸다. 각 노드는 PageRank 점수를 속성으로 가지며 점수가 높을수록 상대적 중요성이 크다는 것을 의미한다. PageRank 의 'Damping Factor' d 의 값은 0.85 로 사용하였다.

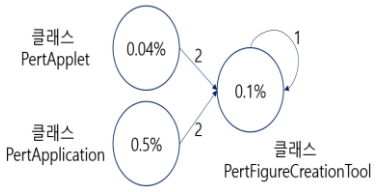


그림 14 PageRank 알고리즘이 적용된 그래프

3.4 문서 생성기

전처리기에서 추출된 단어 집합과 호출 그래프에 PageRank 알고리즘을 적용함으로써 할당된 클래스별 PageRank 점수를 활용하여 문서를 생성한다.

모든 클래스 식별자의 PageRank 점수는 1 보다 작은 매우 작은 값을 가진다. 본 연구에서는 각 PageRank 점수에 특정 값을 곱함으로써 점수가 0 이상의 정수인 변형된 PageRank 점수를 사용했다. 해당 클래스 식별자가 변형된 PageRank 점수의 수만큼 각 문서에 추가되도록 하였다. 변형된 PageRank 점수를 만들기 위해서 사용되는 값은 실험을 통해서 정하였으며 본 연구에서는 특정 값으로써 200 을 사용하였다.

3.5 토픽 모델링

토픽 모델링은 문서 집합의 각 문서에 대해 임의의 토픽들에 대한 분포 값과 함께 단어 집합을 찾아낸다.

문서 생성기를 통해 생성된 문서 집합을 대상으로 토픽 모델링을 한다.

LDA 기반 토픽 모델링은 기본적으로 각 문서가 k 개의 토픽 중에 하나 이상을 다룬다고 가정하고 있으며, 모델링의 결과물인 토픽이란 임의의 토픽에 대한 단어 집합이다. 출력되는 단어 집합들을 명명 하는 기술은 현재로서 존재하지 않는다. 결국, 사용자가 토픽 모델링의 결과를 확인하여 토픽 이름을 수동적으로 구성해야 한다. 사용자는 구성된 토픽을 통해 프로젝트의 도메인 토픽을 파악할 수 있다.

토픽의 수(k)는 모델링이 수행되기 전에 사용자로부터 요구되는 매개변수이다. 토픽의 수를 자동으로 결정하는 알고리즘 역시 현재로서는 존재하지 않는다. 따라서 여러 개의 k 값을 사용하여 가장 큰 우도(likelihood)를 갖는 경우의 k 값을 선택한다. JTopic 시스템에서는 프로젝트를 대상으로 하는 실험을 통해 k 의 값을 10~30 으로 하였다.

4. 실험 및 평가

JTopic 시스템은 자바로 구현하였으며 시스템의 GUI 는 그림 15 와 같다. 사용자는 분석 대상 프로젝트의 소스 코드 파일과 jar 파일을 입력하고, 시스템으로부터 출력될 토픽 수와 토픽별 단어 수 그리고 샘플링 반복 횟수를 설정 후 시스템을 실행한다. 그림 16 은 토픽 수=20, 토픽별 단어 수=20, 샘플링 반복 횟수=1000 으로 설정한 상태에서 프로젝트 JHotDraw 를 분석한 결과 화면이다. JTopic 시스템은 입력된 프로젝트의 토픽에 대한 단어 집합과 함께 토픽 분포 값을 출력하며 사용자는 각 단어 집합을 확인하여 도메인 토픽을 결정한다.

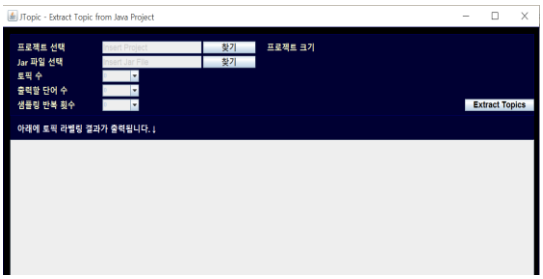
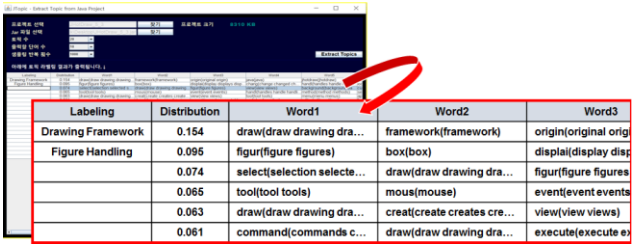


그림 15 JTopic 시스템 GUI



Labeling	Distribution	Word1	Word2	Word3
Drawing Framework	0.154	draw(draw drawing dra…	framework(framework)	origin(original orig
Figure Handling	0.095	figur(figure figures)	box(box)	displai(display disp
	0.074	select(selection selecte…	draw(draw drawing dra…	figur(figure figures
	0.065	tool(tool tools)	mous(mouse)	event(event events
	0.063	draw(draw drawing dra…	creat(create creates cre…	view(view views)
	0.061	command(commands c…	draw(draw drawing dra…	execute(execute ex

그림 16 JTopic 시스템 결과

토픽 분포 값이란 하나의 문서가 k 개의 토픽(T)에 포함된다고 할 때, 문서가 각 토픽(t_i)에 포함될 확률 또는 각 토픽에 대한 분포 값을 의미한다. 토픽 분포 값이 작으면, 해당 토픽은 문서가 가진 의미 있는 토픽이 될 가능성이 적다고 할 수 있으며, 반대로 토픽 분포 값이 크면, 해당 토픽은 의미 있는 토픽이 될 가능성이 크다고 할 수 있다. 해당 시스템은 이러한 분포를 기반으로 단어들의 집합을 추출하였고 토픽 결정의 신뢰도를 높였다.

본 논문에서는 의미 있는 토픽이 될 가능성이 높은 토픽을 추출하기 위해 토픽 분포 값에 대한 기준이 되는 임계값을 설정하였다. 실제 대상 프로젝트의 도메인 토픽을 수동적으로 추출하고 MALLET 으로 자바 오픈 소스 프로젝트들을 다양하게 실험해 본 결과, MALLET 을 통해 생성된 도메인 토픽이 실제 대상 프로젝트의 도메인 토픽이 될 수 있는 경우의 토픽 분포 최소값이 0.15 로 나타났다. JTopic 시스템과 MALLET 의 비교하기 위해 이 값을 임계값으로 결정했다.

시스템의 평가는 다음 세 가지 항목으로 구성하였으며 첫 번째 항목의 조건을 만족하지 못할 경우 나머지 두 개의 항목은 평가하지 않는다.

항목 ① : 추출된 토픽이 임계값 이상의 토픽 분포 값을 갖는가?

- 토픽 분포가 임계값 보다 작다는 것은 대상 프로젝트의 도메인 토픽이 될 가능성이 매우 적다는 것을 의미한다.

항목 ② : 도메인 토픽을 구성할 때 추출된 토픽의 단어를 사용하는 비중

- 시스템의 결과를 활용하는 비중이 클수록 시스템의 효율이 높고 및 사용이 용이하다.

항목 ③ : 구성된 도메인 토픽이 실제 대상 프로젝트의 도메인 토픽을 대표 할 수 있는가?

- 구성한 도메인 토픽이 실제 대상을 대표할 수 없다면 도메인 토픽으로서의 의미가 없다.

본 논문에서는 오픈 소스 프로젝트 JHotDraw [3], JEdit[4], CheckStyle[5] 세 가지를 대상으로 사용하였다. JHotDraw 는 기술적이고 구조화된 그래픽을 모델링하는데 사용되는 자바 GUI 프레임워크로써 여러 가지 메뉴의 선택을 통해 그림을 그릴 수 있는 다양한 그리기 관련 기능이 포함된 프로젝트다. JEdit 는 프로그래머들이 소스코드를 편하게 작성할 수 있도록 만든 자바로 작성된 텍스트 편집기다. CheckStyle 은 프로그래머가 자바 표준에 맞는 코드를 작성하도록 지원하는 개발 도구다.

위 3 개의 프로젝트를 대상으로 하여 MALLET 토픽 모델링과 JTopic 시스템을 사용하여 실험을 진행하였고, 공통적인 실험 조건으로 토픽 수, 출력 단어 수, 샘플링 횟수를 각각 20, 15, 1000 으로 설정하였다.

시스템의 결과로 출력되는 단어 집합들을 통해 도메인 토픽을 구성하고 토픽 분포를 비교함으로써 시스템을 평가하였다.

먼저, JHotDraw 를 대상으로 한 실험에서, JTopic 은 임계값 이상의 토픽 분포 값을 갖는 토픽이 출력된 반면 MALLET 은 임계값 이상의 토픽 분포 값을 갖는 토픽을 출력하지 못하였다. 표 6 은 JTopic 에서의 출력 결과 중 하나의 토픽을 단어 순위에 따라 나타낸다. 사용자는 출력된 단어 집합을 확인하여 연관된 단어들을 하나 이상의 단어로 표현하고 이를 활용하여 토픽을 명명하도록 한다. 예를 들어, 표 6 에서, 단어 'handle', 'change', 'select', 'connector', 'locate'를 'handling'이라는 단어로 표현하고 단어 'draw, line,

[3] http://www.jhotdraw.org/

[4] http://www.jedit.org/

[5] http://CheckStyle.sourceforge.net/

decorate, palette, polygon, layout'을 'draw 또는 drawing'과 같은 단어로 표현할 수 있다. 그리고 나머지 단어들과 함께 활용하여 'Drawing Handling Tool'과 같이 토픽을 명명할 수 있다. 실제로 이와 같이 명명된 토픽은 JHotDraw 의 도메인 토픽을 대표할 수 있다.

표 6 JTopic 시스템 결과(JHotDraw)

명명된 토픽	출력 단어	분포 값
Drawing Handling Tool	handle, draw, tool, change, select, connector, locate, line, text, strategy, standard, decorate, palette, polygon, layout	0.6136

JEdit 을 대상으로 한 실험에서는 JTopic, MALLET 모두 임계값 이상의 토픽 분포 값을 갖는 토픽을 출력하였으며 표 7 과 표 8 은 각각 JTopic, MALLET 의 출력 결과 중 하나의 토픽을 단어 순위에 따라 나타낸다. JTopic 의 결과에서는 추출된 단어들의 대부분을 활용함으로써 토픽을 명명할 수 있고 명명된 토픽이 JEdit 의 도메인 토픽을 대표 할 수 있다. 반면, MALLET 은 출력된 단어간의 연관성이나 의미 있는 단어들의 수가 매우 적기 때문에 소수의 단어만을 활용하여 토픽을 명명하며 명명된 토픽은 JEdit 의 도메인 토픽이라 할 수 없다.

표 7 JTopic 시스템 결과(JEdit)

명명된 토픽	출력 단어	분포 값
Text/File Editor	bsh, edit, buffer, manager, handler, option, pane, text, list, dialog, file, vfs, class, factory, provide	0.3126

표 8 MALLET 결과(JEdit)

명명된 토픽	출력 단어	분포 값
Free Software	program, software, free, generate, fold, warranty, version, foundation, size, tab, false, explicit, jedit, java, indent	0.2314

CheckStyle 을 대상으로 한 실험에서는 JTopic 에서만 임계값 이상의 토픽 분포 값을 갖는 토픽이 출력되었으며 표 9 는 출력 결과 중 하나의 토픽을 나타낸다. 명명된 토픽은 'Handler to Check Java code'이며, 이는 CheckStyle 의 도메인 토픽을 대표할 수 있다.

표 9 JTopic 시스템 결과(CheckStyle)

명명된 토픽	출력 단어	분포 값
Handler to Check Java code	check, handler, abstract, javadoc, option, token, type, util, tree, line, class, import, stream, tool, draw	0.1978

표 10 은 위 3 개의 프로젝트를 대상으로 한 JTopic 시스템과 MALLET 의 평가 결과를 나타낸다. MALLET 은 프로젝트 JEdit 에 대해서만 임계값 이상의 토픽 분포 값을 갖는 토픽을 추출하고, 추출된 토픽들의 약 26.7%를 활용하여 도메인 토픽을 구성(또는 명명)한다. 또한, 구성된 도메인 토픽은 실제 프로젝트를 대표할 수 없는 토픽인 사실을 확인하였다. 반면, JTopic 시스템은 모든 프로젝트에 대해 임계값 이상의 토픽 분포 값을 갖는 토픽을 추출하였으며, 추출된 토픽 내 단어를 50%이상 활용함으로써 도메인 토픽을 구성할 수 있다. 특히, 구성된 도메인 토픽은 대상 프로젝트를 대표할 수 있는 토픽으로 매우 적절하다는 것을 확인하고 JTopic 시스템의 성능이 우수하다는 평가를 내렸다.

표 10 JTopic(J) 시스템과 MALLET(M) 평가 결과

프로젝트	JHotDraw		JEdit		CheckStyle	
시스템 / 항목	J	M	J	M	J	M
①	O	X	O	O	O	X
②	100%	X	53.3%	26.7%	86.7%	X
③	O	X	O	X	O	X

5. 결론

기존의 LDA 알고리즘을 활용하여 소프트웨어의 토픽을 추출하는 방법은 소프트웨어 내의 소스 코드의 주석 및 식별자와 같은 비구조적이고 텍스트에 기초한 정보만을 대상으로 한다. 기존 방법에 대한 결과 역시 사용자가 지정한 토픽 수만큼의 단어 집합과 단어 집합에 대한 분포 값이지만 각 분포 값은 매우 작으며 모든 분포 값이 고르게 나타난다. 토픽 분포 값이 작다는 것은 결국 각 토픽들이 소프트웨어의 도메인 토픽이 될 가능성이 매우 희박하다는 것을 의미하고 분포 값이 고르다는 것은 어느 하나의 토픽에 중점을 둘 수 없다는 것을 의미한다.

본 논문에서는 위와 같은 문제점을 보완하여 소프트웨어로부터 의미 있는 도메인 토픽을 추출하기 위한 방법을 제안했다.

소스코드내의 클래스 식별자는 해당 소스코드의 주제와 관련이 있지만 프로젝트 전체적으로 보았을

때에는 해당 클래스 식별자의 가치를 평가할 수 없다. 클래스 식별자의 가치를 평가하기 위해서, 소프트웨어로부터 클래스간의 호출 관계를 추출한 후 PageRank 알고리즘을 적용함으로써 각 클래스에 클래스별 중요도 값을 할당하였다. 그리고 주석 및 식별자를 추출하여 LDA 알고리즘을 적용할 대상 문서 집합을 생성할 때에 클래스 중요도 값에 따라 해당 클래스 식별자 단어를 추가해줌으로써 단어의 중요도 및 모든 문서에서의 중요도를 높일 수 있도록 하였다.

소프트웨어의 도메인 토픽 추출에 있어서, 낮은 토픽 분포 값과 함께 고르게 분포된 분포 값을 갖는 토픽들을 추출하는 기존 시스템들과는 달리, 본 논문에서 제안하는 시스템은 상대적으로 더 큰 토픽 분포 값을 갖는 토픽이 추출될 수 있도록 하였다.

이 후 연구에서는 소프트웨어의 재사용을 위해 소프트웨어 및 JTopic 시스템을 통해 추출된 도메인 토픽 외에 소프트웨어의 패턴 등을 추출하여 데이터베이스에 저장 한다. 사용자는 질의를 통해 원하는 소프트웨어를 찾아내고 해당 소프트웨어를 재사용할 수 있다.

참고문헌

[1] C. H. Papadimitriou, H. Tamaki, P. Raghavan, S. Vempala, “Latent Semantic Indexing: A Probabilistic Analysis”, Proceeding PODS'98 Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, 1998, pp. 159-168.

[2] D. Andrzejewski, A. Mulhern, Ben Liblit, Xiaojin Zhu, “Statistical Debugging Using Latent Topic Models”, 18th European Conference on Machine Learning(ECML), 2007.

[3] D. Blei, A. Ng, M.Jordan, “Latent Dirichlet Allocation”, Journal of Machine Leaning Research 3, Jan. 2003, pp. 993-1022.

[4] D. Blei, T.L. Griffiths, M.I. Jordan, J.B. Tenenbaum, "Hierarchical topic models and the nested Chinese restaurant process", In Advances in Neural Information Processing Systems, 2004.

[5] D. Steidl, B. Hummel, E. Juergens, “Quality analysis of source code comments”, IEEE 21st International Conference, May 2013, pp. 83-92.

[6] G. Maskeri, S. Sarkar, K. Heafield, “Mining Business Topics in Source Code using Latent Dirichlet Allocation”, Proceedings of the 1st India software engineering conference, 2008, pp. 113-120.

[7] H. U. Ansuncion, A. U. Ansuncion, R. N. Taylor, “Software tracebility with topic modeling”, ICSE, 2010, vol 1. pp. 95-104.

[8] K. Tian, M. Revelle, D. Poshyvank, “Using Latent Dirichlet Allocation for automatic categorization of software”, IEEE MSR, 2009, pp. 163-166.

[9] M. Alenezi, “Extracting High-Level Concepts from Open-Source Systems”, International Journal of Software Engineering and Its Applications, Vol9. 9, No. 1, 2015, pp. 183-190.

[10] M. F. Porter, “An algorithm for suffix stripping”, Morgan Kauffmann Publishers Inc. San Francisco, CA, USA, 1997.

[11] PW. McBurney, C. Liu, C. McMillan, T. Weninger, “Improving Topic Model Source Code Summarization”, Proceedings of the 22nd International Conference, 2014, pp. 291-294.

[12] S. Brin, L. Page, “The Anatomy of a Large-Scale Hypertextual Web Search Engine”, International World-Wide Web Conference(WWW), April 1998.

[13] S. Rao, A. Kak, “Retrieval from software libraries for bug localization: a comparative study of generic and composite text models”, IEEE MSR, 2011, pp. 43-52.

[14] T. Karrer, JP. Krämer, J. Diel, B. Hartmann, “Stacksplorer:call graph navigation helps increasing code maintenance efficiency”, Proceedings of the 24th annual ACM symposium on User interface software and technology(UIST), 2011, pp. 217-224.

[15] T. Savage, B. Dit, M. Gethers, D. Poshyvank, “Topic XP:Exploring Topics in Source Code using Latent Dirichlet Allocation”, IEEE International Conference, Sept. 2010, pp. 1-6.

[16] T. Weninger, Y.Bisk, J. Han, “Document-topic hierarchies from document graphs”, Proceedings of the 21st ACM international conference on Information and knowledge management(CIKM), 2012, pp. 635-644.

[17] W. Li, A. McCallum, "Pachinko allocation: DAG-structured mixture models of topic correlations", Proceedings of the 23rd international conference on Machine learning(ICML), 2006, pp 577-584.

소프트웨어 개발 프로세스를 대상으로 수행하는 추적성 분석의 세부 관계 정의

김재엽, 이동아, 유준범

건국대학교 컴퓨터공학
서울시 광진구 능동로 120 건국대학교
radic2510@gmail.com, {ldalove, jbyoo}@konkuk.ac.kr

요약: 본 연구에서는 표준 규격을 바탕으로 소프트웨어 개발 프로세스 전반에 걸쳐 수행되는 추적성 분석에서 사용되는 세부 관계를 정의한다. 기존의 추적성 분석에서 항목들 간의 연결로 추적을 표현하지만 다양한 관계의 종류를 분류할 필요가 있기 때문에 세분 관계를 정의하였다. 정의한 세부 관계를 ARINC 653 과 Qplus-AIR 요구사항 명세의 추적성 분석에 적용하여 효용성을 보여준다.

핵심어: 추적성 분석, ARINC 653, Qplus-AIR

1. 서론

소프트웨어 개발과정에서 검증과 관리, 유지보수의 중요성이 점차 증가함에 따라 추적성 분석에 대한 중요도가 증가하고 있다. 추적성 분석은 소프트웨어 개발과정에서 생산되는 산출물들이 어떠한 요구사항 요구부터 생산되는 것인지 추적관계를 통해 분석하는 것을 말한다[1].

기존의 추적성 분석에서는 특정 산출물이 어떠한 요구사항으로부터 생산된 것인지 일괄적인 표현방식을 사용하여 표현하는 것이 일반적이다. 하지만 실제 표준 규격을 바탕으로 수행되는 소프트웨어 개발 프로세스에서의 추적성 분석에서는 표준과 요구사항, 요구사항과 설계 간의 다양한 관계가 존재하기 때문에 일괄적으로 표현하는 것으로는 한계가 있다. 따라서 추적성 분석에서 다양한 관계를 표현하기 위한 세부 관계를 정의할 필요가 있다.

본 논문에서는 표준을 바탕으로 수행되는 소프트웨어 개발 프로세스의 추적성 분석에서 사용되는 세부 관계에 대해 정의한다. 그리고 정의한 관계의 효용성을 확인하기 위하여 ARINC 653 과 Qplus-AIR 간의 추적성 분석에 적용하고 그 결과를 확인하여보았다.

2. 추적성 분석

추적성 분석은 시스템 요소를 위한 요구사항과 계층관계에 있는 요구사항이나 설계, 구현 등과 같은 다양한 산출물과 연결하는 프로세스이다[2]. 추적성 분석은 개발 프로세스 전반에 걸쳐 수행되며 좁은 범위로는 문서 간의 분석을 수행해서 요구사항이 모두 반영되었는지 확인하는 용도로 사용되지만 넓은 범위로는 구현된 시스템이 초기 요구사항으로부터 어떤 설계와 구현과정을 거쳐 생성된 것인지에 대한 추적이 가능하다. 복잡한 시스템을 대상으로 하는 추적성 분석 결과는 정형 검증을 위한 속성을 도출하는 용도로 사용하는 것이 가능하다.

3. 추적성 분석 세부 관계

추적성 분석에서 사용되는 세부 관계를 5 가지로 나눠서 정의하여 <표 1>에 정리하였다. Reference 와 Application 은 기존의 추적성 분석에서 연결되는 관계를 세분화 한 것이고 No Relationship 의 세 가지 관계는 기존의 추적성 분석에서 추가로 고려되는 관계이다.

Reference 와 Application 은 기존의 추적 분석에서 연결되는 것을 세분화 한 관계이다. Reference 의 경우 요구사항 명세에서 표준을 직접적으로 언급하고 참조하라고 되어 있는 경우가 해당된다. Application 은 표준에서 요구하는 부분을 프로젝트에 적합하게 바꾼 경우에 해당된다. 예를 들어 임베디드 시스템의 경우 시간과 관련된 부분에서는 보드의 특성을 고려해야 하기 때문에 추가적인 고려가 필요하게 되는데 이러한 경우가 Application 에 해당한다. Reference 는 표준을 그대로 사용하는 관계이지만 Application 은 표준을 바꿔서 사용한다고 할 수 있기 때문에 요구사항에 대한 추가적인 검증이 요구될 수 있는 부분이다. 관계를 세분화 함으로서 집중적으로 검증을 수행하는 부분을 선택하기 용이해진다.

표 1 추적성 분석 세부 관계

이름	설명
Reference	표준과 같거나 표준을 참조를 표현한 경우의 관계
Application	표준의 항목을 프로젝트에 적합하게 변경하여 사용한 경우의 관계
No Relationship (Selection)	다양한 방법 중 일부를 선택하여 개발이 가능한 경우에서 다른 방법을 선택한 경우
No Relationship (Creation)	개발 프로세스에서 새롭게 생성된 경우로 요구사항으로부터 생성된 항목이 아닌 경우
No Relationship (Omission)	표준의 항목이 추적성 분석에서 연결되는 항목이 없는 경우 중 필히 연결되어야 하는 항목인 경우

No Relationship 은 추적성 분석으로 추적이 수행되지 않는 부분이다. 이에 해당하는 관계는 3 가지로 구분될 수 있는데 표준에서 다양한 방법 중 일부를 선택하여 개발이 가능한 경우에서 선택되지 않은 항목(Selection)과 산출물에서 새롭게 생성된 항목(Creation) 그리고 요구사항을 만족하지 못한 경우(Omission)이다. Selection 의 경우 표준에서 언급하긴 하지만 선택적으로 요구되는 부분이다. 예로 파티션 간 통신이 다양한 방법으로 구현될 수 있고 표준에서도 다양한 방법이 항목으로 기술되어 있지만 실제 구현에서는 특정 방법을 사용해서 구현하는 것이 이에 해당한다. Creation 의 경우 표준에는 존재하지 않지만 요구사항 명세에서 새롭게 생성되는 항목으로 대표적인 예로는 구현된 시스템의 하드웨어와 관련된 부분이 속할 수 있다. 두 관계는 기존의 추적성 분석에서는 나타나기 어려운 부분으로 추적성 분석을 수행한 사람이 아니라면 해당 부분이 Omission 또는 추적성 분석을 수행하지 않은 부분으로 착각하여 추가적인 비용을 낭비하게 된다. 추적을 나타낼 수 없는 부분에서도 이러한 관계를 표현하여 비용을 절약할 수 있다.

4. 적용 및 분석

본 연구에서 정의한 추적성 분석 세부 관계가 실제 시스템 개발 프로세스에서 다양한 연결 결과를 보여주는지 확인하기 위해서 ARINC 653 표준 규격[3]과 이를 바탕으로 만들어진 Qplus-AIR [4]의 요구사항 명세서 간의 추적성 분석 결과를 분류해 보았다. ARINC 653 은 195 개 Qplus-AIR 의 요구사항 명세서는 179 개의 항목을 가지는 문서이다.

추적성 분석의 대상이 되는 항목만을 추려내기 위한 기존의 추적성 분석을 수행하였으며 ARINC 653 의 항목을 195 개에서 155 개로 Qplus-AIR 의 항목을 179 개에서 169 개의 항목으로 추려내었다. <표 2>는 추적성 분석 결과를 위에서 정의한 5 가지 세부 관계로 분류한 결과이다. 총 270 개의 관계가 생성되었으며 이 중 65 개의 항목은 논문에서 정의한 관계에 의해서 추적성 분석이 수행된 것이다. 이 중 Omission 으로 관계를 분류해 놓은 27 개의 관계에 대해서는 추가적인 확인이 필요한 부분이다.

표 2 세부 관계 적용 결과

관계	개수
Reference	37
Application	168
No Relationship (Selection)	5
No Relationship (Creation)	33
No Relationship (Omission)	27

총 270 개의 관계 중 No Relationship 에 해당하는 65 개의 항목은 기존의 추적성 분석에서는 모두 Omission 인 경우로 해석되어 추가적인 비용을 요구하게 되는 부분이다. 세부 관계의 경우 이러한 경우를 포함하여 추적성 분석을 수행하기 때문에 비용의 절약이 가능하다. 하지만 현재 본 논문에서 정의하는 Omission 의 경우는 추적성 분석을 수행할 당시 기준의 설정이 적절하기 못한 경우 생길 수 있다는 점과 요구사항에서 의도적으로 생략한 부분에 대해서는 확인 할 수 없다는 점 등의 한계가 존재하기 때문에 추가적인 세분화가 요구된다. Reference 와 Application 은 기존의 추적성 분석에서도 연결이 가능하지만 세부 관계를 통해서 개발 프로세스 내의 다른 단계에서 활용할 수 있었다. Qplus-AIR 을 대상으로 정형검증을 수행하기 위한 검증 속성을 추적성 분석 결과를 활용하여 생성하였다. Reference 관계 보다 Application 관계를 검증하는 것이 더 중요하기 때문에 이를 위한 검증 속성 도출에 더 많은 비용을 소비하였다.

5. 결론 및 향후 연구

본 연구에서 정의한 5 가지 세부 관계는 추적성 분석에서 요구사항과 산출물 간의 관계 표현을 세분화하여 분류할 수 있다는 점에서 의미가 있다. 특히 표준을 기반으로 수행한 개발 프로세스에서 요구사항을 만족 여부에 대한 분석을 기존의 추적성 분석 보다 세분화하여 표현하는 것이 가능하기 때문에 검증 부분에서 얻을 수 있는 이점이 있다.

본 연구는 ARINC 653 표준 규격과 Qplus-AIR 의 요구사항 명세서 간의 추적성 분석 수행을 바탕으로 세부 관계를 정의하였지만, 향후 연구에서는 세부 관계 정의를 보다 세분화하여 정의하고 이를 기반으로 추적성 모델까지 정의하고자 한다. 또한 설계 및 구현까지의 추적성 분석을 수행하고 분석 결과를 평가하고자 한다. 정형검증을 위한 속성 추출 과정에서

세부 항목을 적용한 추적성 분석 결과의 효용성 분석을 수행하고자 한다.

사 사

본 논문은 한국전자통신연구원의 “오류없는 시스템 통합을 위한 안전우선분산 모듈형 SW 플랫폼” 사업의 지원으로 연구한 결과입니다.

참고문헌

[1] 이준기, 조혜경, 고인영, “요구사항 온톨로지 기반의 시맨틱 태깅을 활용한 산출물의 재사용성 지원을 위한 요구사항추적 방법,” 정보과학회논문지 : 소프트웨어 및 응용, 35(6), 357-365, 2008

[2] Antoniol, Giuliano and Canfora, Gerardo and Casazza, Gerardo and De Lucia, Andrea and Merlo, Ettore, “Recovering traceability links between code and documentation,” Software Engineering, IEEE Transactions on, Vol.28, No.10, pp.970-983, 2002

[3] Avionics Application Software Standard Interface, ARINC Specification 653P1-3

[4] 최두열, 장순용, 김재용, 류춘하, 조인제, 김태호, “ARINC 653 을 지원하는 Qplus-AIR 를 활용한 무인기용 비행제어 소프트웨어 개발 연구,” 한국항공우주학회 2012 춘계 학술대회, pp.1176-1180, 2012

Legacy 시스템의 안전성 분석기법 및 사례연구

김선주, 이석원
아주대학교 소프트웨어 특성화학과
ssunshine@ajou.ac.kr

요약: 안전에 관한 연구가 미진했던 우리나라도 세월호 사건을 계기로 안전필수 시스템(Safety-Critical System)의 중요성과 연구가 본격적으로 진행되고 있다.

미국과 일본을 비롯한 선진국에서는 다양한 안전필수 분야에서 STAMP(System-Theoretic Accident Model and Process)와 STPA(System-Theoretic Process and Analysis) 라는 새로운 안전성 분석모델과 기법이 활용되고 있다.

한편, 우리나라의 경우, 최근 몇 년 사이에 특정분야(원자력 시스템)에서만, 위의 기법을 적용하는 연구가 진행 중이다.

이에 STAMP 라는 안전성 분석 모델과 STPA 안전성 기법을 소개하고, 안전필수 시스템의 하나인 119 신고접수시스템의 사례연구를 통해, 해당 시스템에 향후 적용을 제시해 보고자 한다.

핵심어: safety-critical 시스템, 안전성 분석, STAMP, STPA, 119 신고접수시스템,

1. 서론

1.1 연구배경

2000 년대에 진입한 후 과거와 다르게 대규모 재난이나 재해가 발생하고 있다. 911 테러를 비롯하여, 후쿠시마 원자로 폭발사건, 가까이는 우리나라 세월호 침몰 사건 등 인재를 포함하여 수많은 자연재해도 발생하고 있다. 인간의 힘으로 제어할 수 없는 자연재해도 있지만, 정치적인 원인의 테러, 시스템의 고도화에 따른 SOC(사회간접자본)의 오류로 (예: 원자로 제어시스템, 철도 제어시스템, 항공관제시스템과 같은 안전필수 시스템의 사고) 인한 인명과 재산 피해가 심각해 지고 있다.

안전에 관한 연구가 미진했던 우리나라의 경우 2014 년 4 월에 발생한 세월호 사건을 계기로 안전필수 시스템의 연구가 본격적으로 시작되고 있다.

안전필수 시스템(Safety-critical System)이란 시스템에서 발생하는 사고가 인명 피해나 심각한 환경훼손, 막대한 재산피해를 일으킬 수 있는 시스템을 의미한다[4]

최근 활발하게 사용되고 있는, 새로운 안전성 분석 모델로는, Nancy Leveson 의 STAMP (System-Theoretic Accident Model and Process)와 STAMP 를 기반으로 안전성을 분석하는 기법인 STPA(System-Theoretic Process Analysis)가 있다[5][15].

위의 STAMP/STPA 는 기존의 분석 기법인 시스템 요소의 오류 발생시 해당 이벤트의 문제를 해결하고 오류 발생을 막는 관점에서 확장하여, 시스템의 요소간에 상호작용에서 발생할 수 있는 위험요소들을 도출하여 분석하고, 안전제약을 통해 위험을 제거하는 기법이다[5][15].

선행연구에서 보는 바와 같이 미국과 일본 등 선진국에서는 항공, 교통, 의료, 철도 등의 안전필수 시스템에서, 폭 넓게 활용되고 있다.

한편 우리나라의 경우, 원자력 분야[1][2]에서만 STAMP/STPA 적용 연구가 진행되고 있으나, 실생활에서 가장 많이 접하는 안전필수 시스템인 119 신고 접수 시스템은 안전성 분석이 이루어지고 있지 않고 있다.

1.2 연구 제안

지난 2014 년 3 월 13 일, 서울연구원이 2011 년 7 월 27 일 발생한 우면산 산사태에 대한 제 2 차 원인이 더불어 집중호우에 대한 대비 부족으로 인한 인재임[3]을 추가로 발표하였다.

우면산 사태 분석결과에서 보는 것과 같이 집중호우로 인한 천재지변이 원인인 것으로 명백히 보이는 재난도, 다각적인 분석 결과 단순 천재지변이 아님이 밝혀 졌다. 즉, 재해, 재난, 사고에는 사람과 조직, 시스템 등이 유기적으로 연관되어 있다는 것을 알 수 있다[13].

이에 119 신고접수시스템의 안전성 분석에 적합한, STAMP 라는 안전성 모델과 STPA 라는 안전성 기법을 제안 하고자 한다.

STPA 는 STAMP 라는 사고모델(accident model)을 분석하는 새로운 안전성 분석 기법이다[5][15].

이 기법은 시스템과 시스템 요소 자체의 오류가 없더라도, 요소들간의 상호 작용으로 인해 발생할 수 있는 위험 요소들을, 적절한 제약을 통해서, 제거하는 기법이다[5][15]. 무엇보다 STPA 는 하드웨어, 소프트웨어, 사람, 조직(Socio)까지 모두 포함하여 위험요소를 분석하는데 기존의 안전성 분석기법과의 차별화 된 특징이 있다.

2. 선행연구

선행연구를 통해서, STAMP/STPA 가 Chain of event 기반의 기존 모델이 사용하는 기법에서 고려되지 못한 사람과 Socio-technical 한 부분까지 분석의 대상으로 포함하여, 문제를 해결한 것을 확인할 수 있다.

2.1 외국의 STAMP/STPA 적용사례 연구

2.1.1 일본

일본에서 아래와 같은 사례 연구로, 다양한 분야에서 활용되고 있다.

1. 차량에서 사용되는 동력전달장치 시스템에 STAMP/STPA 를 적용한 사례연구[6]
2. 공장에서 사용되는 열 조종장치 시스템에 STAMP/STPA 를 적용한 사례연구[7]
3. 원자력 발전소의 컨트롤 시스템을 대상으로 발생할 수 있는 사고에 대한 분석을 수행한 사례연구[8]

2.1.2 미국

원자력, 의료기기, 항공분야에서도 STAMP/STPA 가 적용되어 사용되고 있다.

1. 원자력 발전소의 컨트롤 시스템을 대상으로 발생할 수 있는 사고에 대한 분석을 수행한 사례 연구[9]
2. 의료기기의 분석에도 활용된 사례 연구[10]
3. 항공분야에서 우주사고에 대한 분석/평가를 수행한 사례 연구[11]

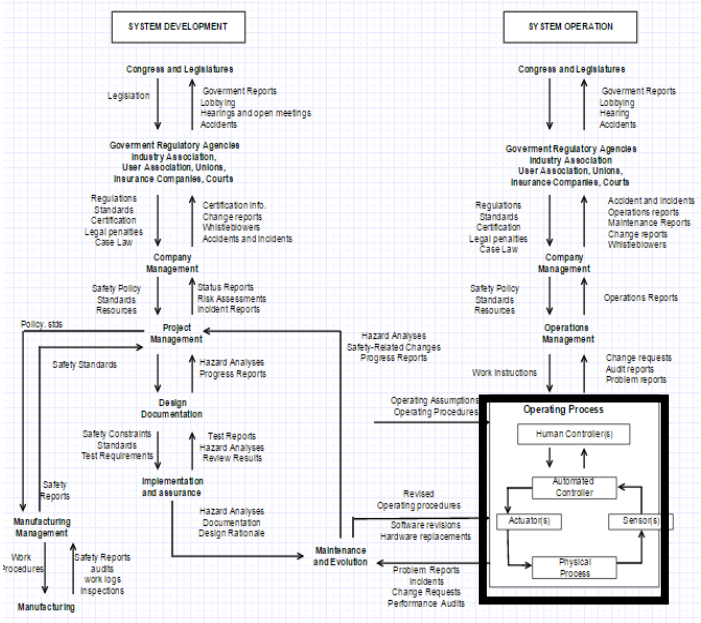
3. 안전성 분석기법

3.1 STAMP 소개

STAMP 는 사고가 발생할 때, 사고와 관계가 있는 요소들 상호간에 어떤 인과 관계가 있는지 분석하는 모델이다.

STAMP 는 3 가지의 기본 개념으로 구성되는데, 첫번째는 안전제약(Safety Constraints) 있고, 두번째는 계층형(A Hierarchical Safety Control Struture) 구조로 되어 있으며, 세번째는 프로세스 모델(Process model)을 포함하여야 한다[5][15].

Figure1. STAMP 의 Socio-technical 제약구조 모델



첫번째의 안전제약은[1] STAMP 의 제일 기본이 되는 개념으로 사고의 발생은 시스템 동작의 제어가 부족하기 때문에 발생한다는 개념으로 레벨에 맞는 적절한 제어를 통해 시스템 안전을 도출한다[5].

두번째의 계층형 구조[2]는 시스템을 계층형으로 나타낸 후 상위레벨에서는 제어, 절차, 정책 등을 결정하고 하위레벨에서는 제어, 절차, 정책 등을 실제로 수행한 후 상위레벨로 결과를 보내어 계층간의 피드백을 통해 시스템 안전을 도출한다[5].

세번째의 프로세스 모델[3] 은, 시스템 모델로 Controller 와 Controlled Process 가 포함되며, 시스템과 사고(accident)에 대한 계층적인 제어 구조를 표현하기 위한 목적으로 사용된다[5].

컨트롤이 필요한 시스템 모델이 Controller 에 포함되어야 하며, 시스템의 현재 상태 및, 프로세스 방식이 표현되어야 한다. Controller 는 Controlled Process 통해, 동작 수행 시 control action 을 보내고, feedback 을 받게 된다.

3.2 STPA 소개

STPA 는 STAMP 모델을 분석하는 안전성 분석 기법이다[5]. STPA 는 STAMP 모델로 시스템을 나타낸 후에, 사고의 원인으로 이어질 수 있는 부적절한(unsafe)한 control action 을 찾아서, 그 원인을 분석한다[15].

STPA 는 4 단계의 절차를 통해 안전성 분석을 수행하게 되며, 해당 절차는 다음과 같다[15]

(1) 시스템의 위험, 안전제약 상황을 설정
(2) 시스템의 Control Structure 도출(구축)
(3) STPA step1 - 시스템의 위험요인이 될 수 있는 부적절한 제어동작(unsafe control action) 식별
(4) STPA step2- STPA 단계 1 에서 도출된 부적절한 Control 이 어떤 경우에 일어나는지, 잠재적인 원인을 식별

STPA 를 수행하기 위해서는, 위의 절차 순서대로 시스템의 위험, 안전제약 상황을 정의하고, Control Structure 를 도출한다. Control structure 가 완료되면, STPA 의 첫 단계에서는 위험을 발생시키는 부적절한 제어를 찾게 되며, 부적절한 제어는 다음과 같이 4 가지 분류하고 있다[15]

[1] Figure1 참조- 각 레벨별로 필요한 제약이 주어짐

[2] Figure1 참조- 시스템 운영과 개발시 프로세스에 따라 계층형으로 나누고, 상하 레벨별로 필요한 제약과 수행 후 결과에 대한 feedback 이 이루어짐

[3] Figure1 참조- Process 모델은 Mental 모델로도 불리며, 위 그림에서, 굵은 선으로 표시됨.

(1) 필요한 안전제어가 제공되지 않는 경우
(2) 제공된 안전제어로 인해 위험을 일으키는 경우
(3) 안전 제어가 잘못된 순서로 제공되거나, 빨리 혹은 늦게 제공 되는 경우
(4) 제공된 안전제어가 일찍 멈추거나 오래 제공되는 경우

위와 같이 4 가지의 위험한 제어에 대하여 밝혀낸 후에는, 위험한 제어들이 발생할 수 있는 원인을 찾는다. 그리고, 원인을 분석하고 발생 가능한 시나리오를 확인 한 후에, 이를 제거하게 된다.

4. 사례 연구

서울방재센터에서 운영중인 119 신고접수 시스템은, 위급한 상황으로부터 국민의 생명, 신체 및 재산을 보호하기 위한 시스템으로서, 24 시간 운영되고 있고, 시스템의 오류가 생명과 연결되어 있는 safety-critical 시스템중의 하나이다. (Figure2.참조)

Figure 2. 119 신고접수 처리 시스템

한국의 119 신고 접수 시스템은 각 지역별 소방 본부별로 운영시스템 및 프로세스가 별도로 구성되어 나, 시스템을 전부 분석할 수 없으므로, 한국에서 가장 큰 도시이며, 가장 많은 인구로 인해 가장 많은 신고건수가 발생하는 서울소방방제 센터의 119 신고접수 시스템을 기준으로 설명하고자 한다.

첫 번째 사례로, 2011 년 7 월 27 일 오전 8 시를 기점으로 집중호우로 인한, 우면산 산사태가 발생하여, 18 명이 사망하고, 170 억 이상의 피해액이 발생하는 사고가 발생했다. 그 당시 서울방재센터의 119 신고접수 건수는 평소 신고접수건수의 3 배인 17,000 여건이 접수 되었다.

우면산 사태와 같은 비상재난이 발생하면, 서울방재센터에서는 119 신고접수시스템의 비상재난 선포화면을 통해 재난을 선포(등록)하고, 비상재난 프로세스에 따라서 신고접수가 이루어지게 된다.

시스템을 통해서 비상재난이 선포된 경우, 센터에서는 비상재난사고에 대한 접수와 사고 통계처리만 하고, 실질적인 비상사태에 대한 처리는(현장 차량출동) 사고지점의 소방서와 안전센터가 수동으로 자체적으로 하도록 되어 있다.

즉, 비상재난 시스템의 리소스 분배 주체가 센터임에도 불구하고, Control 하지 못하고, 소방서와 안전센터에 비상재난임을 전달만 하는 단계에 머물러 있다.

두 번째 사례로, 2014 년 8 월 25 일에 발생한 부산폭우[4]로 인해 시간당 1 천 900 통에 이르는 폭주가 발생하였고, 폭주로 인해, 시스템이 마비 되는 사건이 발생하였다. 119 신고접수시스템의 경우, 신고자의 전화가 접수자 와 연결됨과 동시에, 신고자의 위치정보가 시스템을 통해서 확인되고, 접수자 는 해당 사고의 위치를 확인하여, 가까운 소방서/안전센터의 필요한 인력 및 출동을 시스템을 통해서 할 수 있다. 하지만, 이 사건의 경우, 전화를 받는 인력 부족의 문제, 119 신고 폭주로 연결되지 않을 때, 별도의 회선 및 프로세스 부재 등의 조직과 프로세스, 폭주로 인한 시스템 마비 등의 융합된 문제로 인해, 통화중인 119 전화접속 실패로 인해, 차량 안에서 구조를 기다리다가 사망하는 사고로 연결되었다.

위의 사례를 통해서, 시스템 장애 발생시, 오류를 해결하거나, 장애처리 절차를 따라 문제를 해결하는 관점에서 확장하여, 연계된 시스템에서 발생할 수 있는 위험 요소들과 각 요소들간의 리소스 부족 문제들을 확인하여, 시스템이 안전하게 작동하는지에 대한 검증이 필요하다.

5. 결론

최근까지 119 신고접수시스템에 대한 연구는 사고접수에서 현장 출동까지의 골든 타임을 줄이기 위해, 관련된 하드웨어나, 통신, 네트워크, 조직과 프로세스 개선에 초점을 두고 연구되어왔다.

우리나라에서도 최근 몇 년 사이에 원자력분야에 STAMP/STPA 적용에 관한 연구가 진행 되고 있으나, 실생활에 밀접한 안전필수 시스템인 119 신고접수시스템 자체에 대한 안전성 분석은 연구되고 있지 않은 상황이다.

이에 본 논문에서는 STAMP/STPA 라는 새로운 안전성 분석모델과 기법을 소개하고, 119 신고접수 시스템의 사례 연구를 통해 안전성 분석이 필요성을 설명하였다.

위의 사례 연구를 통해 볼 때 STAMP/STPA 와 같은 안전성 분석이 필요한 이유는 다음과 같다. 첫째로, 하드웨어와 소프트웨어에서 오는 시스템의 오류뿐만 아니라 사람과 조직간의 리소스 부족 등의 문제에서

[4] 부산폭우로 인해, 통화량 폭주로 인한 119 시스템마비와 사망사건 기사
http://www.kookje.co.kr/news2011/asp/newsbody.asp?code=0300&key=20140829.22001205700
http://www.nocutnews.co.kr/news/4079972#todaynewscon

오는 위험요소들을 정의하거나 도출할 수 있기 때문이다. 둘째, 각 요소뿐만 아니라 요소 간의 문제점을 해결하기 위한 안전제약을 강화하여 문제를 해결할 수 있기 때문이다. 셋째, 기 사용중인 시스템(Legacy)에 다른 하드웨어, 소프트웨어, 또는 프로세스를 추가할 경우, 설계단계에서 필요한 안전제약을 같이 고려 함으로써, 개발과 적용 후의 위험성을 줄여, 안전성을 높일 수 있다.

6. 향후 계획

기 논문의 최종목적은, 사용중인 Legacy 시스템(119 신고접수시스템)에, 안전성분석모델(STAMP)와 안전성 분석기법(STPA)을 통해서, Risk를 제거하고, 안전성을 평가하는 것에 있고, 기 논문은 사례연구를 통해 필요한 모델과 기법을 제시하는 도입에 해당하는 것이므로, 119 신고접수시스템에 대한 구체적인 분석 내용을 기록하지 못한 한계가 있다.

이에 향후 연구 계획은 첫째로 119 신고접수 시스템의 주요 프로세스인 출동지령과 비상재난에, STAMP 모델과 STPA 기법을 단계별로 적용하여, 발생가능한 사고 위험을 줄이는 방법을 제시 하고 둘째로 STPA 에 기반한 안전 요구사항을 도출하여, 안전성 테스트[14]까지 포함할 계획이다.

또한 최근 다수의 인명이나, 재산 피해를 일으키는 큰 사고의 원인에는, 과거의 주요 사고의 원인이었던 천재지변에 의한 피해를 제외하고, 시스템 운영이나, 개발 시에 의도치 않은 실수나, 악의적인 의도에 의한 행위로, 대재앙으로 까지 이어지는 경우가 점점 증가되고 있다[12]. 이 두 가지 행위 모두, 사람의 사고 오류를 포함한 요소들간의 작용이 주요원인인 되고 있는 바, 이 부분의 안전성 분석 방법에 대해서도 확대해서 연구할 계획이다.

Acknowledgment

이 논문은 2013년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(2013M3C4A7056233).

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW특성화대학원 지원 사업의 연구결과로, 진행되었음 (R0346-15-1017).

참고문헌

[1] Dong-Ah Lee, Jang-Soo Lee, Se-Woo Cheon, Junbeom Yoo. “Application of system- theoretic process analysis to engineered safety features-component control system”. Proceedings of the 37th enlarged halden.

[2] 이동아, 김의섭. 유준범. “원자력 발전소 I&C 시스템의 안전성 분석을 위한 신기술 적용 사례”. 한국정보과학회, 2013.

[3] 우면산 사태에 대한 서울연구원 2 차조사결과 기사. http://www.yonhapnews.co.kr/society/2014/03/13/0701000000AKR20140313060300004.HTML.

[4] Nancy G. Leveson. “SafeWare: system safety and computers”. Addison-Wesley, 1995.

[5] Nancy G. Leveson. “The Need for New Paradigms in Safety Engineering”. Springer-Verlag, 2010.

[6] Yasuhiko Kawabe, Tatsuya Yanagisawa. “Applying STAMP/STPA to Human Safety System for Four Wheel Drive Power-train”.

[7] Morishita N, “Applying STAMP/STPA to Analyze the Cause of the Unexpected Fire Happening at the Heat Treatment Process”. 2014 STAMP Workshop, 2014.

[8] Torok, R., and B. Geddes. "Systems Theoretic Process Analysis (STPA) Applied to a Nuclear Power Plant Control System”. Presentation at MIT STAMP Workshop, 2013.

[9] Thomas IV, John P. “Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis”. Diss. Massachusetts Institute of Technology, 2013.

[10] Samost, Aubrey. “Evaluating Systems with Multiple Processes Using STPA: A Case Study in a Medical Intensive Care Unit”.

[11] Nancy G. Leveson. “Evaluating Accident Models using Recent Aerospace Accidents”. Technical Report, MIT Dept. of Aeronautics and Astronautics, 2001.

[12] William Young, Nancy G. Leveson. “An integrated Approach to Safety and Security Based on Systems Theory”. Communications of the ACM, 2014.

[13] Ali Avni Cirik, David Mendonca. “Responding to Disaster in Socio-technical System”. IEEE international Conference, 2009.

[14] Changyong Yang. “Software Safety Testing Based on STPA”. ISAA, 2013.

[15] Nicolas Dulac, Nancy Leveson. “An Approach to Design for Safety in Complex Systems”.

프로젝트 중심 소프트웨어 공학 교육의 성과 분석과 개선 방안

최은만, 김하영

동국대학교 컴퓨터공학과
서울 중구 필동 3 가 26
emchoi@dongguk.edu, leannain@naver.com

요약: 이 논문은 프로젝트 중심의 소프트웨어 공학 교육이 얼마나 효과적이고 어떤 부분에 문제가 있는지 분석한다. 또한 컴퓨터 과학 전공의 소프트웨어 개발 능력 교육에 대한 캡스톤 과목으로써 성공적인 교육 모델을 제시한다. 소프트웨어 공학 교육의 핵심적인 부분은 실제 소프트웨어 제품을 개발하는 것과 같은 환경에서 첨단 원리와 방법, 절차를 실제적으로 훈련하는 것이다. 이 논문에서는 전통적인 소프트웨어 공학 교육의 성과를 분석해 보고 그 결과를 바탕으로 프로젝트 교육에서 플립 러닝을 이용한 통합적 교육 모델을 제시한다.

핵심어: 프로젝트 기반 학습, 소프트웨어 공학 교육, 플립 러닝.

1. 서 론

소프트웨어 공학을 가르치는 일은 복잡하고 어려운 일이다. 사용자 요구를 찾아내고 모델링하여 소프트웨어로 구현하기 위하여 설계 및 프로그래밍하며 테스팅하고 유지보수 작업까지 방대하고 다양한 주제를 학생에게 전달하여야 하기 때문이다. 소프트웨어 엔지니어로 훈련시키기 위해서는 소프트웨어 라이프 싸이클의 각 단계에 대한 다양한 방법을 이해할 뿐만 아니라 도구와 프로세스를 경험하여야 소프트웨어 시스템을 성공적으로 개발할 수 있다.

강의식 교육은 소프트웨어 공학 주제에 충분하지 않다. 소프트웨어 공학을 배우는 학생은 경험을 통하여 기술과 통찰력을 습득하여야 하기 때문이다. 기술을 전수받기 위하여 달인과 견습생 사이에 끊임없는 실습과 전문가적인 조언이 필요하다. 예를 들면 좋은 설계와 나쁜 설계 사이의 차이는 미묘하고 복잡한 설계 원리와 패턴에 의하여 갈린다. 설계에서의 잘못이 전문가에서 지적되고 그 차이를 느낌으로써 객체지향 설계 기술을 배울 수 있는 것이다.

소프트웨어 개발의 경험을 얻기 위하여 소프트웨어 공학 강의에서는 프로젝트 과제를 부여하는 경우가 많다. 하지만 프로젝트 과제가 학생의 학습 성과를 측정하기 위하여 평가되는 경우는 많지만 교육 성과를 분석하기 위하여 평가되는 경우는 많지 않다. 학생들의 그룹 작업은 창의력을 발휘하기 위한 동기도 되며 소프트웨어 공학의 각 프로세스를 경험하기에 좋은 방법이다. 이 논문에서는 프로젝트 결과를 소프트웨어 공학 교육의 성과에 대한 분석 대상으로 보았다. 아마추어인 학생들의 프로젝트 결과는 수행이 되지 않을 수도 있고 신뢰성 또는 유지보수성 등의 품질의 결함이 있을 수도 있다. 이런 부족함이 소프트웨어 공학 교육의 내용과 어떤 측면이 부족한 것인지 파악하기 위한 연구가 필요하다.

이 연구에서는 학부과정의 소프트웨어 공학 교육을 받은 학생들의 그룹 프로젝트 25 개를 분석하여 교육 성과를 평가하였다. 각 프로젝트는 3~4 명의 학생이 1 학기에 프로젝트 계획부터 요구분석, 설계, 구현 및 테스팅 작업의 결과물이다. 도제식 교육의 전문가적 조언을 위하여 디자인 스튜디오를 운영하였고 교수와 조교가 프로젝트 오너가 되었다.

프로젝트 결과물은 소프트웨어 공학 강의 계획에 포함된 여러 개념과 방법, 스킬이 제대로 사용되었는지에 초점이 맞추어져 있다. 또한 Steve Tockey[1]가 제안한 소프트웨어 엔지니어가 가져야 할 지식과 스킬을 바탕으로 프로젝트 결과물을 평가하여 보았다. 특히 소프트웨어 결과물의 품질은 코드에 대한 품질이 중요하다. 따라서 학생들이 구현한 코드에 대하여 리팩토링이 적용될만한 코드 스멜을 찾고 어떤 설계와 구현 스킬이 부족한지를 파헤쳤다.

교육 성과의 분석 결과를 바탕으로 프로젝트 중심 소프트웨어 공학 교육에 적합한 통합 교육 방법을 제시한다. 통합 교육 모델은 수동적인 강의식 교육에서 탈피하여 적극적인 학습이 될 수 있는 오프라인 수업과 이를 뒷받침 할 수 있는 온라인 교육을 아우른다.

프로젝트 중심 교육의 내용과 프로젝트 시나리오에 대하여 2 장에서 설명하고 프로젝트 교육 성과에 대하여 3 장에 소개한다. 4 장에는 효과적인 프로젝트 교육을 위한 통합 교육 모델을 제시하고 5 장에서 결론을 맺는다.

2. 프로젝트 중심 교육

프로젝트 중심 소프트웨어 공학 교육의 핵심은 그

야말로 한 학기 동안 쓸만한 소프트웨어 시스템을 개발하는 프로젝트이다. 따라서 적극적인 학습이 이루어지는 오프라인 수업은 개발 프로세스 단계를 따라 이루어지는 그룹 활동과 디자인 스튜디오가 주를 이룬다. 강의실 밖에서의 교육은 온라인에서 이루어진다. 즉 하이브리드 형태의 플립 러닝으로 구성되었다.

2.1 교육 내용

표 1 에 나열한 것이 소프트웨어 공학 교육이 다루는 주요 주제들이다. 대부분의 주제가 팀 소프트웨어 개발의 핵심이 되는 분석 및 설계와 협력 코딩 작업에 맞추어져 있다. 강의 주제는 프로젝트 프로세스와 동기화 되어 있으며 오프라인 형태의 교육과 퀴즈 및 질문세션이 포함되어 있다.

표 1 교육 내용

지식 영역	지식 단위	교육 내용
모델링과 분석	모델링 기초	모델링 원리(분할, 추상화, 일반화, 프로젝션/뷰 등), 모델링 언어, 정형적 모델
	모델의 종류	정보 모델링(ER 다이어그램, 클래스 다이어그램) 행위 모델링(구조적 분석, 상태 다이어그램, 유스케이스 분석, 인터랙션 다이어그램 등) 임베디드 시스템 모델링, 엔터프라이즈 모델링
	요구 기초	요구의 정의, 추출, 변경 및 관리, 명세와 검증
소프트웨어 설계	설계 개념	설계 원리(정보은닉, 응집력, 결합력), 설계 목표가 되는 품질(신뢰성, 사용용이성, 성능, 보안 등) 설계 Trade-Off, 설계 전략(함수 중심, 객체지향)
	아키텍처 설계	아키텍처 스타일(파이프-필터, 계층, 트랜젝션 중심, Peer-to-peer, 클라이언트-서버 등) 도메인 중심 아키텍처, 프로덕트 라인
	HCI 설계	HCI 설계 원리, 모드와 네비게이션, 코딩과 비주얼 디자인, 반응시간과 피드백, Internationalization
	상세 설계	설계 패턴, 컴포넌트 설계, 설계 표현 방법
	설계 지원도구와 평가	설계 지원도구, 설계 측정 메트릭(응집, 결합, 정보은닉)
코딩과 테스팅	코딩	코딩 스타일, 리팩토링, 객체지향 프로그래밍 원리, SVN
	테스팅	블랙박스, 화이트박스 테스팅, 디버깅
	커뮤니케이션 능력	독해(코딩과 문서), 쓰기, 팀 커뮤니케이션(전화, 이메일, 회의) 프레젠테이션 능력, 공동 작업 방법

2.2 프로젝트 시나리오

프로젝트는 4 명의 팀원이 4 단계의 프로세스로 진행하였다. 모델링과 분석을 배운 후에는 SRS(요구 분석서)를 설계를 배운 후에는 SDD(설계 문서)를 작성하였고 테스트 계획서를 준비하게 한 후 코딩 및 테스팅 작업이 이어졌다.

프로젝트 모든 작업의 초점은 코딩에 맞추어져 있다. 따라서 가급적 불필요한 문서는 줄이되 코딩에 필요한 설계는 매우 상세하게 요구하였다. 설계는 UML 다이어그램으로 표현하였고 도구를 사용하여 효과적으로 변경, 전환할 수 있다.

개발하는 소프트웨어의 유형은 모바일로부터 웹 기반, 임베디드, 유틸리티에 이르기까지 다양하다. 프로젝트 문제를 자유롭게 한 이유는 학생들의 창의력을 발휘할 수 있는 좋은 기회를 살리기 위함이며 다양한 시스템을 접하게 하려는 의도 때문이다.

각 프로세스가 끝날 무렵 디자인 스튜디오를 통하여 빈약한 설계를 보충하고 문제가 될만한 부분을 고쳐나가게 교육하였다.

3. 프로젝트를 통한 교육의 성과 분석

성과 분석의 목표는 학생들이 작성한 프로젝트 코드에서 과연 소프트웨어 공학 교육 내용의 어떤 부분이 부족한지를 찾는 것이다.

실험 대상은 25 개 프로젝트 팀 중 수상을 한 5 팀과 나머지에서 뽑은 5 팀으로 구성하였다. 10 팀은 대부분 안드로이드 앱을 개발하였고 프로그램 규모는 그림 1 과 같다.

그림 1 프로젝트 팀들의 작성 코드라인 통계

10 팀의 원시 코드들의 품질을 분석하기 위해 룰 기반으로 원시코드를 분석할 수 있는 PMD[4]와 소스 모니터[5] 두 가지 도구를 이용하였다.

3.1 PMD 로 분석한 코드 품질

PMD(Programming Mistake Detector)는 자바 정적 분석 도구로 원시 코드에 적용할 수 있는 분석 규칙(rule set)을 제공한다. 본 논문에서는 중복코드 검사 기능과 순환 복잡도(cyclomatic complexity)를 찾는 룰을 이용하여 프로젝트를 분석하였다. 이 두 가지 검사는 리팩토링으로 제거해야 할 문제 있는 코드(bad smell) 중 가장 중요한 중복된 코드와 긴 메소드를 찾는 것과 각각 관련이 있다.

표 2 프로젝트 별 중복 코드 및 순환복잡도 누적 분포

프로젝트 팀	중복 코드	순환 복잡도 (평균 복잡도 / 품질)	
Usb Sniffer	39	11	3.4
여행의 시작	202	12	16.7
뭐해 운동해	83	11.7	5.8

복지 매니저	24	14	2.0
Daisy	155	11	3.8
Class Top	132	11.3	7.3
Plan Bee	113	13	1.9
Study hi	160	11	4.8
Dongnagong	66	15	3.3
Baby	147	12	5.1
평균/표준편차		12.2/1.31	5.41/4.08

발견된 중복코드를 분석한 결과 대부분이 데이터베이스 JDBC 커넥션을 생성하거나 인터페이스를 만드는 코드 부분에서 발생한 것을 알 수 있었다. 이러한 중복은 템플릿 메소드 패턴이나 팩토리 메소드 패턴을 이용하면 코드를 간결하게 관리할 수 있다.

또 다른 매트릭인 순환복잡도는 메소드 내의 결정 포인트(if, while, for 등의 분기문)를 통해 소스의 복잡도를 알 수 있다. 특정 메소드의 순환복잡도 값이 10 이 넘어가면 복잡하다고 평가하며, 품질은 KLOC 당 복잡도로 측정하였다. 평균 복잡도가 12.2 로 측정되어 높은편이며 이러한 순환 복잡도를 낮추기 위해서 메소드 추출 리팩토링 방법을 적용하면 효과적인 케이스가 많았다.

원시코드 내에 산재된 중복 코드는 수정 사항이 발생할 때마다 중복된 횟수만큼의 수정 작업이 필요하고, 순환 복잡도가 높은 코드는 가독성이 떨어지기 때문에 코드의 유연성이 떨어진다. 따라서 중복 코드를 디자인 패턴의 설계에 따라 코딩하면 코드의 유지보수 비용을 줄이고 코드의 품질도 높일 수 있다.

3.2 소스 모니터로 분석한 코드 품질

소스 모니터는 소스 코드를 분석하여 코드 매트릭을 도출하고 이 매트릭을 통해 소스코드의 복잡도를 측정할 수 있는 프로그램이다. 그림 2 는 소스모니터의 기능 중 하나인 Kiviat Graph 로 코드에서 수집한 7 가지 매트릭의 통계를 나타낸 그래프이다.

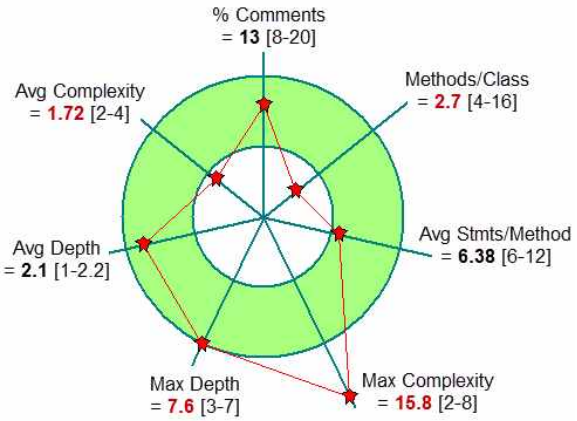


그림 2 Kiviat Graph로 나타낸 10팀들의 평균

10 개의 프로젝트들의 매트릭들을 Kiviat Graph 로 비교한 결과 그림 2 와 같은 형태를 지닌다. 그림의 7 가지 축은 나쁜 냄새(Bad smells)를 측정할 수 있는 지표이고, 각 축의 값은 프로젝트 팀들의 평균이다. 정상적인 코딩 스타일의 매트릭 범위를 대괄호와 색칠된 부분으로 표시하였고, 색칠된 영역을 기준으로 벗어나는 값은 잠재적인 오류가 있을 가능성이 높은 부분으로 판단할 수 있다. 예를 들어 최대 중첩도(Max Depth)는 3 에서 7 이 권장 범위이고 프로젝트 평균은 범위를 벗어난 7.6 임을 알 수 있다.

가장 범위에서 벗어난 값이 두드러지게 나타난 지표는 클래스당 메소드 수(Methods/Class)가 2.7 로 평균보다 작게 나타났고, 최대 복잡도(Max Complexity)가 15.8 로 매우 높게 나타났다. 즉 대부분의 프로젝트 코드는 메소드의 크기가 크고 복잡하며, 한 메소드가 가진 기능이 많기 때문에 클래스당 메소드 수는 작은 것을 파악할 수 있었다.

소프트웨어 공학 수업에서 소프트웨어 라이프 사이클의 단계에 따라 프로젝트를 진행하는 것은 실무 경험이 없는 학생들이 직접 기획부터 테스팅까지 체계적으로 개발할 수 있다는 점에서 긍정적이다. 하지만 개발 프로세스 단계를 따라 완성하는 것이 소프트웨어의 품질을 보장하지는 못한다. 개발한 소프트웨어의 품질을 높이기 위해서는 설계 단계에서부터 구현할 프로젝트의 상세한 이해를 바탕으로 알맞은 설계 원리와 구현 스킬을 적용하는 것이 필요하다. 위 프로젝트 교육 결과의 문제점은 코드 분량에 비해 품질이 낮다는 것이다. 추상 클래스를 이해하여 상속을 이용하는 경우가 드물었고, 클래스 내부의 변수를 public 으로 선언하는 오류가 많아 객체지향의 특성인 정보 은닉을 지키지 못하였다.

설계 원리, 디자인 패턴, 리팩토링 등의 소프트웨어 공학의 유용한 방법론들을 강의로만 듣고 프로젝트를 시작한다면 프로그램이 동작되기 위한 기능을 개발하는 것에서 그치게 된다. 이런 문제점을 방지하기 위해서는 디자인 스튜디오에서 디자인 패턴을 적용하여 개선된 코드를 만드는 연습을 통해 이론들을 체득하는 것이 좋다. 코드 품질을 개선할 수 있는 패턴들을 체계적으로 적용하면 원시 코드에서 나타날 수 있는 오류들을 설계와 구현단계에서부터 제거할 수 있다. 그러므로 간단한 패턴을 구현해보고 실제 프로젝트에도 적용하는 것이 필요하다.

4. 교육 개선 방안

소프트웨어 개발에서는 항상 문제가 되는 것은 프로젝트 경험이 없는 또는 새롭게 투입된 인력이 제대로 된 생산성을 발휘하기 위하여 매우 긴 시간이 걸리기 때문이다. 한편 하드웨어 엔지니어링에서는 이 문제를 오래 전에 해결하였다. 표준화 된 배선도로 투입된지 하룻만에 납땜을 하여 생산성을 낼 수

있다. 소프트웨어 개발도 프로그래밍 언어를 표준화하고 표준 라이브러리를 도입하였지만 한계가 있다.

더욱 효과적인 방법은 디자인 패턴을 적용하는 것이다. 디자인 패턴은 객체 기반 시스템에서 골격을 이루는 클래스들이 어떤 행태로 연관을 맺어야 하고 어떻게 메시지를 보내야 하는지를 잘 나타낸 것이다. 즉 소프트웨어 기본 단위들에 대한 연결 방법과 조직을 표현한 것이라 할 수 있다.

최근 소프트웨어는 매우 복잡해지고 다양해져서 객체 기반 시스템을 도입하지 않으면 다루기 어렵다. 따라서 소프트웨어 개발 교육에서는 디자인 패턴에 대한 학습이 중요하다. 리서치랩에 의한 산업 분야별 필수 교과목 조사결과[6]에 의하면 디자인 패턴 과목의 수요도는 SI 분야(4 위), Potal(2 위), 패키지 솔루션(3 위), 금융(2 위), 빅데이터(1 위) 모두 상위에 랭크되어 있다. 하지만 학부 과정에 다루는 학교는 매우 적은 수에 불과하다.

디자인 패턴이나 리팩토링은 강의식 수업에 적합하지 않다. 강의식 수업은 수용적 러닝, 교수자 중심이지만 디자인은 개인적 스킬 중심이기 때문이다. 또한 21 세기 창의의 시대에 요구되는 인재상은 질문, 비판적 사고력, 능동적 학습자, 협업 중심이다. 따라서 패러다임의 전환(Flipped Learning)이 필요하다. 수동적 학습은 교실 밖에서 온라인으로 이루어지며 능동적 학습은 교실 안에서 문제 해결 중심의 협업으로 이루어진다. 즉 온라인과 오프라인이 혼합된 형태(Blended Learning)가 적합하다.

표 3 플립 러닝 강의

	교실 안	교실 밖
교수	설계 과제 제시 소그룹 활동 지원	온라인 강의지원(강의 자료 제공)
학생	탐구지향적 문제 해결 학습 소그룹활동(설계, 발표 등)	개별적으로 온라인 강의 수강

커리큘럼 및 강의 내용은 UML 과 설계 방법이 주를 이룬다. 특히 객체지향 코딩 원리(단일 책임, 개발폐쇄, 리스코프 대체, 인터페이스 분리, 의존 관계 역전) 같은 것이 중요하다. Bad Smell 및 리팩토링은 코드 품질을 높이는데 아주 중요하다. 또한 무엇보다도 디자인 패턴(GoF)을 어느 수준에서 가르치는가가 소프트웨어 설계 교육에 관건이다.

온라인 강의의 시퀀스는 도입, 개념, 개념 적용, 예제, 개념 확인 순으로 이루어진다. 특히 오프라인 수업과 연결이 중요하다. 이를 위하여 연결 문제(Bridge Exercise)가 잘 설계 되어야 한다. 연결 문제는 오프라인 수업에서의 챌린지 문제를 푸는데 필요한 여러 요소 개념을 확인하기 위한 것이다.

오프라인 강의 구성은 먼저 연결 문제에 대한 학습결과를 발표하게 하고 이의 풀이를 설명한다. 또한 동영상 강의에 대한 질의 응답 시간을 갖는다. 오프라인 강의의 중요한 핵심은 소그룹 중심의 설계 및 구현이다.

마지막으로 오프라인 강의에서의 중요한 점은 챌린지 과제이다. 챌린지 과제는 그룹 멤버들이 가진 지식을 총동원하여야 풀이를 생각할 수 있는 높은 수준의 설계과제로써 [8]에서 그 샘플을 찾아볼 수 있다.

5. 결 론

산업화 시대의 교육과 정보화 시대의 교육은 달라야 한다. 공급자 중심의 교육에서 개인 학습 중심, 암기 위주에서 활동과 사고 중심으로 되어야 한다. 이런 측면에서 소프트웨어 공학 교육의 성과를 분석하여 보았다. 소프트웨어 공학 교육의 핵심인 프로젝트 결과를 기반으로 분석한 결과 코드 품질을 개선할 수 있는 패턴들을 체계적으로 적용하는 것이 부족하다. 패턴과 리팩토링 기법의 훈련을 강화함으로써 설계 수준을 끌어 올릴 수 있고 코드 품질을 향상시킬 수 있다. 디자인 패턴과 리팩토링 기법은 강의식 교육이 아니라 프로젝트 중심의 코칭 스타일이 되어야 효과적으로 교육할 수 있다.

참고문헌

[1] Steve Tockey “Recommended Skills and Knowledge for Software Engineers”, Proceedings. 12th Conference on Software Engineering Education and Training, pp. 168 – 176, 1999.

[2] Sohan Yadav, Kiambing Xiahou, “Integrated Project Based Learning in Software Engineering Education”, Proceeding on 2010 International Conference on Educational and Network Technology, pp. 34-36, 2010.

[3] John Knight, Thomas Horton, “Evaluating A Software Engineering Project Course Model Based On Studio Presentations”, Proceeding on 35th ASEE/IEEE Frontiers in Education Conference, 2005.

[4] How to write PMD tutorial, http://pmd.sourceforge.net/

[5] Source Monitor Version 3.6 http://www.campwoodsw.com/

[6] 산업분야별 필수교과목 조사결과, 리써지 랩, 2013.

[7] L. Dee Fink, Creating Significant Learning Experiences – An Integrated Approach to Designing College Courses, Jossey-Bass, 2003.

[8] Steven Metsker, Design Patterns Workbook, Addison-Wesley, 2002.

DO-178C 기반 소프트웨어 동적 테스트 방안

윤상은, 이종민, 정영은

한국정보통신기술협회, 소프트웨어시험인증연구소
경기도 성남시 분당구 분당로 47
gmlssnfl@tta.or.kr

요약: 본 연구에서는 항공용 소프트웨어 개발 가이드라인인 DO-178C 를 적용하여 소프트웨어 동적 테스트를 수행하는 절차를 고찰한다. 이를 위해 소프트웨어 테스트 준비단계, 테스트 케이스 작성단계, 테스트 수행 단계, 커버리지 분석단계 별로 만족해야 할 기준, 수행 작업, 적용 가능 기법 등을 기술하였다. 마지막으로 DO-178C 에서 소프트웨어 테스트를 효율적으로 수행하기 위한 방안을 제안하였다.

핵심어: DO-178C, 소프트웨어 테스트, 동적 테스트, 테스트 커버리지, 구조 커버리지

1. 연구배경

DO-178C 는 미연방항공국(FAA)의 표준화 기관인 RTCA 에서 지난 2011 년 12 월 표준으로 제정한 " 항공용 시스템 및 장비 인증을 위한 소프트웨어 고려사항(Software Considerations in Airborne Systems and Equipment Certification)" 이다. RTCA 는 이전 버전인 DO-178B 에 비해 요구사항의 정의 및 프로세스 시작과 종료 조건을 개선하였으며 최신 SW 개발 기법을 반영하기 위해 DO-330(도구검증), DO-331(모델링), DO-332(객체지향) 및 DO-333(정형기법) 등의 표준도 함께 제정하였다[2][3][4][5].

최근 MFD(Multi Function Display, 다기능 시현기) 등 항공용 시스템의 국내 개발이 추진되면서 항공용 소프트웨어 개발 및 검증 표준인 DO-178C 가 주목받고 있어 DO-178C 의 동적 테스트 프로세스에 대해 살펴보고자 한다.

2. 기본 개념

2.1 소프트웨어 테스트

소프트웨어 테스트는 동적 테스트와 정적 테스트로 분류할 수 있다. 동적 테스트는 요구사항 문서, 분석서, 설계서 등의 테스트베이시스(test basis)를 기반으로 입력값/사전조건과 예상결과/사후조건으로 구성된 테스트 케이스를 작성한 후 이를 평가 대상 소프트웨어로 실행하여 실제결과와 예상결과를 비교하여 소프트웨어의 결함을 찾는 기법이다. 이에 반해 정적 테스트는 소프트웨어를 직접 실행하지 않고 설계문서와 소스코드를 분석하거나 코딩표준, 체크리스트 등을 통해 검토하여 결함을 찾아내는 기법이다.

2.2 DO-178C

DO-178C 에는 소프트웨어 개발 시 만족해야 할 목표(objective)를 제안하고 있다. 소프트웨어 결함 발생 시 발생하는 인적, 물적 피해의 크기에 따라 결정되는 소프트웨어 수준(SL, Software Level, 표 1 참조) 별로 각 목표들을 테일러링하여 적용하도록 제시하고 있다.

Software Level	Failure Description	# of Objective
A	Software Resulting in a **catastrophic failure** condition for the system	71
B	Software Resulting in a **hazardous or severe-major failure** condition for the system	69
C	Software Resulting in a **major failure** condition for the system	62
D	Software Resulting in a **minor failure** condition for the system	26
E	Software Resulting in **no effect** for the system	0

표 1. DO-178C 및 DO-278A 의 보증수준[1]

가장 높은 수준인 A 수준에서는 모든 목표를 만족해야 하며, 소프트웨어 수준이 낮아질 수록 만족해야 할 목표가 적어진다. 목표의 개수는 A(71 개), B(69 개), C(62 개)로 비슷한 수준이며, D 수준의 경우 26 개로 현저히 낮아진다. 이는 A/B/C 수준에서는 요구사항 정의, 분석, 설계, 구현 및 시험과 관련한 목표들이 포함되어 있으나, D 수준에는 요구사항 정의 및 시험과 관련한 목표만이 정의되어 있기 때문이다. E 수준은 만족해야 할 목표가 없으므로 사실상 DO-

178C 를 적용하지 않는 경우이다.

또한 소프트웨어 동적 테스트에 대해서는 고수준 및 저수준 요구사항 기반 테스트 케이스 생성, 테스트 수행 및 테스트 커버리지 분석까지의 절차를 제시하고 있다.

2.3 소프트웨어 테스트 커버리지

소프트웨어 테스트 커버리지는 특정 테스트케이스 혹은 테스트케이스의 집합이 테스트 대상 소프트웨어를 얼마나 충분히 테스트 했는지에 대한 측정 기준이다. 테스트 커버리지는 테스트 케이스(집합)을 얼마나 잘 만들었는지에 대한 측정 값이며 테스트 커버리지 측정 값이 높을 수록 테스트를 충실히 수행한 것이다. 테스트 커버리지는 아래와 같이 명세기반 커버리지와 구조기반 커버리지로 나눌 수 있다.

- 명세기반 커버리지 : 요구사항, 설계서 등에 명시된 항목을 얼마나 충실히 테스트 하였는지 측정

- 구조기반 커버리지 : 프로그램의 논리 흐름을 얼마나 충실히 테스트 하였는지 측정

명세기반 커버리지의 대표적인 예로는 요구사항 커버리지(Requirement Based Coverage)가 있으며 이는 요구사항 정의서 및 설계서에 기록된 항목에 대한 테스트 케이스가 작성되었는지 측정함으로써 계산된다. 일반적으로 요구사항 커버리지는 요구사항 및 설계항목과 테스트 케이스간의 추적성이 유지되는 경우 만족한다고 간주하므로 테스트 케이스가 테스트 대상을 충분히 테스트하는지 판단하기 어렵다는 단점이 있다. 이를 보완하기 위해 DO-178C 에서는 단일 요구사항(설계항목)에 대해 정상범위(normal) 테스트 케이스와 비정상범위(robust) 테스트 케이스를 모두 테스트 할 것을 제시하고 있다.

구조기반 커버리지는 작성된 테스트케이스가 소스코드 내의 문장, 분기, 조건 등 논리적인 구조를 테스트 했는지 측정하는 방법이다. 항공, 자동차, 철도, 의료 및 원자력 등 안전중요 소프트웨어 분야에서는 명세기반 커버리지를 적용함과 동시에 측정기준이 명확한 구조기반 커버리지를 테스트 커버리지 측정 기준으로 제시하고 있다. 구조기반 커버리지는 아래와 같은 기준으로 측정된다.

- 문장 커버리지(Statement coverage) : 테스트케이스가 테스트 대상 프로그램에 존재하는 문장을 얼마나 테스트 했는지 측정

- 분기(결정) 커버리지(Branch coverage) : 테스트케이스가 테스트 대상 프로그램에 존재하는 분기문(예 : IF 문)의 분기(True/False)를 얼마나 테스트 했는지 측정

- 조건 커버리지(Condition coverage) : 테스트케이스가 테스트 대상 프로그램에 존재하는 분기문 내의 기본조건(예 : X > 1)을 얼마나 테스트 했는지 측정

- 조건/분기 커버리지(Condition/Decision coverage) : 테스트케이스가 테스트 대상 프로그램에 존재하는 분기문의 분기와 분기문 내의 기본조건을 얼마나 테스트 했는지 측정

- 다중 조건 커버리지(Multiple Condition coverage) : 테스트케이스가 테스트 대상 프로그램에 존재하는 분기문 내의 기본조건의 논리적인 조합을 얼마나 테스트 했는지 측정

- 변경 조건/분기 커버리지(Modified Condition/Decision Coverage) : 테스트케이스가 테스트 대상 프로그램에 존재하는 MC/DC 진리값의쌍을 얼마나 테스트 했는지 측정

- 기본 경로 테스트(Basis path test) : 테스트케이스가 테스트 대상 프로그램에 존재하는 기본경로 집합을 얼마나 테스트 했는지 측정

- 데이터 커플링 커버리지(Data Coupling Coverage) : 테스트케이스가 테스트 대상 프로그램에 존재하는 함수간의 데이터 전달을 얼마나 테스트 했는지 측정

- 컨트롤 커플링 커버리지(Control Coupling Coverage) : 테스트케이스가 테스트 대상 프로그램에 존재하는 함수간의 호출을 얼마나 테스트 했는지 측정

- 추가 코드(Additional Code) : 테스트케이스가 컴파일러 등에 의해 추가적으로 생성되어 설계내역과는 추적되지 않는 코드를 테스트 했는지 측정

구조 커버리지	소프트웨어 수준			
	D	C	B	A
문장		필수	필수	필수
분기			필수	필수
MC/DC				필수
데이터 커플링			필수	필수
컨트롤 커플링		필수	필수	필수
추가코드				필수

표 2. 소프트웨어 수준별 구조 커버리지 목표[1]

DO-178C 에서는 표 2 와 같이 소프트웨어 수준별로 구조기반 커버리지를 만족하도록 요구하고 있다.

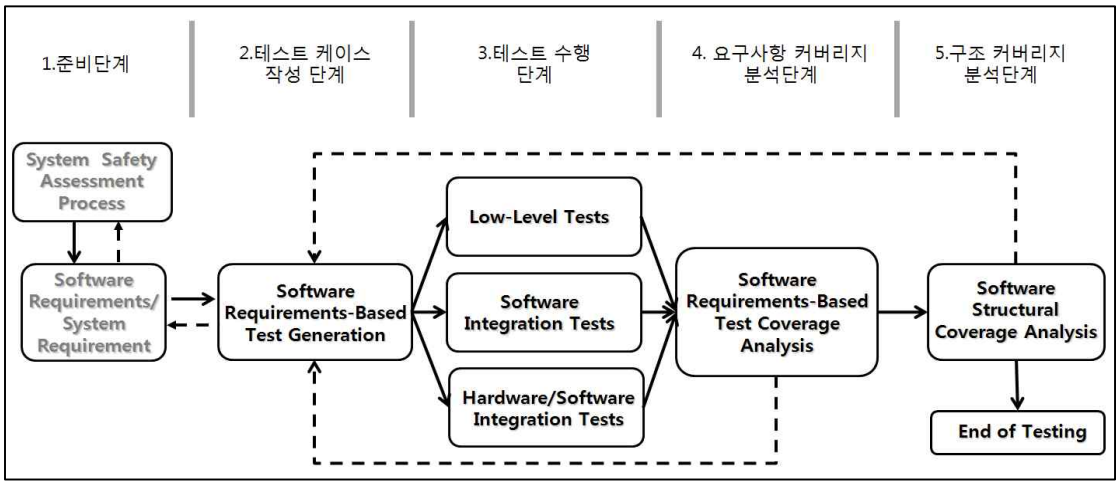


그림 1. DO-178C 의 소프트웨어 테스트 프로세스

3. DO-178C 의 소프트웨어 동적 테스트 절차

DO-178C 에서 제시하는 소프트웨어 동적 테스트 절차는 그림 1 과 같이 5 단계로 구성할 수 있으며 단계별 activity 는 아래와 같이 구성된다.

가. 준비 단계 : 시스템 안전성 평가(System Safety Assessment) 및 시스템 기능 분석을 통해 도출된 시스템 요구사항 및 소프트웨어 요구사항 검증

나. 테스트 케이스 작성 단계 : 소프트웨어 요구사항에 기반하여 테스트 케이스 작성

다. 테스트 수행 단계 : 작성한 테스트 케이스로 단위(Low-level) 테스트, 소프트웨어 통합 테스트, 하드웨어/소프트웨어 통합 테스트 수행

라. 요구사항 커버리지 분석 단계 : 수행된 테스트가 모든 소프트웨어 요구사항을 검증 했는지 분석하여 확인

마. 구조 커버리지 분석 단계 : 수행된 테스트가 소프트웨어 수준에서 제시하는 구조 커버리지 목표를 달성 했는지 분석하여 확인

3.1 준비단계

FTA(Fault Tree Analysis) 등의 기법을 통해 시스템 안전도 평가를 수행하면 시스템의 위험도를 완화시키기 위한 안전 요구사항(Safety requirement)이 식별되며, 안전 요구사항은 시스템의 기능 요구사항과 함께 시스템 요구사항으로 도출된다. 시스템 요구사항을 하위 시스템으로 전개하는 과정을 반복적으로 수행하여 특정 컴퓨터 유닛(unit)까지 세분화 하면 해당 컴퓨터 유닛의 요구사항을 구현하기 위한 하드웨어 및 소프트웨어 요구사항을 도출할 수 있다.

최초에 소프트웨어에 할당된 요구사항을 고수준 요구사항(high level requirement)이라 하며 이를 저수준 요구사항(low level requirement)으로 반복적으로 세분화한다. 세분화 하는 과정에서 상위수준 요구사항에 속하지 않지만 새롭게 도출되는 하위수준 요구사항을 파생(derived) 요구사항이라 한다. 최종 단계의 저수준 요구사항은 추가나 변경 없이 소스코드로 구현이 가능해야 한다.

동적 테스트 케이스 작성 전에 도출된 요구사항들에 대한 검증이 이루어져야 하며, 요구사항이 DO-178C 에서 제시하는 목적(objective)을 만족하였을 때 테스트 케이스 작성을 시작한다.

3.2 테스트케이스 작성 및 테스트 수행단계

요구사항에서 테스트 케이스를 도출하는 기법은 동등 분할(Equivalence partitioning), 경계값 분석(Boundary value analysis), 결정 테이블 테스팅(Decision table testing), 상태 전이 테스팅(State transition testing), 조합 테스팅 (Pairwise testing) 등 다양한 기법이 있으며 요구사항을 적절히 테스트 할 수 있는 기법을 사용한다. 일반적으로 하위수준 요구사항은 결정 테이블 기법을 통해 테스트 케이스를 생성하는 것이 커버리지를 높일 수 있다. 테스트 케이스 생성 후 구조 커버리지 측정을 지원하는 자동화 도구를 사용하여 테스트 케이스를 실행한다.

DO-178C 표준에서 제시하는 목표(objective) 입증 시 추가적인 분석 없이 도구의 결과만을 사용한다면 해당 도구는 DO-330 표준에서 제시하는 도구검증(Tool Qualification) 목표를 만족해야 함을 제시하고

있다. 구조 커버리지의 경우 DO-178C 에서 제시하는 목표 중의 하나이며 도구에서 측정한 구조 커버리지 결과를 사람이 재검증하는 것은 매우 비효율적이므로 구조 커버리지 측정에 사용하는 도구는 반드시 DO-330 기준을 만족해야 한다.

3.3 테스트 커버리지 분석단계

DO-178C 를 만족하려면 소프트웨어 수준에 관계없이 요구사항 커버리지를 100% 만족해야 한다. 하지만 구조 커버리지는 표 2 와 같이 소프트웨어 수준별로 만족해야 할 목표를 다르게 제시하고 있다.
요구사항 및 구조 커버리지 목표를 달성하지 못하면 테스트 케이스를 추가한다. 하지만 소프트웨어나 시스템 요구사항이 불충분하여 테스트 케이스를 추가하여도 구조 커버리지를 만족하지 못하는 경우 요구사항을 보완하고 테스트 케이스를 추가한다.

4 결 론

일반적인 소프트웨어의 동적 테스트는 요구사항 커버리지를 100% 만족하는 것으로 비교적 쉽게 테스트 목표를 달성할 수 있다. 하지만 DO-178C 의 경우 동적 테스트 시 구조 커버리지를 만족해야 하므로 테스트 목표달성이 매우 어렵다. 이를 극복하기 위해서는 3.2 절에서 소개한 체계적인 기법을 적용하여 테스트 케이스를 도출해야 한다. 하지만 요구사항을 부실하게 작성한 경우 테스트 케이스 도출에 투입한 노력과 상관없이 구조 커버리지 목표를 달성할 수 없다. 뿐만 아니라 개발 후반에 추가된 요구사항을 설계하고 구현해야 하는 예상치 못한 작업이 발생하며, 소프트웨어 개발일정이 지연될 수 있다. 그러므로 DO-178C 에 기반하여 소프트웨어 테스트를 효율적으로 수행하기 위해서는 동적 테스트 수행 이전에 요구사항 정의, 분석, 설계 및 구현 단계의 산출물이 부족함 없이 작성되었음을 검증하는데 많은 노력을 기울여야 한다.

Acknowledgement. 본 연구는 국토교통부 항공안전기술개발사업의 연구비지원(14ATRP-A085964-01)에 의해 수행되었으며, 본 연구 내용의 일부는 국토교통부 CNSTODAY 제 4 호에 수록되었음 [6]. 본 논문에 수록된 내용은 저자 개인의 의견으로 소속기관의 공식 견해가 아님.

참고문헌

[1] RTCA, Software Considerations in Airborne Systems and Equipment Certification(DO-178C), 2011
[2] RTCA, Software Tool Qualification Considerations, (DO-330), 2011
[3] RTCA, Model-Based Development and Verification Supplement to DO-178C and DO-278A(DO-331), 2011
[4] RTCA, Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A(DO-332), 2011
[5] RTCA, Formal Methods Supplement to DO-178C and DO-278A(DO-333), 2011
[6] 국토교통부, CNSTODAY 4 호, 2015 년 2 월

안전 무결성 기준을 활용한 테스트 케이스 우선순위

송옥수, 김보운, 최병주

이화여자대학교 컴퓨터공학과
서울특별시 서대문구 이화여대길 52 이화여자대학교
oksu@ewhain.net, bowoonk@ewhain.net, bjchoi@ewha.ac.kr(교신저자)

요약: 본 논문에서는 안전 중심 임베디드 시스템(Safety Critical Embedded System)에서의 안전성(Safety)을 보장하기 위해 안전 무결성 기준(Safety Integrity Level)을 활용한 테스트 케이스 우선순위에 대해 제안한다. 이 기법을 적용하여 안전 무결성 기준이 높은 기능의 우선 검증을 통해 결함을 조기에 찾아 수정함으로써 위험(Risk)을 낮추고자 한다.

핵심어: 테스트 케이스 우선순위, 안전 무결성 기준(SIL)

1. 서론

최근 제한된 시간과 비용 내에서 고품질의 소프트웨어 시스템을 개발하기 위해 효율적인 테스트 수행을 위한 다양한 연구들이 진행 중이다. 효과적인 결함 검출과 효율적인 테스트 수행을 위한 연구로는 테스트 스위트 최소화(Test Suite Minimization), 테스트 케이스 선택(Test Case Selection), 테스트 케이스 우선순위(Test Case Prioritization)가 있다.

테스트 스위트 최소화는 테스트 스위트 크기를 줄이기 위해 프로그램이 테스트 되면서 측정되는 커버리지를 기반으로 불필요한 테스트 케이스들을 식별하고 제거한다. 테스트 케이스 선택은 수정된 소스코드가 이전 버전의 시스템에 부정적인 영향을 끼치지 않음을 보장하기 위해 프로그램의 수정과 관련된 테스트 케이스를 선택한다.

테스트 스위트 최소화와 테스트 케이스 선택은 테스트 시간을 줄이는 반면 결함을 검출할 수 있는 중요한 테스트 케이스를 놓칠 수 있다는 문제가 제기되었다. 테스트 케이스 우선순위는 전체 테스트 스위트를 실행하되 개발 초기에 결함을 검출하기 위해 테스트 케이스 실행의 이상적인 순서를 찾는 것을 목적으로 한다. 현재 대부분의 테스트 케이스 우선순위 기법들은 소프트웨어 시스템을 실행했을 때 수집한 정보를 기반으로 테스트 스위트에 우선순위를 할당한다[1].

최근 자동차와 국방/항공, 의료와 같은 안전 중심 임베디드 시스템(Safety Critical Embedded System) 분야에서 제한된 자원 내에서 고품질을 유지하기 위한 효과적인 방법으로 모델 기반 개발 프로세스가 제기되면서, 모델 기반 테스트 케이스 우선순위 기법에 대한 연구가 증가하고 있다[2].

기존의 모델 기반 테스트 케이스 우선순위 기법은 요구사항의 변경과 관련된 전이들에 가중치를 부여하여 테스트 케이스에 우선순위를 적용하고 있다. 이는 리그레션 테스트(Regression Test)를 통해 요구사항 변경에 따른 수정된 소스코드가 원래의 시스템에 부정적인 영향을 끼치지 않음을 보장한다. 그러나 안전 중심 시스템의 경우, 기능 테스트를 완전히 수행하는 것으로 기능 안전성을 보장 할 수 없다. 따라서 안전 관련 요구사항을 우선 검증하는 것이 필요하다.

본 논문에서는 기능 요구사항과 안전 요구사항(Safety Requirement)을 충족시키기 위해 요구사항 변경과 안전 무결성 기준(SIL: Safety Integrity Level)[3]에 가중치를 부여하여 안전 중심 임베디드 시스템에서 SIL 등급이 높은 기능의 안전 관련 결함을 조기에 검출하고 수정하여 안전성를 보장하고자 한다.

본 논문은 다음과 같이 구성된다. 2 장은 관련연구로써 기존 테스트 케이스 우선순위 기법에 대해 소개하고, 3 장은 안전 무결성 기준을 활용한 테스트 케이스 우선순위 기법을 제안하고 4 장에서 적용사례를 기술한다. 5 장에서는 결론과 향후 연구에 대하여 기술한다.

2. 관련연구

테스트 케이스 우선순위 중 요구사항 변경을 활용한 모델 기반 테스트 케이스 우선순위 기법과 이를 적용할 경우 나타나는 문제점을 설명하고자 한다.

모델 기반 테스트 케이스 우선순위는 결함의 조기 검출을 위해 모델을 사용하여 테스트 케이스의 우선순위를 결정하는 것을 말한다. 대표적인 모델 기반 테스트 케이스 우선순위는 요구사항을 활용한 테스트 케이스 우선순위[4], 모델의 커버리지를 만족시키기 위해 이벤트 흐름 분석을 기반으로 한 테스트 우선순위[5], 모델을 실행하여 얻은 정보를 기반으로 전이들 간의 의존성 분석하고 이를 사용한 테스트 케이스 우선순위[6]가 있다.

본 논문에서 제안하는 방법과 가장 유사한 [4]는 요구사항 변경과 관련된 전이들을 식별하고 이를 반영한 테스트 케이스 우선순위 방법을 다음과 같이 제안하고 있다.

● 선택적 테스트 케이스 우선순위

요구사항 변경과 관련된 전이를 실행하는 테스트 케이스를 랜덤하게 선정하는 기법이다.

● 발견적 테스트 케이스 우선순위 #1

요구사항 변경과 관련된 전이를 많이 실행하는 테스트가 결함을 검출할 가능성이 높다는 점을 반영한다. 변경된 전이를 많이 실행하는 테스트 케이스가 가장 높은 우선순위가 되도록 한다. 동일한 우선순위의 경우는 랜덤으로 결정한다.

● 발견적 테스트 케이스 우선순위 #2

발견적 테스트 케이스 우선순위 #1 을 수정한 버전으로, 낮은 우선순위의 테스트들도 결함을 검출하는 가능성이 있기 때문에 낮은 우선순위의 테스트들도 반영하는 방법이다. 높은 우선순위와 낮은 우선순위를 함께 반영하여, 가장 높은 우선순위의 테스트 케이스를 선정하고 이를 제외한 나머지에서도 번갈아 가며 선택하는 방법이다.

● 발견적 테스트 우선순위 #3

요구사항 변경과 관련된 전이가 최우선적으로 반영되도록 하는 방법이다.

● 모델 의존성 기반 테스트 우선순위

모델의 추가 또는 삭제된 전이들과 이를 제외한 나머지 전이들과의 상호 관계를 데이터 의존성(Data Dependency)과 제어 의존성(Control Dependency) 관계를 기반으로 파악하여 테스트 케이스 우선순위를 결정하는 방법이다.

[4]에서 수행한 실험결과에 따르면 발견적 테스트 케이스 우선순위 #3 과 모델 의존성 기반 테스트 케이스 우선순위의 결함 검출율이 더 높다고 하였다. 이는 요구사항 변경과 관련된 전이의 실행 횟수만을 고려하는 것은 결함 검출에 있어서 큰 영향을 끼치지 않음을 의미한다.

발견적 테스트 케이스 우선순위 #3 은 모델 의존성 기반 테스트 케이스 우선순위보다 구현하기 쉽고, 비교적 적은 정보만을 필요로 하기 때문에 비용 대비 더 효율적인 방법이라 할 수 있다. 그러나 모든 결함이 동일한 심각도를 가지고 있다는 전제하에 적용할 수 있기 때문에, 본 논문에서 고려하는 안전 관련 결함들에 대한 심각도를 고려하지 않는다는 문제점이 있다.

3. 안전 무결성 기준을 활용한 테스트 케이스 우선순위

본 장에서는 안전 무결성 기준을 활용한 모델 기반 테스트 케이스 우선순위에 대해 정의하고, 다음 장에서는 이를 차량의 Button Engine Start (BES)시스템[7]에 적용한 사례를 보이고자 한다.

안전 무결성 기준을 활용한 테스트 케이스 우선순위는 요구사항 변경과 관련된 전이를 식별하고, SIL 등급이 높은 기능과 관련된 전이를 식별한 뒤 가중치를 계산하여 테스트 케이스에 우선순위를 결정한다. SIL 등급이 높은 기능과 관련된 전이(ST: Safety Transition)를 요구사항 변경과 관련된 전이(MT: Modified Transition)보다 높은 가중치를 부여함으로써 안전 요구사항이 변경되지 않더라도 안전 관련 기능들이 우선적으로 테스트 되도록 한다. 예를 들어 자동차의 경우는 ISO26262[8]에 제시된 ASIL 에 따라 표 1 과 같이 가중치를 부여할 수 있다. 표 1 에서 ASIL C, D 등급의 기능들은 결함이 발생함으로 인해 사람의 생명을 위협할 수 있기에 같은 가중치를 부여하였다.

표 1 자동차 임베디드 시스템의 가중치 예제

		가중치
MT	요구사항 변경	1
ST	ASIL A, B	2
	ASIL C, D	3

본 논문에서 제안하는 기법을 요약하면 그림 1 과 같이 나타낼 수 있다. 이때 테스트 케이스 우선 순위 방법은 다음과 같다.

① 요구사항 변경과 관련된 전이 식별

변경된 요구사항과 변경 이전의 요구사항을 비교하여 이와 관련된 전이들을 식별한다.

② 안전 무결성 SIL 기준과 관련된 전이 식별

안전 무결성 기준이 높은 기능의 우선검증을 위해 이와 관련된 전이들을 식별한다.

③ 테스트 케이스 우선 순위 식별

각 테스트 케이스 t 에 대하여 식별된 MT 와 ST 의 전이에 가중치를 부여한 W(t)를 계산한다. W(t)가 높은 순으로 테스트케이스의 우선순위를 할당하며, 동일한 경우 랜덤으로 우선순위를 결정한다.

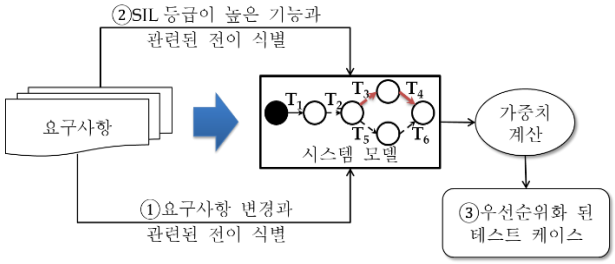


그림 1 테스트 케이스 우선순위 결정 과정

4. 적용 사례

본 논문의 적용사례 대상인 BES 시스템은 운전자가 자동차 키 (Mechanical Key)를 사용하는 것 대신 차에 장치된 시동 버튼(Stop/Start Button, SSB)을 누름으로써 자동차를 작동시키는 것이다. 또한 [7]의 스펙에 따르면 운전자가 특별한 행동을 하지 않아도 스티어링 컬럼(ESCL)의 잠금(Lock)과 풀림(Unlock)을 관리한다. 예를 들어 운전자가 브레이크를 밟고 SSB을 누르면 Fob 인증을 실시한다. Fob 인증이 완료되면 BES 시스템은 스티어링 컬럼의 Lock/Unlock을 실시하고, 엔진을 제어한다.

BES 시스템은 ESCL 제어, 버튼 엔진 시동 등으로 구성한다. 본 논문의 지면의 한계상 버튼 엔진 시동을 대상으로, 기어가 P 단일 경우로 한정하여 사례 결과를 기술하겠다. 해당 요구사항은 표 2 와 같다. 변경된 요구사항은 R_004 와 R_006 이다.

표 2 기어 P 단 버튼 엔진 시동 관련 요구사항

요구사항 항목		전이
R_001	스마트키가 차량 내에 있을 경우 P 단에서 SSB 버튼을 누르면 ESCL 이 Unlock 되고, ACC 로 전원이 이동한다.	T_1
R_002	ACC 상태에서 P 단인 경우 SSB 버튼을 누르면 IGN 으로 전원이 이동한다.	T_2
R_003	ACC 상태에서 P 단인 경우 1 시간 방치 시 밧데리 방전을 방지하기 위해 시동이 꺼진다.	T_3
R_004 (변경전)	IGN 상태에서 P 단인 경우 SSB 버튼을 2 번 누르면 시동이 꺼진다.	
R_004	IGN 상태에서 P 단인 경우 SSB 버튼을 누르면 시동이 꺼진다.	T_4
R_005	IGN 상태에서 N 단이고, 속도가 0 인 상태에서 SSB 버튼을 누르면 ACC 로 전원이 이동한다.	T_5
R_006 (변경전)	브레이크 스위치 단선시 스마트키가 차량 내에 있을 경우 P 단에서 SSB 버튼을 20 초간 누르면 시동이 켜진다	
R_006	브레이크 스위치 단선시 스마트키가 차량 내에 있을 경우 P 단에서 SSB 버튼을 10 초간 누르면 시동이 켜진다.	T_6
R_007	스마트키가 차량 내에 있고 IGN 상태이며 P 단 위치에서 브레이크를 밟고 SSB 버튼을 누르면 시동이 켜진다.	T_7
R_008	스마트키가 차량 내에 있고 ACC 상태이며 P 단 위치에서 브레이크를 밟고 SSB 버튼을 누르면 시동이 켜진다.	T_8
R_009	주행 중에는 SSB 버튼을 3 번이상 누르면 시동이 꺼지면서 ACC 상태가 된다.	T_9
R_010	주행 중에는 SSB 버튼을 2 초동안 길게 누르면 시동이 꺼지면서 ACC 상태가 된다.	T_{10}
R_011	시동이 켜진 경우 P 단에서 브레이크를 밟고 SSB 버튼을 누르면 시동이 꺼진다.	T_{11}
R_012	시동이 켜진 경우 P 단에서 SSB 버튼을 누르면 시동이 꺼진다.	T_{12}
R_013	주행 중에는 Engine Running 상태가 된다.	T_{13}
R_014	스마트키가 차량 내에 있을 경우 브레이크 페달을 밟고 SSB 버튼을 누르면 ESCL 이 Unlock 되고, 시동이 켜진다.	T_{14}

자동차 기능안전표준 ISO26262[8]의 HARA(Hazard Analysis and Risk Assessment)를 통한 버튼 엔진 시동과 관련한 위험요소(Hazardous Event)는 표 3 이며, 해당 요구사항은 R_006, R_009, R_010 이다.

표 3 위험요소

Hazard	브레이크 스위치 단선
Operational Situation	차량에 시동을 걸 때
Hazardous Event	차량에 시동을 걸 때 브레이크 스위치 단선으로 인해 시동이 걸리지 않는다.
Harm	운전자의 불편함을 초래
Hazard	악셀 페달 리턴 불량
Operational Situation	차량 충돌 사고 발생시
Hazardous Event	차량 충돌 사고 발생시 악셀 페달 리턴 불량으로 인해 차량이 멈추지 않아 P 단으로 이동할 수 없기 때문에 시동을 끌 수 없다.
Harm	인명 피해를 일으키는 사고 발생

식별된 위험 요소의 심각도(Severity)와 노출 가능성(Probability of Exposure), 제어성(Controllability)에 따라 파악된 ASIL 등급은 표 4 이다.

표 4 ASIL 등급

요구사항	심각도	노출가능성	제어성	ASIL
R_006	S2	E4	C3	C
R_009	S3	E4	C3	D
R_010	S3	E4	C3	D

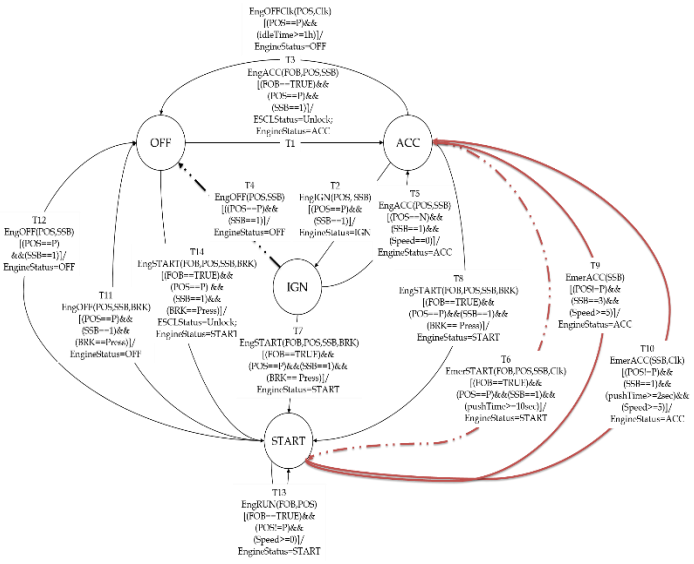


그림 2 기어 P 버튼 엔진 시동 관련 상태도

(1) 요구사항 변경과 관련된 전이 식별

기어 P 단 버튼 엔진 시동 요구사항에 대한 상태도는 그림 2 이며, 표 2 의 요구사항 변경과 관련한 전이는 점선인 T4 와 T6 이다.

(2) 안전 무결성 SIL 기준과 관련된 전이 식별

표 5 에 관련하여 파악된 전이는 그림 2 의 굵은 선인 T_6, T_9, T_{10} 이다.

(3) 테스트 케이스 우선 순위 식별

요구사항 변경과 관련된 전이 MT = {T_4, T_6}이며, 안전과 관련된 전이 ST = {T_6, T_9, T_{10}} 이다. 그림 2 상태도의 전이 커버리지를 만족하도록 생성한 테스트 케이스는 총 15 개 $\{t_1, \ldots, t_{15}\}$이다. 각 테스트 케이스 t 에 대한 가중치 W(t)는 표 5 와 같다. 예를 들어 t_3 의 경우, MT 는 T_6 로써 표 4 에 따라 가중치는 1 이다. ST 는 T_6 인데, 표 4 에 따르면 ASIL C 로써 가중치는 3 이다. 따라서 W(t_3)는 이 두 경우 합인 4 이다.

결과적으로 W(t)에 따라 결정된 테스트 케이스 우선 순위는 < t_9, t_{10}, t_{14}, t_3, t_{15}, t_4, t_5, t_2, t_{11}, t_1, t_6, t_7, t_8, t_{12}, t_{13} >이다.

표 5 테스트 케이스별 W(t)

TC t	MT	ST	W(t)	우선순위	TC t	MT	ST	W(t)	우선순위
t_1			0	10	t_9	T_6	T_6, T_9	7	1
t_2	T_4		1	8	t_{10}	T_6	T_6, T_9	7	1
t_3	T_6	T_6	4	4	t_{11}	T_4		1	8
t_4		T_9	3	6	t_{12}			0	10
t_5		T_{10}	3	6	t_{13}			0	10
t_6			0	10	t_{14}	T_6	T_6, T_{10}	7	1
t_7			0	10	t_{15}	T_4	T_{10}	4	4
t_8			0	10					

표 6 적용 실험 결과

P	10	8	4	6	7	11	12	13	1	2	9	14	15	3	5
#1	8	1	2	9	10	11	12	13	3	4	5	14	15	6	7
#2	8	1	2	9	10	11	12	13	3	4	5	14	15	6	7
#3	8	1	2	9	10	11	12	13	3	5	4	14	15	7	6
TC	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}

MT 만 해당 TC | ST 만 해당 TC | MT, ST 해당 TC

P: 제안하는 우선순위 기법
#1: 발견적 테스트 케이스 우선순위 #1
#2: 발견적 테스트 케이스 우선순위 #2
#3: 발견적 테스트 케이스 우선순위 #3

적용 사례인 BES 시스템을 관련 연구에서 기술한 기법들로 테스트 케이스 우선순위를 측정하였으며, 그 결과는 표 6 과 같다. 각 기법 별로 테스트 우선순위를 나타낸다.

안전 사항이 변경된 요구사항에 결함이 있을 경우 가장 치명적이라고 판단될 수 있다. 표 6 에서 ST 와 MT 에 모두 해당되는 테스트케이스(TC)는 t_9, t_{10}, t_{14}, t_{15} 로써, 제안하는 기법(P)이 가장 결함 검출율이 높다고 알려진 발견적 테스트 케이스 우선순위 #3 을 포함 비교대상 모든 기법보다 이와 관련한 테스트 케이스를 우선 선정할 확률이 다소 높다는 것을 보여준다.

본 사례연구의 대상 실험 사이즈가 적어서 이 결과를 일반화하기에는 무리가 있으나, 제안하는 기법의 특징을 보여주는 사례로써는 의미가 있다.

5. 결론

본 논문에서 안전 무결성 기준을 가중치로 반영한 테스트 케이스 우선순위 기법을 제안하였다. 이 기법은 안전 기능의 우선 검증을 통해 안전 관련 결함을 조기에 찾아 수정함으로써 안전 중심 임베디드 시스템의 안전 위험을 낮추는데 기여한다.

현재 다양하고 심도 있는 적용 결과를 토대로 안전 관련 결함 검출 기여도에 관한 실험 연구를 수행하고 있다.

"본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성 지원사업의 연구결과로 수행되었음" (IITP-2015-H8501-15-1012)"

참고문헌

[1] Bogdan Korel, Luay H. Tahat, Mark Harman, "Test Prioritization Using System Models", Software Maintenance, pp.26-29, 2005

[2] Cagatay Catal, Deepti Mishra, "Test case prioritization: a systematic mapping study", Software Quality Journal, Vol.21, No.3, pp.445-478, 2013

[3] IEC 61508, "Functional Safety of Electrical/Electronic/Programmable Electronic Safety related Systems (E/E/PE), 2010

[4] Bogdan Korel, George Koutsogiannakis, Luay H. Tahat, "Application of System Models in Regression Test Suite Prioritization", Software Maintenance, pp.247-256, 2008

[5] Nida GOKCE, Fevzi BELLI, Mubariz EMINLI, Bekir Taner DINCER, "Model-based test case prioritization using cluster analysis: a soft-computing approach", Turk J Elec Eng & Comp, pp.623-640, 2015

[6] Nida GOKCE, Fevzi BELLI, Mubariz EMINLI, Bekir Taner DINCER, "Revealing influence of model structure and test case profile on the prioritization of test cases in the context of model-based testing", Journal of Software Engineering Research and Development, 2015

[7] http://pd1evl.nl/schema-kia/SSB_Logica.pdf

[8] ISO26262, ISO TC22/SC3/WG16, 2011

확장된 라운드 트립 기법을 이용한 발사통제장치 테스트 기법

배정호, 안세준, 장부철, 구봉주

국방과학연구소 제 1 기술연구본부
{deawith, sejoon86, bucheol.jang, tgkoo00}@gmail.com

요약: 발사통제장치는 국방, 항공 분야에서 무장을 발사하기 위하여 상태를 점검하고 정상 발사를 위한 절차를 진행하는 역할을 수행한다. 발사통제장치와 같이 높은 신뢰성이 요구되는 복잡한 반응성 (reactive) 임베디드 시스템에서는 UML 상태 기계을 활용한 모델 기반 테스트의 적용이 적합하다. 기존의 모델 기반 테스트 케이스 생성 기법은 라운드 트립 기법(RTP)이 비용대비 효과성이 우수한 것으로 알려져 있다. 하지만 해당 기법 기법으로 생성한 테스트 케이스는 최종 임무까지 테스트를 진행하지 않기 때문에 발사통제장치에 적용하기에는 미흡하다. 따라서 본 연구에서는 기존의 RTP 기법을 확장하여 최종 임무까지 수행하는 확장된 RTP 기법을 제안한다.

핵심어: 발사통제장치, 모델기반 테스트, 라운드 트립

1. 서론

발사통제장치는 발사 대상을 발사하기 전 상태를 점검하고 정상 발사를 위한 절차를 진행하는 역할을 수행한다. 발사 대상마다 특징 및 점검 요소가 다르므로 정해진 절차에 따라 점검을 수행하며 비정상 상황이 확인되면 발사를 중지해야 한다. 발사통제장치에 문제가 발생하여 비정상적인 대상이 발사되는 경우에는 막심한 인적, 물적 피해가 발생하기 때문에 높은 신뢰성이 요구된다.

발사통제장치와 같은 반응형 (reactive) 임베디드 시스템은 UML [1] 상태 기계(State Machine)를 이용한 모델 기반의 테스트가 적합하다. 이와 같은 시스템의 체계적인 테스트를 위하여 다수의 모델 기반의 테스트 생성 연구가 진행되었다[2]. 모델 기반 테스트 케이스 생성은 요구사항을 기반으로 모델을 만들고 이를 이용하여 특정 조건을 만족하는 테스트 케이스를 생성하는 기법이다. 일반적으로 서술 형태로 기술되어있는 요구사항을 정형적이고 가독성과 유지보수성이 높은 모델을 이용하여 기술함으로써 고품질의 테스트 케이스를 적은 비용으로 생성할 수 있도록 도와준다.

본 논문에서는 발사통제장치를 대상으로 모델 기반 테스트를 수행한 결과를 보여준다. 발사통제장치의 행위를 표현하는 모델은 UML 의 상태 기계를 사용하였고, 테스트 케이스를 생성하는 기법으로는 가장 성능이 우수하다고 알려져 있는 라운트 트립 기법(round-trip path)[3, 4]을 기반으로 발사통제장치를 테스트하기 적합하도록 확장한 기법을 사용하였다.

본 논문의 구성은 다음과 같다. 2 장에서는 연구 배경으로 기존의 테스트 방법과 UML 상태 기계, 모델기반 테스트 생성 기법을 설명하고, 3 장에서는 구체적인 테스트 모델 생성과 테스트 케이스 생성 절차를 소개한다. 4 장에서 결론 및 향후 연구를 기술한다.

2. 기본 개념

2.1 UML 상태기계

본 연구에서는 발사통제장치의 행위를 나타내는 테스트 모델로서 UML 상태 기계(State Machine)를 사용한다. UML 상태 기계의 정의는 다음과 같다.

정의 1. 컴포넌트의 동적 행위는 상태 기계 $SM = (S, s_0, S_\Psi, I, O, \sigma)$로 표현된다.

- S는 공집합이 아닌 상태의 집합이다.
- s_0는 초기 상태로 S의 원소이다.
- S_Ψ는 최종상태로 S의 부분 집합이다.
- I는 입력 이벤트의 집합이다.
- O는 출력 이벤트의 집합이다.
- σ는 $S \quad I \rightarrow O \quad S$ 로, 두 상태 간의 전이 함수이다. O는 생략 가능하다.

그림 1 은 상태 기계에 대한 간단한 예를 보여준다. 이 예의 상태 기계에는 2 개의 상태와 하나의 전이가 있다. 상태 집합 S = {*IDLE*, *PWR_ON*}이고, 전이 집합 σ = {(*IDLE*, *PwrOn*, *wating*(20), PWR_*ON*)} 이다. 상태는 모서리가 둥근 사각형으로 표현하며, 전이는

상태들 간의 화살표로 표현한다. 전이의 화살표 위에는 (입력 / 출력) 의 형태로 전이가 천이되기 위한 입력과 그에 따른 출력을 표현하며, 입력은 하나 이상이 될 수 있고 출력은 생략할 수 있다. 초기 상태는 작은 점이 가리키는 상태 *s0* = *IDLE* 이고, 최종 상태는 이중 원으로 향하는 상태 집합 S_Ψ = {*PWR_ON*} 이다. 이 컴포넌트가 처리할 수 있는 입력은 *I* = {*PwrOn*}이고 수행할 수 있는 출력은 *O* = {*wating*(20)}이다. *IDLE* 상태에서 받아들일 수 있는 입력은 *PwrOn* 이고, *PwrOn* 이 입력되면 20 초가 대기(*wating*(20))한 후에 *PWR_ON* 상태가 되고 종료된다.

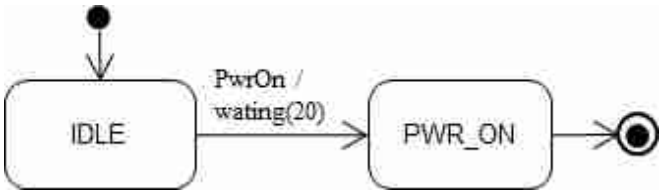


그림 1 UML 상태 기계 예

2.1 기존 테스트 케이스 생성 기법

상태 기반 테스트(state-based testing: SBT)는 상태 기계 다이어그램으로부터 테스트 케이스들을 도출하고, 각 테스트 케이스를 수행함으로써, 소프트웨어의 오류를 검출하는 것이다. SBT 의 대표적인 테스트 케이스 생성 기준에는 모든 전이 커버리지 (AT), 모든 전이쌍 커버리지 (ATP), 모든 라운드 트립 커버리지 (RTP)등이 있다 [5].

RTP 의 기준은 모든 라운드 트립 패스 (round-trip path)를 수행하는 것이다. 라운드 트립 패스는 시작 상태와 끝 상태가 동일하며, 시작/끝 상태를 제외하고는 중복되는 상태가 없는 패스를 의미한다. 이 기준을 만족하는 테스트 케이스 세트를 생성하는 방법으로는 깊이 우선 (depth-first) 혹은 너비 우선 (breadth-first) 탐색 알고리즘을 사용하여 전이 트리 (transition tree)를 만드는 것이다. 이때 초기상태에서 기 방문한 상태를 만날 때까지 테스트를 수행함으로써 RTP 를 만족하는 테스트 케이스를 생성할 수 있다. 이 전이 트리에서 각 패스가 테스트 케이스가 된다.

비용은 적게 들되, 오류 검출은 많이 하는 테스트 케이스 세트가 좋은 것이라 할 수 있는데, 준비시간, 실행시간, 테스트 크기, 뮤테이션 점수를 기준으로 AT, RTP, ATP 등을 평가했을 때, RTP 가 가장 우수하다고 알려져 있다 [7].

발사통제시스템은 최종임무를 끝까지 수행 가능한지를 확인하는 것이 중요한데, 기존 기준들은 최종상태까지 진행되지 않을 수 있으므로 이를 확인하기 힘들 수 있다. 본 연구에서는 이러한 한계를 극복하기 위하여 RTP 를 확장하는 기법을 제안한다. 본 연구에서 제안하는 방법은 RTP 를 기반으로 전이 트리를 만들되, 각 패스에서 기 방문 상태에서 끝나는 것이 아니라, 발사절차 소프트웨어의 최종상태까지의 패스를 추가하는 방법이다.

3. 모델 기반 발사통제장치 테스트 케이스 생성

이 장에서는 발사통제장치를 테스트하기 위하여 테스트 모델을 생성하고 이를 이용하여 테스트 케이스를 생성하는 절차를 설명한다.

3.1 모델 기반 발사통제장치 테스트 케이스 생성 절차

모델 기반의 발사통제장치 테스트 케이스 생성은 테스트 모델 생성, 테스트 케이스 생성의 두 단계로 수행된다.

1) 테스트 모델 생성: 발사통제장치의 요구사항에 따라 상태 기계를 생성한다. 본 연구에서는 컴포넌트를 대상으로 상태 기계를 생성하였다.
2) 테스트 케이스 생성: 1)에서 생성한 모델로부터 RTP 를 확장한 제안하는 방법을 이용하여 테스트 케이스를 생성한다.

3.2 테스트 모델 생성

본 논문에서는 장비 상태 관리, 발사 리스트 관리, 발사제어 등 발사통제장치의 다양한 기능 중 핵심 기능인 발사제어 컴포넌트의 간소화한 기능을 대상으로 예를 설명한다. 그림 2 는 간소화된 발사제어 컴포넌트의 테스트 모델을 보여준다. 이 예의 컴포넌트는 6 개의 입력 *I* = {*PwrOn*, *PwrOff*, *StartCntDn*, *FireInhibit*, *Fired*, *Failed*}를 받을 수 있으며 그에 따라 5 개의 상태 S = {*IDLE*, *PWRON*, *COUNT_DOWN*, *FIRED*, *ABORTED*}로 구성된다. 초기 상태인 *IDLE* 상태에서 *PwrOn* → *StartCntDn* → *Fired* 를 입력함으로써 정상발사를 수행할 수 있으며 이 과정에서 *PwrOff*, *FireInhibit*, *Failed* 의 입력을 통해 상태를 제어할 수 있다.

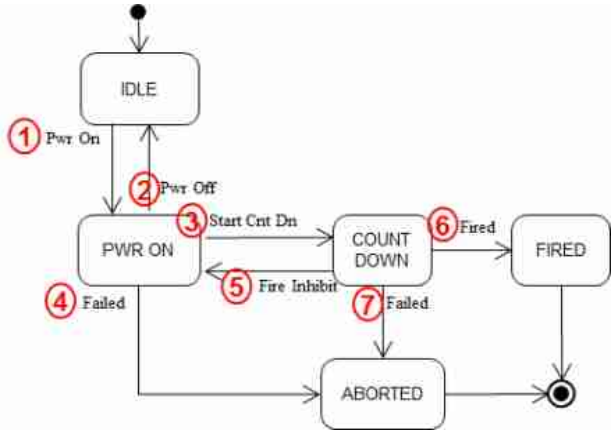


그림 2 발사통제컴포넌트 상태기계 예

3.3 테스트 케스트 생성

이 절에서는 본 연구에서 제안하는 테스트 케이스 생성 기법을 기존의 테스트 케이스 생성 기법인 RTP 와 비교하여 설명한다.

3.3.1 기존 테스트 케이스 생성 기법

본 연구에서는 기존의 RTP 기법을 확장한 테스트 케이스 생성 기법을 제안한다. RTP 는 상태 기계를 이용하여 너비 또는 깊이 우선 탐색을 통해 트리를 구성하여 테스트 케이스를 생성한다. 이때 트리의 단말 노드는 최종 상태 또는 해당 줄기의 기 방문 상태가 된다.

그림 3 은 RTP 를 이용한 간소화된 발사제어 컴포넌트 의 테스트 케이스 생성 예를 보여준다. 그림 2 의 테스트 모델의 초기 상태인 *IDLE* 에서 가능한 입력인 *PwrOn* 이 입력되면 *PWR_ON* 상태가 된다. *PWR_ON* 상태에서는 총 3 개의 입력 *PwrOff*, *StartCntDn*, *Failed* 가 입력될 수 있다. 이 중 *PwrOff* 가 입력되면 *IDLE* 상태가 된다. *IDLE* 상태는 초기 상태로 기 방문 상태이므로 단말 노드가 된다. *Failed* 가 입력되면 *ABORTED* 상태가 되고 이 상태는 최종 상태이므로 단말 노드가 된다. *StartCntDn* 이 입력되면 *COUNT_DOWN* 상태가 되고 이 상태에서 테스트 케이스 생성을 계속한다. 이러한 방법을 통하여 전이 ②, ④, ⑤, ⑥, ⑦의 목표 상태를 단말 노드로 하는 5 개의 테스트 케이스가 생성된다.

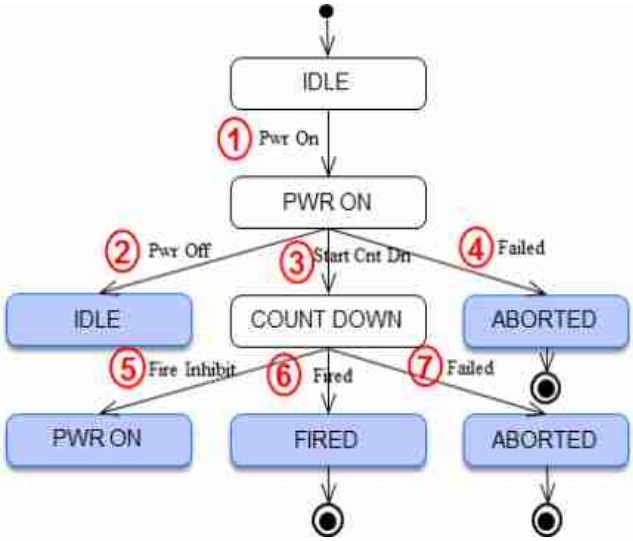


그림 3 기존의 라운드 트립 기법으로 생성한 테스트 케이스

3.3.2 테스트 케이스 확장

기존의 RTP 를 이용하여 생성한 테스트 케이스의 단점은 관찰가능성(observability)이 낮다는 것이다. 발사통제 시스템과 같은 최종 임무 수행이 중요한 시스템에서는 해당 시스템의 역할을 최종적으로 수행하여 최종 상태까지 진행되는지 확인하는 것이 필수적이다. 하지만 기존 RTP 를 이용한다면, 기 방문된 상태에 도착하면 테스트를 종료하기 때문에 임무를 정상적으로 완료할 수 있는지 확인하는 것이 힘들다. 예를 들어, 그림 3 에서 ②의 테스트 케이스를 실행한다면 *IDLE* 상태에서 테스트가 종료된다. 이러한 경우에는 전원을 켜고 중간에 끈 후, 다시 전원을 켠 경우에 최종 발사 임무를 완료할 수 있는지 확인할 수 없다.

따라서 본 연구에는 기 방문한 상태에서 테스트가 종료되는 것이 아니라 그 상태에서 최종 상태까지 테스트가 진행되도록 기존의 RTP 를 확장한 기법을 제안한다. 그림 4 는 기존의 RTP 를 확장하여 테스트 케이스를 생성한 예를 보여준다. 그림 3 에서 ②와 ⑤까지 진행되는 테스트 케이스는 기 방문 상태에서 종료한다. 본 연구에서 제안한 기법은 이 두 상태에서 최종 상태까지 진행되도록 트리를 확장한다. 이때, 최종 상태로 도달하는 다양한 방법 중 정상발사를 위한 입력을 수행하도록 선택하였다.

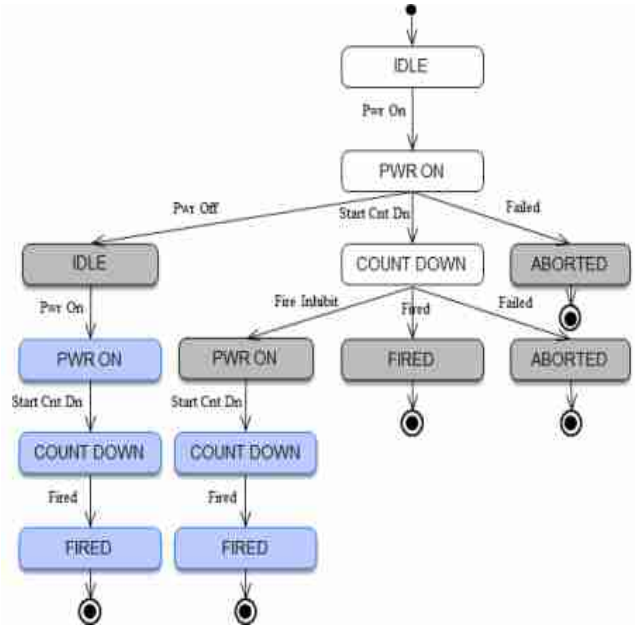


그림 4 확장된 RTP 기법으로 생성한 테스트 케이스

본 연구에서 제안하는 기법은 기존의 테스트 케이스를 확장하는 방법이기 때문에 기존의 기법보다 수행해야하는 테스트의 양이 같거나 많다. 기존의 RTP 방법(그림 3)에 비해서 제안한 확장된 RTP 방법으로 생성한 테스트 케이스(그림 4)의 수는 5 개로 동일하다 하지만, 그림 3 에서 ②와 ⑤의 테스트 케이스가 확장되었기 때문에 테스트 전체의 입력 수는 5 번 증가하였고 그에 따라 테스트 케이스 당 평균 입력의 수는 2.6 번에서 3.6 번으로 증가되었다.

발사통제장치와 같은 높은 신뢰성이 요구되는 시스템에서는 테스트의 양 보다는 보다 많은 오류의 검출이 중요하다. 그림 3 의 ⑤와 같이 (초읽기시작-발사중지)에서 테스트가 끝나는 것이 아니라 발사중지 후 정상발사가 진행되는지를 끝까지 확인해보는 것이 신뢰성을 높이는데 도움이 될 것이다. 따라서 우리는, 비록 제안한 방법이 테스트의 양을 증가시키지만 확장된 테스트 케이스를 통해 검출할 수 있는 오류의 양도 증가할 것이라고 기대한다.

4. 결 론

발사통제시스템과 같은 국방, 항공 분야의 시스템은 높은 신뢰성이 요구된다. 이와 같은 반응형 임베디드 시스템은 상태 기계를 이용한 모델기반 테스트 기법이 유용하게 적용될 수 있다. 기존의 모델기반 테스트 케이스 생성 기법 중 가장 효용성이 높다고 알려져 있는 기법은 RTP 기법이다. 하지만 기존 RTP 기법은 최종 임무 단계까지 테스트를 수행하지 않으므로 높은 신뢰성을 만족하는데 한계가 있다. 따라서 본 연구에서는 기존의 RTP 를 최종 임무까지 수행하도록 확장된 RTP 기법을 제안하였다.

향후에는 본 기법을 적용하여 실제 발사통제장비의 테스트를 수행할 계획이다. 이를 통하여 본 연구에서 제안한 기법으로 생성한 테스트 케이스가 더욱 많은 오류를 검출하는지를 확인하고 기존의 기법과 비교하여 효율성을 확인할 것이다.

참고문헌

[1] UML, [Online]. Available: http://www.omg.org/spec/UML

[2] A.C. Dias Neto, R. Subramanyan, M. Vieira, and G.H. Travassos, "A Survey on Model-based Testing Approaches: a Systematic Revie," *Proc. ACM Int'l. workshop on Empirical Assessment of Sotware Eng. Languages and Technologies*, pp. 31-36, 2007

[3] G. Antoniol, L.C. Briand, M.D. Penta, and Y. Labiche, "A Case Study Using the Round-Trip Strategy for State-Based Class Testing", *Proc. 13th Int'l. Symp. IEEE Software Reliability Eng.* , pp. 269-279, 2002.

[4] L.C. Briand, M.D. Penta, and Y. Labiche. "Assessing and Improving State-based Class Testing: A Series of Experiments". *IEEE Trans. Software Eng.*, vol. 30, no. 11, pp. 770-783 , 2004.

[5] N.E. Holt, L.C. Briand, and R. Torkar, "Empirical evaluations on the cost-effectiveness of state-based testing: An industrial case study," *Information and Software Technology*, vol. 56, no. 8, pp. 890-910, 2014

[6] R. Binder, *Testing Object-Oriented Systems*, Addison-Wesley, 2000.

[7] L. Briand, Y. Labiche, Y. Wang, "Using Simulation to Empirically Investigate Test Coverage Criteria based on Statechart," *Proc. 26th International Conference on Software Eng.*, pp.86-95, 2004.

안드로이드 어플리케이션 개발 생산성을 위한 권한 자동 검사 기법

(An Automated Inspection of Permission for Productivity Improvement in Android Development)

노승학, 인호

고려대학교 컴퓨터학과
서울 성북구 안암로 145
{blue619, hoh_in}@korea.ac.kr

요약: 안드로이드 개발에 있어서 권한 설정은 어플리케이션의 보안 및 생산성에 있어 중요한 부분을 차지한다. 그러나 대부분의 초보 개발자들은 앱을 개발할 때 어떤 코드에 어떤 권한이 사용되는지 모르는 경우가 많다. 초보 개발자가 아니더라도 권한의 종류가 매우 다양하기 때문에 API 별 필요 권한을 외우고 있는 개발자는 거의 없다. 본 논문에서는 안드로이드 어플리케이션의 개발단계에서 어떤 권한이 사용되어야하는지 또는 불필요한지 판단하는 권한 자동 검사 기법을 제안한다.

핵심어: 안드로이드, 어플리케이션, 권한, 생산성, 정적 분석

1. 서론

안드로이드 어플리케이션을 개발함에 있어서 권한 설정은 필수적인 과정이다[1]. 안드로이드 시스템상의 중요한 시스템 리소스 및 API 에 접근할 때 안드로이드 OS 는 해당하는 권한을 체크하게 되고 만약 어플리케이션에 해당하는 권한이 없다면 런타임 오류(Runtime Error)를 출력한 뒤 강제 종료하게 된다. 많은 초보 개발자들은 어떤 API 에 어떤 권한이 요구되는지 모르는 경우가 많다. 아직까지 Stackoverflow 등 개발자 커뮤니티에 안드로이드 권한 관련 이슈가 종종 올라오는 것이 이를 뒷받침한다[2]. 그러나 해당 API 에 권한이 필요한가에 대한 여부는 개발단계에서 따로 문서를 확인하지 않는 이상 확인 할 방법이 없으며 빌드 단계에서 조차 오류를 발생시키지 않아 런타임 오류를 통해 확인할 수밖에 없다. 이는 개발단계에서의 생산성에 악영향을 주고 권한이 필요한 API 를 사용할 때마다 인터넷을 찾아보거나 문서를 확인해야하는 불편함이 있다.

더불어 안드로이드 M(Marshmallow) 버젼이 릴리즈되면서 사용자들이 실행환경에서 선택적으로 권한을 선택 할 수 있는 기능이 생겼지만 구글에서는 여전히 개발자들의 올바른 권한 처리를 핵심 개발과정의 하나로 규정하고 있다[6]. 권한이 안드로이드의 핵심 보안 정책으로 자리매김하는 이상 개발자들에게 권한의 올바른 설정 및 최적화는 개발 단계에서 꼭 확인하고 넘어가야하는 이슈이다.

본 논문에서는 개발단계에서 어떤 API 에 어떤 권한이 사용되어야 하는지, 현재 불필요한 권한이 사용되고 있는지를 API-권한 맵(Map)을 이용해 자동으로 검사하는 정적 분석에 기반한 안드로이드 권한 자동 검사 기법을 제안한다. 이를 통해 안드로이드 어플리케이션 개발 시 인터넷 검색과 같은 불필요한 요소를 배제하여 생산성을 향상시키는 것을 목표로 한다.

2. 연구 배경

안드로이드 보안정책 중 하나인 응용 어플리케이션 권한 정책은 타 스마트폰 OS 의 보안 모델과 비슷한 개념으로 특정 응용 프로그램이 내부의 다양한 데이터, 하드웨어 장치 또는 다른 응용프로그램의 컴포넌트에 접근하려 할 때 적절한 권한(Permission)을 획득해야만 이를 허용하도록 하는 개념이다[3]. 이 권한은 안드로이드 어플리케이션을 개발하는 단계에서 AndroidManifest.xml 이라는 설정 파일에 그림 1 과 같이 <uses-permission>태그를 이용해 정의하며 어플리케이션 사용자에겐 설치단계에서 어플리케이션이 요구하는 권한을 고지한다.

이러한 안드로이드 권한을 소프트웨어 공학적인 측면으로 접근하여 안드로이드 시스템 API 들과 권한 간의 관계를 매핑(Mapping)하려는 기존 연구로는 안드로이드 공식 문서를 분석하여 매핑하는 방안[4]과

안드로이드 소스코드를 정적 분석하여 매핑하는 방안[5] 등이 있다.

그러나 [4]의 방법은 기존 안드로이드 공식 문서가 모든 API 를 문서화하지 않기 때문에 적용범위가 불완전하다고 볼 수 있다[5]. 따라서 본 연구에서는 [5]의 연구에서 개발된 PScout 이라는 안드로이드 소스 코드 정적 분석 툴을 통해 안드로이드 시스템 API 와 권한 관계를 매핑하여 모델링에 활용한다.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="esel.tokirin.lenvest" >

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WHITE_EXTERNAL_STORAGE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
```

그림 1. AndroidManifest.xml 예

표 1. 기존 연구와 제안 연구의 특장점 비교

비교 항목	기존 연구[4]	제안 연구
API-권한 맵의 불완전성	불완전한 문서 기반 매핑	소스코드 정적 분석 기반 매핑
생산성 측면	불필요한 권한만 제시	개발 단계에서 누락된 권한 및 불필요한 권한 모두 제시

표 1 은 기존 연구와 본 연구의 특장점을 비교하여 나타낸다. [4]의 연구 결과를 보면 불필요한 권한을 감지하는 것에서 그쳤고 개발자에게 누락된 권한을 제시하지 못한다는 점에서 생산성에 향상에 기여하지 못한다는 단점이 있다. 또한 불완전하게 명세된 안드로이드 문서를 사용한다는 점에 있어서 API-권한 맵의 신뢰성 또한 떨어진다는 문제도 존재한다.

3. 안드로이드 권한 자동 검사 기법

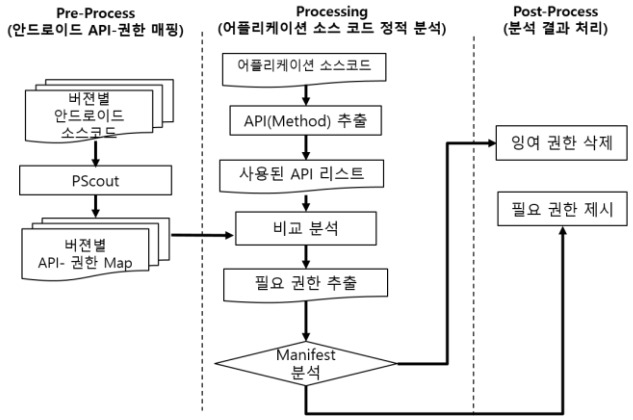


그림 2. 시스템 구조도

본 연구에서 제안하는 기법은 그림 2 와 같은 과정을 거쳐 타겟 어플리케이션의 필요 권한을 추출하여 현재 소스코드에서 필요한 권한을 제시하고 불필요한 권한을 삭제하도록 유도한다. 상세한 과정은 다음과 같이 크게 세 부분으로 나누어 설명한다.

3.1 안드로이드 API-권한 매핑

타겟 어플리케이션의 개발환경에 맞는 안드로이드 플랫폼 버전의 소스 코드를 PScout[5]을 통해 분석하여 API-권한 맵을 추출한다. PScout 은 안드로이드 플랫폼의 소스 코드를 분석하여 안드로이드 시스템 API 가 어떤 권한을 요구하는지에 대한 분석을 수행한다.

```
Permission:android.permission.CHANGE_WIFI_STATE
17 Callers:
<android.net.wifi.WifiManager: boolean reassociate()>
<android.net.wifi.WifiManager: boolean startScan()>
<android.net.wifi.WifiManager: void setCountryCode(java.lang.
<android.net.ConnectivityManager: int startUsingNetworkFeatur
<android.net.wifi.WifiManager: boolean disableNetwork(int)>
```

그림 3. 추출된 API-권한 맵 예

추출된 API-권한 맵은 그림 3 과 같이 텍스트 파일 형태로 저장되고 임의의 Key-Value 형태의 자료구조에 Key 는 API 메서드명으로, Value 는 권한명으로 저장될 수 있다.

$$\mathcal{F}: M \rightarrow P \qquad (1)$$

이는 곧 수식 1 의 함수 $\mathcal{F}$ 로 표현할 수 있다. 여기서 M 은 안드로이드 시스템 API 메서드 들의 집합이며 P 는 안드로이드 시스템 권한들의 집합이다.

3.2 어플리케이션 소스 코드 정적 분석

본 연구에서 제안하는 기법은 분석할 안드로이드 어플리케이션의 소스코드를 입력으로 받아 정적 분석을 수행하여 소스코드에서 필요한 권한들을 추출한다. 해당 내용을 수행하기 위하여 필요한 권한 집합들을 간단하게 수식으로 모델링하였다. 아래의 수식 2, 3, 4 는 각각 필요 권한 집합, 불필요한 권한 집합, 누락된 권한 집합을 의미한다.

$$P_r = \mathcal{F}(M_a) \qquad (2)$$

$$P_u = P_m \cap (P_r)^c \qquad (3)$$

$$P_n = (P_m)^c \cap P_r \qquad (4)$$

먼저 3.1 의 과정을 통해 추출된 API-권한 맵과 수식 2 를 통해 필요 권한 집합 P_r을 추출한다. M_a는 어플리케이션 소스 코드에서 사용되는 안드로이드

API 메서드들의 집합이다. P_r을 이용하면 수식 3과 4를 통해 불필요한 권한 집합 P_u 와 누락된 권한 집합 P_n 을 계산할 수 있다. P_m은 타겟 어플리케이션의 AndroidManifest.xml에 명시되어있는 권한 집합을 의미한다.

3.3 분석 결과 처리

3.2의 과정을 통해 계산된 불필요한 권한과 누락된 권한은 그림 4와 같은 IDE 플러그인 형태로 출력할 수 있다. 불필요한 권한의 경우에는 삭제 권고 메시지를, 누락된 권한의 경우 추가 권고 메시지를 출력하여 개발자들에게 런타임 오류로써 권한에 대한 오류를 다루게 하지 않고 코드를 작성함과 동시에 권한에 대한 통찰력을 부여할 수 있다. 이는 곧 권한에 대한 오류를 디버깅하는 데에 있어 어플리케이션을 빌드하고 실행시키는 시간을 절약하게 해주며 생산성의 향상으로 이어진다.

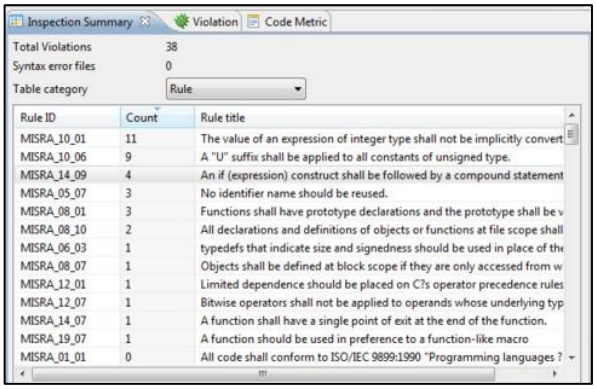


그림 4. 출력 예시

4. 결론 및 후속 연구

아직까지도 많은 개발자들이 안드로이드 개발 시 권한 설정을 잘못해 런타임 오류를 겪는 경우가 많다. 이로 인해 초래되는 인터넷 검색 등과 같은 불필요한 행동이 곧 생산성의 저하로 이어진다. 본 연구에서 제안한 기법이 이 문제를 해결해주리라 생각한다. 본 연구는 기존 연구에 비해 안드로이드 어플리케이션 개발 단계에 있어 권한 설정에 대한 개발자들의 생산성을 제고할 수 있다는 점이 가장 큰 장점이다.

향후 연구로서 본 연구가 개발자들의 생산성에 얼마나 영향을 미치는지에 대한 개발자 대상 설문 조사와 제안한 권한 집합 모델의 정확성과 성능 향상에 대한 연구를 진행할 것이다.

사사

이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보・컴퓨팅기술개발사업의 지원을 받아 수행된 연구임 (2012M3C4A7033345)

참고문헌

[1] Android Developers API Guide, “App Manifest”, http://developer.android.com/intl/ko/guide/topics/manifest/manifest-intro.html

[2] Stackoverflow,“ java.lang.securityexception: permission denied”, http://stackoverflow.com/questions/32823599/java-lang-securityexception-permission-denied-missing-internet-permission-ev, 2015.09

[3] 김익환, 김태현, “안드로이드 플랫폼에서의 유연한 응용프로그램 권한 관리 기법 설계 및 구현”, KIPSTC, 2011

[4] Thomas Vidas 외, “Curbing android permission creep”, Proceedings of the Web 2.0 Security and Privacy 2011 workshop (W2SP 2011), 2011.

[5] Kathy Wain Yee Au 외, “PScout: analyzing the Android permission specification”, CCS '12 Proceedings of the 2012 ACM conference on Computer and communications security, 2012

[6] Jamal Eason, “안드로이드 M 개발자 프리뷰 및 도구”, Google Developers Korea, http://googledevkr.blogspot.kr/2015/06/m.html, 2015.06

안드로이드 이미지 로딩 라이브러리 비기능 품질 속성 비교

조성래
(주)삼성전자
KAIST 소프트웨어 대학원
서울시 강남구 논현로 28 길 25
maxcrom@kaist.ac.kr

정연철
kt ds
KAIST 소프트웨어 대학원
서울시 강남구 논현로 28 길 25
handahan@kaist.ac.kr

민상윤
KAIST 전산학부
KAIST 소프트웨어 대학원
서울시 강남구 논현로 28 길 25
symin@kaist.ac.kr

요약 : 개발자는 안드로이드 어플리케이션에서 이미지 사용 시 다음 사항을 고려해야 한다. 1) 비트맵 객체의 메모리 관리 2) 네트워크 연결 처리 3) 백그라운드 스레드 운영 4) 이미지 캐쉬 사용. 이러한 사항들을 쉽게 구현하기 위해 오픈소스 이미지 로딩 라이브러리들이 존재한다. 또한 Google, Facebook 등 인터넷 기업들도 자신들이 사용하는 이미지 로딩 라이브러리들을 오픈소스로 공개했다. 그리고 오픈소스로 제공된 이미지 라이브러리를 사용하기 전, 각 라이브러리의 품질에 대한 검토가 필요하다. 본 연구에서는 여러 품질 속성 중 DSM(Design Structure Matrix)을 이용한 모듈성(Modularity)과 이미지 로딩 속도를 측정한 성능(Performance)를 선택하여 두 가지 측면으로 나누어 비교 분석해 보고자 한다.

핵심어: Android, Design Structure Matrix, Modularity, Performance, Image Loading

1. 서 론

안드로이드 어플리케이션 개발에서 이미지를 나타내는 작업은 크게 다음과 같이 나눌 수 있다.

1) 이미지 리소스에 대한 요청
2) 이미지 데이터 수신
3) 이미지 데이터 변환
4) 화면의 출력
5) 이미지 캐쉬 관리

이미지 로딩은 복잡하고 고려할 사항이 많은 작업으로 Google 에서는 이미지 로딩에 대한 구현 가이드를 제공한다. 하지만 구현 가이드에서 제공하는 정보만으로 다양한 요구 사항을 만족하는 어플리케이션을 구현하기에 부족하다. 그 이유는 다음과 같다.

첫째, 백그라운드 스레드 구현 시 AsyncTask 클래스를 이용하도록 가이드한다. AsyncTask 는 안드로이드에서 스레드를 편리하게 사용할 수 있게 해준다는 장점이 있다. 반면, 제약 사항으로 UI 스레드에서 수행해야 하고, 플랫폼마다 동작 방식이 다를 수 있다[1]. 둘째, 개발자가 직접 구현해야 하는 항목들이 많다. 로딩 속도를 빠르게 하기 위한 캐쉬 구현, 외부 네트워크 요청 시 발생할 수 있는 예외 사항 처리 및 올바른 이미지 크기 설정 작업을 직접 구현해야 한다.

이러한 구현 가이드의 단점을 보완하고 개발자가 쉽고 빠르게 이미지 로딩을 구현할 수 있도록 다양한 이미지 로딩 라이브러리들이 개발되었다. 최근 Google, Facebook 과 같은 인터넷 기업에서도 자신들이 사용하고 있는 이미지 로딩 라이브러리를 오픈소스로 공개하였다. 그리고 개발자는 많은 이미지 로딩 라이브러리 중 어플리케이션에 적합한 라이브러리를 선택해야한다. 개발자는 라이브러리 선택 시 라이브러리 기능을 기준으로 선택할 수 있다.

하지만 오픈소스로 개발되는 소프트웨어이므로 지속적인 유지보수성이 중요하다. 그리고 사용자가 체감할 수 있는 비기능 품질 속성 중 이미지가 얼마나 빨리 로딩 되는가는 중요한 비기능 품질 속성이다. 따라서 본 연구에서는 여러 비기능 품질 속성 중 유지보수성을 나타낼 수 있는 모듈성(Modularity)와 성능(Performance)을 나타낼 수 있는 이미지 로딩 속도 측정 방법을 제시하고 실제 라이브러리에 적용시켜 결과를 비교 평가해 보고자 한다. 추가적으로 각 라이브러리의 메모리 사용량에 대해서도 측정, 비교한다.

2. 이미지 로딩 시 문제점

각 이미지 로딩 라이브러리들을 분석하기에 앞서 안드로이드에서 이미지 로딩 시 발생할 수 있는 문제점들에 대해서 알아보자.

2.1 메모리 관리

모바일 기기에서 표시되는 이미지는 비트맵 형태로 디코딩되어 메모리에 할당된 뒤 화면에 표시된다. 이 때 메모리에 할당되는 크기는 ARGB_8888 포멧을 사용할 경우 설정한 이미지 크기 * 4 byte 만큼의 메모리 공간을 차지하게 된다[1]. 예를 들어 테스트에 사용된 Galaxy Note 3 모델의 경우, 전체 화면의 해상도는 1920 * 1080 이므로 전체 화면으로 이미지를 로딩할 경우 1920 * 1080 * 4 = 8,294,400 byte(약 8MB)를 차지한다. 이미지가 차지하는 메모리 공간은 고해상도 기기로 갈수록 증가한다. 또한 잘못된 이미지 사이즈 설정도 불필요한 메모리 공간을 차지한다. 이렇게 큰 메모리 사용은 빈번한 Garbage Collection 또는 Out of Memory 을 유발하여 어플리케이션의 성능을 저하시킬 수 있다[2]. 따라서 이미지 객체에 대한 올바른 메모리 관리가 필요하다.

2.2 네트워크 환경

네트워크를 통해서 이미지를 로딩할 경우 개발자는 다양한 예외 상황에 맞는 코드를 작성해야 한다. 모바일 기기에서 네트워크 환경은 무선 네트워크 환경이므로 연결 상태를 보장할 수 없다. 따라서 네트워크가 비정상적인 경우를 대비한 방어 코드를 작성해야 한다.

2.3 백그라운드 작업 처리

Google 에서 제시한 구현 가이드는 네트워크 요청, 이미지 디코딩, 디스크 입출력 등 처리 시간이 많이 소요되는 작업을 위해 AsyncTask 클래스를 사용한다. AsyncTask 는 안드로이드에서 백그라운드 스레드 작업이 필요할 경우 쉽게 스레드를 구현할 수 있는 클래스이다. 하지만 UI 스레드에서만 사용할 수 있다는 제약 사항이 있다. 또한 안드로이드 플랫폼 버전마다 다르게 동작한다[1]. 결과적으로 개발자는 자신의 어플리케이션이 실행될 환경을 고려하여 스레드 코드를 작성해야 한다.

2.4 이미지 캐쉬

출력된 이미지는 재사용시 로딩 속도 향상을 위해 캐쉬를 사용한다. 캐쉬는 메모리 캐쉬와 디스크 캐쉬로 나누어 구현한다. 구현을 위한 방법으로 안드로이드에서는 LruCache 라는 기본 클래스를 제공한다. 하지만 어플리케이션 환경에 맞게 구현하고자 한다면 많은 노력이 필요하며, 모듈화를 하지 않을 경우 중복 구현의 요소가 존재한다.

3. 비기능 품질 속성 선택 기준

2 장에서 언급한 문제점들에 대해 개발자들이 쉽게 사용할 수 있도록 오픈소스로 제공되는 이미지 라이브러리는 표 1 과 같다.

Library	URL
AQuery	https://github.com/androidquery/androidquery/releases/
AUIL	https://github.com/nostra13/Android-Universal-Image-Loader
Driod4me	https://github.com/smartnsoft/droid4me
Fresco	http://frescolib.org
Glide	https://github.com/bumptech/glide
Novoda Image Loader	https://github.com/novoda/image-loader
Picasso	http://square.github.io/picasso/
Volley	https://android.googlesource.com/platform/frameworks/volley/

표 1. 안드로이드 이미지 라이브러리

표 1 과 같이 다양한 라이브러리들이 존재하며 개발자는 자신에게 적합한 라이브러리를 선택하여 사용할 수 있다.

3.1 모듈성

표 1 에서 기술한 라이브러리들은 모두 오픈소스로 제공된다. 따라서 라이브러리 사용 시 발생한 결함 또는 품질 문제에 대한 수정 구현을 직접 해야한다. 그리고 라이브러리의 지속적인 업데이트 및 지원이 중단될 가능성도 존재한다. 결론적으로 라이브러리 사용에 따른 유지보수 책임은 개발자의 몫이므로 라이브러리 선택 시 지속적인 유지보수성을 고려하여야 한다. 이를 위해 다음과 같이 두가지 사항을 생각할 수 있다.

첫째, 개발자가 쉽게 오픈소스를 수정할 수 있는가? 오픈소스는 사용하는 개발자가 직접 개발한 소프트웨어가 아니다. 따라서 특정 기능을 수정, 삭제, 개선하였을 때 side effect 가 발생할 가능성이 있다. 이 가능성을 최소화하기 위해서는 개발자가 쉽게 구조를 파악할 수 있어야 하며, 각

기능 혹은 모듈간 의존성은 낮고 결합도는 높아야 한다.

둘째, 해당 오픈소스 커뮤니티가 참여자들에 의해 활발하게 운영될 수 있는가? 일반적으로 오픈소스는 해당 커뮤니티에 참여하는 사람이 많으면 많을수록 버그 수정과 기능 향상이 빈번하게 이루어진다. 따라서 오픈소스 커뮤니티가 얼마나 활발하게 운영될 수 있는지도 중요한 검토 사항이다. 이에 대해 O' Reilly[3] 은 사용자 참여에 의해 운영되는 커뮤니티의 성장을 위해 높은 모듈성이 필요하다고 말하고 있다. 또한 Baldwin 와 Clark[4]는 모듈화된 오픈 소스 코드가 보다 자발적인 참여를 이끌어 낸다고 주장한다.

따라서 모듈성은 오픈소스 사용 시 검토해야 할 중요한 비기능 품질 속성 중 하나이다.

3.2 이미지 로딩 속도와 메모리 사용량

이미지 처리 작업은 이미지 요청, 캐쉬 검사, 이미지 변환, 캐쉬 저장, 화면 출력 등과 같은 작업으로 이루어져 있다. 이를 Fresco 에서는 그림 1 과 같이 표현한다[2].

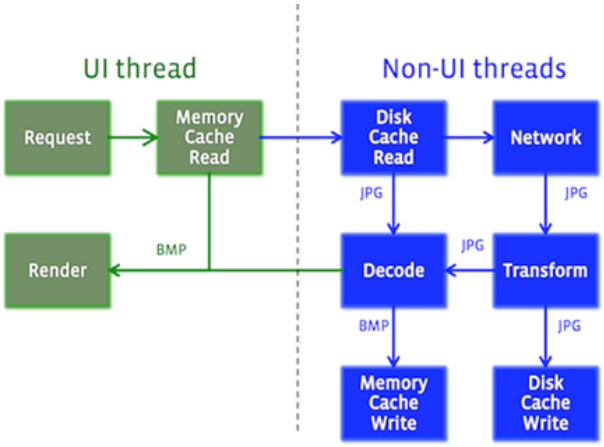


그림 1. 이미지 처리 작업 흐름도

다른 라이브러리들도 구현된 클래스의 이름은 다르지만 동일한 흐름으로 이미지 처리를 한다. 그리고 작업 시간이 많이 소요되는 데이터 요청 및 비트맵 변환 과정, 캐쉬에 저장하는 과정 등은 모두 백그라운드 스레드로 처리한다. 전체적인 이미지 처리 흐름은 유사하지만 라이브러리별로 스레드를 처리하는 방식에 차이가 있다.

Fresco 의 경우 네트워크 요청 스레드, 디스크 입출력에 관련된 스레드(캐쉬 저장), 이미지 디코딩에 관련된 스레드를 별도로 생성하여 수행한다[2]. Picasso 의 경우 RequestCreator 를 통하여 네트워크 처리와 이미지 처리를 분리하여 구현한다. AUIL 의 경우 캐쉬 처리를 위한 스레드와 이미지 처리를 위한 스레드, 그리고 이 두 스레드를 관장하는 스레드를 별도로 구현한다.

이미지가 로딩되어 화면에 나타나는 속도는 사용자가 직접 체감할 수 있는 어플리케이션의 품질 속성 중 하나이다. 화면에 이미지가 빠르게 출력되기 위해서는 백그라운드 스레드들이 빠르게 처리되어야 한다. 그리고 각 라이브러리 별 백그라운드 스레드 운영 방식에는 차이가 있으므로, 이미지 로딩 속도를 측정하여 비교하고자 한다.

또한 비트맵 이미지 객체는 큰 메모리 공간을 차지한다. 복수의 이미지 로딩 시 올바른 메모리 관리가 이루어지지 않을 경우 Out of Memory 현상이 발생할 수 있다. 따라서 메모리 사용량에 대해서도 측정, 비교하고자 한다.

3.3 비교 대상 라이브러리

비교 대상 라이브러리는 Google, Facebook 과 같은 인터넷 기업이 자사의 어플리케이션에 사용하고 있는 Volley, Picasso, Glide, Fresco 를 선정하였다. 그리고 Github 에서 많이 사용되고 있는 AUIL 을 선정하여 비교 분석하고자 한다.

Library	설명	기준 버전
AUIL	Github Android 관련 프로젝트 중 가장 많이 활용되고 있는 오픈소스 라이브러리	1.9.5
Fresco	Facebook 이 2015 년에 공개한 이미지 라이브러리 Facebook 및 Facebook 메신져에 사용[2]	0.7.0
Glide	Google 이 2014 년에 공개한 라이브러리, Bumptech 를 인수하면서 해당 라이브러리 공개, Android Camera 앱과 Google I/O 2014 앱에 사용	3.6.1
Picasso	Square Inc.가 공개한 라이브러리	2.5.2
Volley	Google 이 2013 년 발표한 라이브러리, Google Play Store 어플리케이션에 사용[5]	1.0.19

표 2. 비교 대상 라이브러리

4. 모듈성 분석

4.1 Design Structure Matrices

모듈화가 잘 되어있는지 확인하기 위한 도구로 DSM(Design Structure Matrix)를 이용한다. DSM 은 각 시스템 요소들간의 상호 의존성을 나타낼 수 있는 분석 도구로 Steward[6]에 의해 개발된 도구이다.

따라서 DSM 을 이용할 경우 소프트웨어 시스템 간의 의존성을 한 눈에 쉽게 파악할 수 있다. DSM 은 정방행렬의 형태로 행렬의 각 원소는 시스템 구성 요소간 의존성을 의미한다.

예를 들어 모듈 A 가 모듈 B 에 의존성이 있고 모듈 B 가 모듈 C 에 의존성이 있다면 세 개 모듈에 대한 DSM 은 그림 2 처럼 나타낼 수 있다.

	A	B	C
A			
B	1		
C		1	

그림 2. DSM 예제

4.2 Propagation Cost

DSM 만으로 각 라이브러리 간의 모듈성을 정량적으로 비교하기 어렵다. 따라서 모듈화의 정도를 수치로 표현하기 위하여 MacCormack[7]이 사용한 Propagation Cost 라는 개념을 이용하고자 한다.

Propagation Cost 는 시스템 구성 요소 간 직간접적인 의존성의 정도를 나타내는 값이다. Propagation Cost 를 계산하기 위해 먼저 각 모듈간 간접적인 의존성을 계산한다. 예를 들어 그림 2 에서 모듈 C 가 수정이 되었을 경우 직접적으로 영향을 받는 모듈은 B 이지만 간접적으로 모듈 A 까지 영향을 받을 수 있다. 이를 확인하기 위해서는 DSM 의 행렬을 서로 곱한다. 그림 2 의 예제에 대해 수행하면 그림 3 와 같은 행렬이 계산된다.

	A	B	C
A			
B			
C	1		

그림 3. 2 단계 의존성 표현

즉, DSM 자체는 의존성 경로가 1 일 경우를 의미하고 서로 곱한 행렬은 의존성 경로가 2 일 경우의 의존성 관계를 나타낸다. 따라서 모든 의존성 경로를 계산하기 위해서는 아래와 같은 연산이 필요하다.

$$V = \sum_{i=0}^{N} D^i \, (D^i = D \times D^{i-1}, if\ i > 1)$$

MacCormack[7]이 제시한 Propagation Cost 에서는 단순히 의존성 유무만을 계산한다. 따라서 행렬 V 에 대해 각 원소가 1 이상일 경우에만 값을 1 로 설정한다.

$$V' = f(V) = \begin{cases} v'_{ij} = 0, if\, v_{ij} = 0 \\ v'_{ij} = 1, if\, v_{ij} \geq 1 \end{cases}$$

Propagation Cost 는 행렬 V′에 대해 모든 원소를 더하고 이것을 전체 원소의 개수로 나눈 값이 된다.

$$Propagation\ Cost = \left(\sum_{j=0}^{N} \sum_{i=0}^{N} v'_{ij} \right) / N^2$$

해당 값은 시스템에서 하나의 구성 요소를 수정하였을 때 직접적으로 또는 간접적으로 영향을 받는 모듈의 비율을 나타내는 수치이다[8].

4.3 측정 결과

Propagation Cost 를 이용하여 각 이미지 라이브러리의 모듈성을 측정한 수치는 그림 4 과 같다.

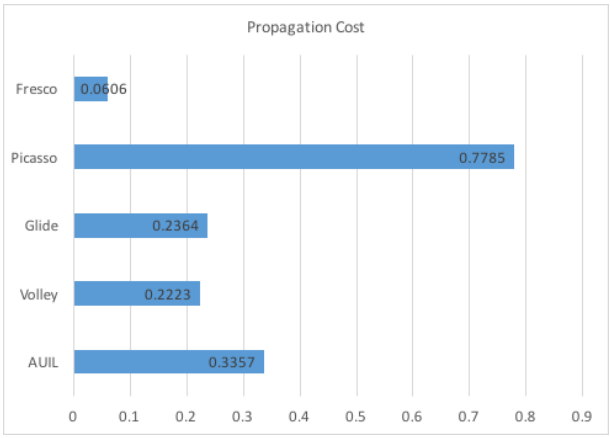


그림 4. Propagation Cost 측정 결과

측정 결과를 보면 Fresco 가 가장 Propagation

Cost 값이 낮다. Glide 와 Volley 은 유사한 수준을 보이며 Picasso 는 가장 높은 값을 가지고 있다. Picasso 의 경우 Picasso, RequestCreator, BitmapHunter, 그리고 Dispatcher 클래스 간 순환 참조와 의존성이 많이 발견되었다. 그리고 이러한 사실은 DSM 으로도 쉽게 확인이 가능하다. Picasso 의 경우 그림 5 에서 처럼 Picasso 클래스를 기준으로 각 클래스간의 의존성이 복잡하게 존재하며, 순환 참조의 모습도 많이 나타남을 알 수 있다.

반대로 Fresco 의 경우 그림 6 에서 보듯이 역호출이나 순환 참조의 모습은 보이지 않는다.

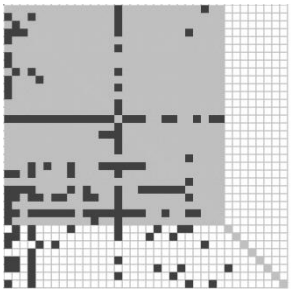


그림 5. Picasso DSM

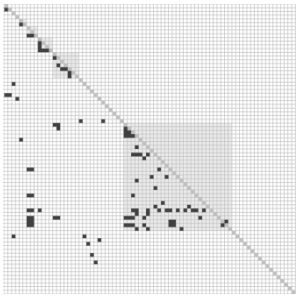


그림 6. Fresco DSM

5. 이미지 로딩 속도와 메모리 사용량 측정

5.1 실험 환경

실험 환경은 표 3 과 같다. 동일한 환경 설정을 위해 메모리 캐시 크기와 비트맵 디코딩 설정은 각각 4MB 와 ARGB_8888 로 고정하였다. 그리고 구현 가이드에서 제공되는 샘플 어플리케이션을 별도 Basic 이라는 카테고리로 분류하여 비교 측정하였다. 이는 가이드에서 제공되는 방법과 라이브러리간의 성능 비교 목적으로 측정하였다.

항 목	사 양
시험장비	Galaxy Note3
안드로이드 버전	4.4.2
프로세스	CPU 속도 : 2.3GHz CPU 종류 : Quad-Core
메모리	RAM Size : 3GB ROM Size : 32GB
Wifi 속도	다운로드 : 25.1 Mbps 업로드 : 27.4Mbps

표 3. 시험 환경

기본적으로 안드로이드에서 이미지를 불러와 표현하는 작업은 그림 7 와 같은 구조를 가진다[5]. 실험에서는 Activity 와 GridView, Adapter 는 동일하게 유지하고 Adapter 에서 이미지를 불러오는 부분만을 각 라이브러리를 이용해서 불러오도록 수정하였다. 이미지는 GridView 에서 15 장의 이미지를 동시에 로딩하도록 구현하였다.

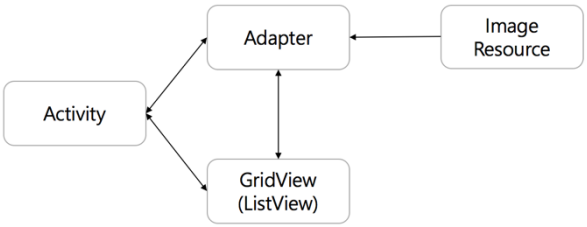


그림 7. 안드로이드 이미지 로딩 구조

그림 8 은 라이브러리를 실험하기 위한 어플리케이션 화면으로 구현 가이드에서 제공하는 기본 이미지 로딩 앱을 기준으로 하였다.

그림8. 이미지 로딩 라이브러리 테스트 어플리케이션 화면

5.2 이미지 로딩 속도 측정 결과

테스트 어플리케이션의 속도 측정 결과는 그림 9 와 같다. 각각 10 회씩 측정하여 최대값과 최소값, 그리고 평균값을 나타낸 결과이다. 캐쉬가 존재할 경우 메모리에서 곧바로 이미지가 로딩되어 백그라운드 스레드가 수행되지 않으므로 캐쉬가 존재하지 않는 상태를 설정하여 실험한 결과이다.

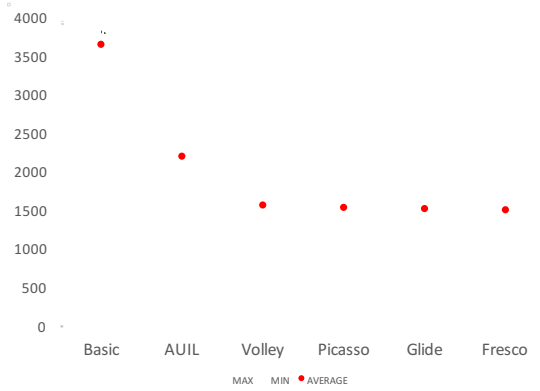


그림 9. 이미지 라이브러리 로딩 속도(ms)

측정된 결과는 최대값과 최소값의 차이가 존재하지만 Volley, Picasso, Glide, Fresco 4 개의 라이브러리가 로딩 속도가 유사함을 알 수 있다. AUIL 의 경우 다른 라이브러리들에 비해 로딩 속도가 느리지만 구현 가이드 방식보다는 빠른 결과를 보여준다. 샘플 어플리케이션은 다른 라이브러리에 비해 2 배 이상 로딩 속도가 느리다. 샘플 어플리케이션의 가장 큰 차이점은 스레드 생성 시 AsyncTask 를 사용한다는 점이다.

5.3 메모리 사용량 측정 결과

메모리 사용량은 안드로이드 SDK 에서 제공하는 adb dumpsys meminfo 명령어를 이용하여 실제 할당된 메모리 영역을 측정하였다. 각 이미지 라이브러리가 적절하게 이미지를 릴리즈하고 있는지 확인하기 위해서 99 개 이미지를 스크롤하는 시나리오를 사용하였다. 그림 10 은 해당 시나리오로 각 라이브러리별 할당된 메모리를 측정한 결과이다. 측정은 각각 5 회씩 측정하여 평균값을 구하였다.

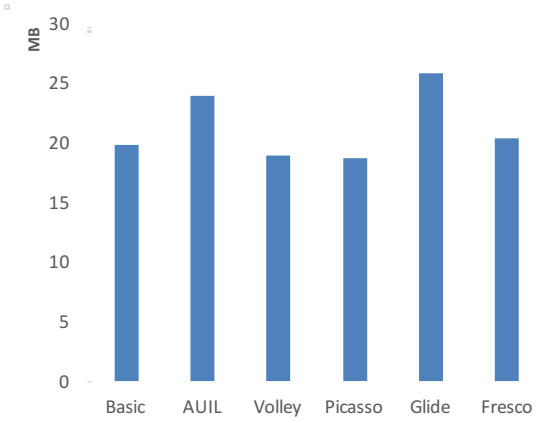


그림 10. 라이브러리 메모리 측정 결과

메모리 사용량에서는 AUIL 과 Glide 가 메모리를 많이 사용하는 것으로 나타났다. 그리고 Volley 와 Picasso 의 경우 메모리를 작게 사용하는 것으로 나타났다. 하지만 Volley 와 Picasso 의 경우 이미지의 크기를 지정해주지 않으면 원본 이미지를 메모리에 로딩하므로 주의가 필요하다.

아래 그림 11 는 이미지 로딩 시 할당되는 view 의 크기 정보를 라이브러리에게 전달하지 않았을 경우 사용되는 메모리 양을 측정한 결과이다.

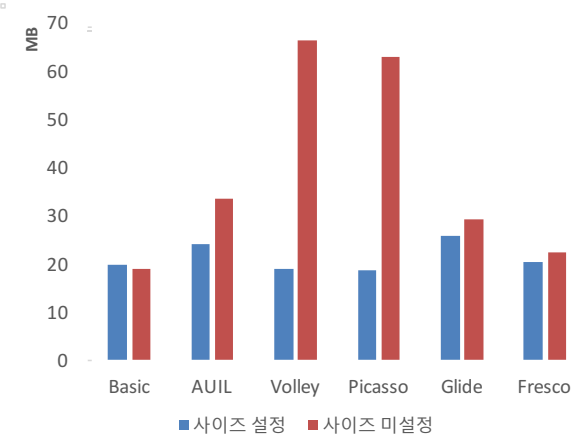


그림 11. 이미지 크기 미설정 시 메모리 사용량

Glide 와 Fresco 의 경우 별도로 이미지 크기를 지정하지 않아도 메모리 사용량이 유사함을 확인할 수 있다. 이는 라이브러리가 보여지는 view 크기에 따라 자동으로 비트맵 크기를 조정하기 때문이다. Volley 와 Picasso 의 경우 이미지 크기를 지정하지 않을 경우 원본 이미지 전체를 메모리에 할당한다. 결과적으로 3 배 이상 메모리 사용량이 증가함을 확인할 수 있다. 따라서 Volley 와 Picasso 의 경우 반드시 이미지 크기를 화면에 보여지는 view 크기에 따라 지정하여 사용하여야 한다.

6. 결론 및 향후 계획

지금까지 안드로이드 이미지 라이브러리들에 대해서 단순 기능 비교가 아닌 비기능적 품질 속성들을 비교하였다. Propagation Cost 를 이용한 모듈성 분석에서 Fresco 가 가장 우수하였다. 이미지 처리 속도 분석에서는 Volley, Picasso, Glide, Fresco 가 동등한 수준임을 확인하였다. AsyncTask 를 이용하였을 경우에는 스레드를 직접 구현한 다른 라이브러리들에 비해 속도가 느림을 알

수 있었다. 메모리 사용 분석에서는 Picasso 와 Volley 사용 시 이미지 크기 지정에 주의하여야 함을 확인하였다. 그리고 Github 에서 가장 많이 사용되는 AUIL 의 경우 다른 라이브러리들에 비해 모든 면에서 부족한 모습을 보였다.

이렇게 비기능적 품질 속성을 비교함으로써 단순 기능 비교 이외에 알 수 없었던 사실을 알 수 있다. 오픈소스는 사용에 따른 책임이 사용자에게 있다. 따라서 기능과 라이선스 검토 외에 비기능적 품질 속성에 대한 검토도 필요하다. 본 연구에서는 비기능적 품질 속성 중 모듈성과 성능을 기준으로 검토하였다. 하지만 이 외에도 다른 품질 속성 비교가 필요할 것이다. 따라서 향후 추가적인 비기능적 품질 속성 기준 개발과 오픈소스 도입 시 체계적인 검증 절차에 대해 연구하고자 한다.

참고문헌

[1] "Android Developers." [Online]. Available: http://developer.android.com/index.html.

[2] "Fresco | An image management library." [Online]. Available: http://frescolib.org/.

[3] T. O'Reilly, "The Open Source Paradigm Shift," *Various Things I've Written: Tim's Archive*, 2004. [Online]. Available: http://tim.oreilly.com/opensource/paradigmshift_0504.html.

[4] C. Y. Baldwin and K. B. Clark, "The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model?," *Manage. Sci.*, vol. 52, no. 7, pp. 1116-1127, 2006.

[5] "Google I/O 2013 - Volley: Easy, Fast Networking for Android." [Online]. Available: https://www.youtube.com/watch?v=yhv819F44qo.

[6] D. V. Steward, "Design Structure System: A Method for Managing the Design of Complex Systems," *IEEE Trans. Eng. Manag.*, vol. EM-28, no. 3, pp. 71-74, 1981.

[7] A. MacCormack, J. Rusnak, and C. Y. Baldwin, "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code," *Manage. Sci.*, vol. 52, no. 7, pp. 1015-1030, 2006.

[8] R. Milev, S. Muegge, and M. Weiss, "Design Evolution of an Open Source Project Using an Improved Modularity Metric," *Open Source Ecosyst. Divers. Communities Interact.*, vol. 299, pp. 20-33, 2009.

코드마인드: 온더플라이 소스코드 정적분석 도구

신승철, 이욱세, 노상훈, 김제민

코드마인드(주)
서울 구로구 디지털로 33 길 50
{shin,oukseh,rho,jemin}@codemind.co.kr

요약: 코드마인드®는 기존 소스코드 정적분석 도구의 맹점 극복을 위해 개발된 온더플라이(on-the-fly) 소스코드 정적분석 도구이다. 여기서는 정적분석 도구의 부진한 활용성을 개선하고자 채용된 기법들을 소개하고 소스코드 정적분석 기술의 발전 방향에 대하여 논의한다. 정적분석 도구는 시맨틱 분석 기술을 발전시켜서 더 정확하고 빠른 분석으로 소프트웨어 보안성과 안전성 확보 방안으로 떠오르고 있지만 실질적인 활용성은 부진한 편이다. 코드마인드는 온더플라이 분석, 결함 추적 시각화, 병렬성 극대화, 머신러닝 활용 분석 등의 특징을 이용하여 정적분석 도구의 활용성을 테스트 단계에서 개발 단계에까지 확장하고자 개발되고 있다.

핵심어: 정적분석, 실행오류 검출, 보안약점 검출, 온더플라이 정적분석, 소스코드 질의 기반 정적분석

1. 정적분석과 소프트웨어 안전성

소프트웨어 개발 프로세스에 따라서 설계하고 개발하고 테스트를 거친다고 해도 소스코드는 여전히 잠재적인 실행오류와 보안약점을 포함하기 때문에 시스템 비정상 종료, 메모리 누수, 보안 결함을 일으킬 수 있다. 소프트웨어의 안전성, 신뢰성, 보안성을 확보하기 위해서는 이들을 모두 찾아내서 제거해야 하는데 그 비용과 노력이 상당히 필요하다. 따라서 소프트웨어 공학적인 방법론 뿐 아니라 소스코드의 품질 제공을 위한 자동 도구가 필수적으로 사용된다.

특히 정적분석 도구는 프로그램 소스코드를 실행하지 않고도 프로그램 실행 값을 자동으로 예측하는 기술로 최근에 소프트웨어 공학 도구로서 한 자리를 차지하고 있다. 정적분석 기술은 (1) 소스코드의 구문 패턴을 식별해내거나, (2) 데이터나 제어 흐름을 분석하거나, (3) 심지어 요약해석 같은 기술을 이용해 실행 값을 예측하는 깊은 분석을 할 수 있다. (1)은 가장 값싼 분석이지만 가장 간단한 오류나 취약점만 찾을 수 있고 거짓경보의 가능성이 높다. (3)은 실행 중에 계산되는 값도 예측할 수 있어 더 정확한 검출이 가능하지만 비용이 비싸다. 보통 오픈소스 도구들은 대개 (1)에 해당하고 간혹 (2)를 포함하는 경우도 있다. 상용 도구들의 경우엔 (1)과 (2)를 포함하는 경우가 대부분이고 (3)을 포함하는 도구들은 소수이다.

국내에서는 전자정부 소프트웨어 납품시에 정적분석 도구를 이용한 보안취약점 검사가 의무화되었고 해외에서도 정적분석 도구를 이용한 소프트웨어 품질 확보가 트렌드로 확산되고 있다. 그러나 시장확대나 트렌드에도 불구하고 도구의 활용이 실질적이지 못하다는 지적이 나오고 있다[1,2]. 이것은 정적분석 기술의 태생적인 한계를 극복하기 위해 분석기술 자체에만 몰두했기 때문이다. 정적분석 도구가 실질적으로 활용되기 위해서는 기존 도구가 가지는 몇 가지 공통적인 맹점을 극복할 필요가 있다.

본 논문은 온더플라이 정적분석 도구인 코드마인드가 대용량 소스코드 분석문제, 거짓경보 판단문제, 다른 목적 도구와의 조합문제 등을 해결하기 위해 적용한 기법에 대하여 설명한다.

2. 정적분석도구의 맹점

정적분석기술은 크게 두 가지 차원에서 바라볼 수 있는데 하나는 분석하는 깊이로서 (1)구문패턴, (2)흐름패턴, (3)실행 값이 있고, 다른 하나는 분석하는 넓이로 (a)프로시저내 분석과 (b)프로시저간 분석이 있다. (1,a)는 가장 간단한 기술이고 (3,b)는 가장 복잡한 기술이다. 정확한 검출을 위해서 (3,b)의 분석을 하면 분석시간이 급격히 늘어난다. 따라서 대부분의 정적분석도구는 (3,b)를 아예 포기하거나, (3,b)를 (1),(2)에 가까운 성능수준으로 끌어올리려 노력하고 있다. 하지만 백만라인 이상의 대용량 소스코드를 효과적으로 다루기 어렵고 2 시간~8 시간이 걸리기 때문에 대부분 개발 중에는 정적분석도구를 사용할 수 없다.

또한 정적분석기술은 태생적으로 프로그램의 실제 값을 다루는 것이 아니고 요약된 값만을 다루기 때문에 거짓양성(false positive)를 가질 수 밖에 없다. 더 정확한 분석을 통해 정확도(=진짜 결함수/검출 결함수)를 높여가고 있지만 궁극적으로 정확도를 100%로 만드는 일은 불가능하다. 실제 프로젝트에서 정확도를 정확하게 계산하긴 어렵지만 벤치마크 SAMATE 를 기준으로 어림하면 도구와 검출규칙에 따라서 약 60% ~ 95%를 나타낸다. 한편, 활발하게 사

주) 코드마인드는 코드마인드(주)의 등록상표임.

용되는 오픈소스들도 정적분석도구로 검사하면 1,000 라인당 1.5 개의 결함이 검출된다. 갓 개발한 프로젝트의 경우 5 배~10 배의 결함이 검출된다. 이렇게 검출된 결함이 거짓양성인지 아닌지를 판별하는 일이 개발자에게 주어진다.

개발자가 코딩 중인 모듈을 머리 속에 가지고 있을 때 즉, 코딩 중간에 결함정보를 바로 알 수 있다면, 개발과 테스트가 끝난 후에 한꺼번에 결함 정보를 살펴봐야 할 때보다 거짓양성 판별은 한결 쉬울 것이다. 정적분석 도구가 테스트 도구로서가 아닌 개발 도구로 편입되어야 하는 이유이다.

3. 코드마인드 정적분석 기술

코드마인드는 소스코드 질의 기반 정적분석기술로 개발된 온더플라이 방식의 상용도구이다. 소스코드 질의란 소스코드를 데이터베이스나 지식베이스에 자료구조로 구축하고 데이터베이스와 지식베이스에 대한 질의를 작성하여 소스코드를 분석하는 기법이다 [3,4,5,6].

코드마인드는 검사 대상 소스코드를 분석하여 데이터흐름, 제어흐름, 호출관계 등 기반분석 정보를 데이터베이스에 저장한다. 또한 깊은 시맨틱 분석을 통해 실행요약값과 메모리요약값을 얻어낸다. 실행오류나 보안약점을 검출하는 체커모듈은 이런 정적분석 정보가 저장된 데이터베이스에 질의를 함으로써 결함 검출을 시행한다.

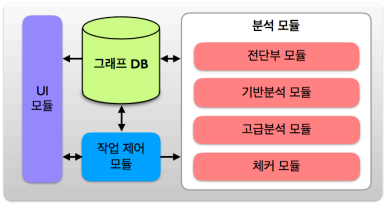


그림 1. 코드마인드 기술 개요

아래와 같은 Java 코드 몇 라인을 보면 SQL 인젝션의 전형적인 예제이다.

```
String tableName = props.getProperty("jdbc.tableName");
String name = props.getProperty("jdbc.name");
String query = "SELECT * FROM " + tableName + " WHERE Name =" + name;
PreparedStatement stmt = con.prepareStatement(query);
stmt.executeQuery();
```

이를 그림 2 와 같이 데이터베이스에 그래프로 구축하면 구문트리, 데이터흐름, 제어흐름, 호출관계 등을 하나의 그래프로 구성할 수 있다. 이에 대해 외부 입력이 SQL 문장에 직접 사용되는 경우를 검출하도록 질의를 작성하면 다음과 같다.

```
MATCH (src:Call{Name:'getProperty', ReceiverTypeName:'java.util.Properties'})-[:DATA*]->
(sink:Call{Name:'prepareStatement', ReceiverTypeName:'java.sql.Connection'})
CREATE (:Src)-[:AT]->(src)
CREATE (:Sink)-[:AT]->(sink)
```

이 질의를 통해 getProperty 결과값의 데이터흐름이 prepareStatement 의 인수값으로 전달됨을 확인하고 이 둘 사이의 추적 경로를 생성한다.

이 방식은 소스코드와 분석정보 모두를 데이터베이스를 통해 공유함으로써 모든 분석모듈 뿐 아니라 시각화 모듈, UI 모듈까지 모두 독립적이고 병렬적으로 작업할 수 있다.

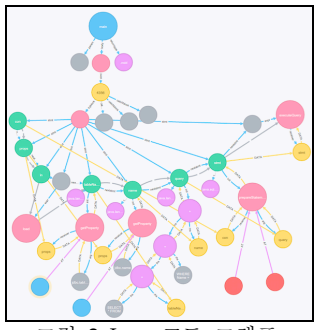


그림 2. Java 코드 그래프

따라서 전체 코드의 검사가 끝나지 않아도 해당 파일의 검사결과가 나오는대로 바로 볼 수 있다. 또한 검사 중간에 언제든지 시각화 모듈과 UI 모듈의 연동으로 결함추적을 통해 거짓양성 판별이 용이하다. 게다가 데이터베이스를 통한 인터페이스로 클론 체커, 영향분석기, 메트릭모듈 등 다른 용도의 모듈과의 연동이 자연스럽다.

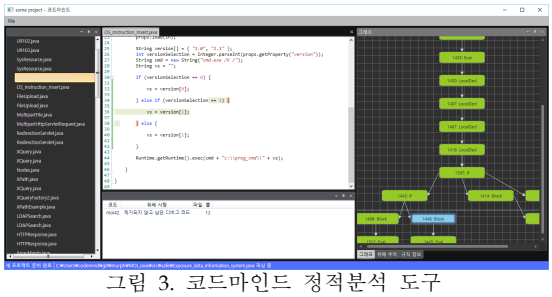


그림 3. 코드마인드 정적분석 도구

4. 결 론

코드마인드는 소스코드 질의 기반 정적분석 기술로 개발된 온더플라이 방식의 상용도구이다. 대용량 소스코드 분석시간 과다문제와 거짓양성 판별문제를 해결하는 새로운 대안으로 개발되고있다.

참고문헌

[1] B. Johnson, Y. Song, E. Murph-Hill, R. Bowdidge, Why Don't Software Developers Use Static Analysis Tools to Find Bugs?, ICSE 2013

[2] M. Bradley, F. Cassez, A. Fehnker, T. Given-Wilson and R. Huuck, High Performance Static Analysis for Industry, ENTCS 289, pp.3–14, 2012

[3] E. Hajiyev, M. Verbaere, O. De Moor, Codequest: Scalable source code queries with datalog, ECOOP–OOP, pp. 2-27, 2006

[4] R.-G. Urma, A. Mycroft, Expressive and scalable source code queries, OOPSLA 2013

[5] 홍성문, 신승철, 도경구, 지식베이스를 이용한 보안취약점 정적분석, 한국정보과학회 한국컴퓨터종합학술대회논문집, pp. 1618-1620, 2014

[6] F. Yamaguchi, Modeling and discovering vulnerabilities with code property graphs, IEEE Security and Privacy, 2014

유사도 메트릭을 이용한 컴포넌트 상호연관성 추출

이재권, 김기섭, 정우성

충북대학교 전자정보대학
충북 청주시 서원구 충대로 1, E8-1 동
{ exatoa, nadanear, wsjung}@cbnu.ac.kr

요약: 컴포넌트 재사용 기반 소프트웨어 개발 방법론은 개발 시간 단축, 보안성 강화, 품질향상 등의 다양한 장점을 가진다. 하지만 개발자가 접해보지 않은 분야에서는 컴포넌트에 대한 선행 학습이 필요할 뿐 아니라, 컴포넌트 조합들 간에 발생하는 잠재적 문제점 때문에 재사용이 이루어지기 어렵다. 본 논문에서는 컴포넌트 종속성 정보를 기반으로 사용패턴들을 분석하여 함께 사용되는 컴포넌트들의 상호연관성을 정의하고 오픈 소스 프로젝트를 대상으로 이를 추출해봄으로써 향후 컴포넌트 개발자 별 맞춤형 추천 시 정확도를 높일 수 있는 방법을 제안한다.

핵심어: 소프트웨어 컴포넌트, 재사용 패턴, 연관규칙 마이닝, 유사도 메트릭, 상호연관성

1. 서 론

최근, 소프트웨어의 크기, 복잡도가 증가하면서 개발비용, 유지보수 비용 등이 증가하고 있다. 이에 따라 비용단축을 위한 다양한 개발 방법론이 도출되었고, 그 일원으로 컴포넌트를 재사용하여 문제를 해결하고자 하는 컴포넌트 기반 소프트웨어 개발 방법론(CBSD)이 발전해왔다[1]. 컴포넌트의 재사용은 개발시간을 단축하는 효과가 있지만[2], 컴포넌트의 탐색, 구매, 학습, 적용비용 발생이 불가피하다. 뿐만 아니라 컴포넌트 조합에 따라 창발적 속성에 의해 문제를 일으키기도 한다[3]. 따라서 내 프로젝트와 어울리는 적절한 컴포넌트를 검색하는 문제가 대두되었다.

재사용 가능한 컴포넌트 검색을 위해 Ichii 는 협업 필터링(Collaborative Filtering)을 이용한 컴포넌트 추천 시스템을 제안하였다[4]. 그러나 컴포넌트의 관계에 대해서는 고려하지 않아 추가적인 비용이 발생한다. 이에 메이븐(Maven) 프로젝트에서는 종속정보를 가지고 컴포넌트 사용시 자동으로 하위 컴포넌트를 연결해주는 시스템을 개발하였고, Thung[5]은 컴포넌트의 종속성 정보에 연관규칙 마이닝(Asso-ciation Rule Mining)[6]과 협업 필터링을 적용하여 컴포넌트 간의 사용패턴에 기반한 추천방법을 제안하였다.

본 논문에서는 메이븐 저장소의 컴포넌트 종속성 정보를 기반으로 연관규칙 마이닝을 통해 컴포넌트들 간의 사용패턴을 분석하고, 단순 연관성 및 상호연관성을 정의 및 추출함으로써 추후 컴포넌트를 추천하는 과정에 개발자의 실수를 줄이고 도움을 줄 수 있는 방안을 제시하였다.

2. 컴포넌트 연관관계 마이닝

2.1 컴포넌트 관계 속성 정의

본 절에서는 컴포넌트 간의 관계를 정의하고, 관계 식별을 위한 메트릭을 정의한다.

종속성 X 컴포넌트가 Y 컴포넌트를 사용할 때 X 는 Y 에 종속성을 가진다고 한다. 또한 직접 사용하지는 않지만 간접적으로 사용하게 되는 하위 종속성도 존재한다. 그림 1 은 컴포넌트 A 가 두 개의 컴포넌트 B, D 를 사용하고 있는 모습이다. 이 때 C 는 A 가 사용하는 컴포넌트 B 가 사용하기 때문에 A 는 C 에 하위 종속성을 가진다.

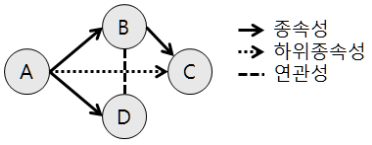


그림 1. 컴포넌트의 관계

연관성 두 컴포넌트가 종속성을 가지지 않지만 함께 자주 사용되는 경우, 두 컴포넌트는 연관성이 있다고 한다. 이는 여러 개의 컴포넌트 종속정보를 분석하여 도출한다. 연관성에 판단 시 간접적 종속인 하위 종속성이 포함되면 잘못된 규칙을 발생시킬 수 있으므로 마이닝 데이터에서 제거하고 계산한다.

종속정보 분석을 위해 연관규칙 마이닝 시에 사용하는 메트릭인 지지도(support), 신뢰도(confidence), 향상도(lift)를 활용한다[6]. 연관규칙 마이닝은 특정 사건이 발생한 후 다음 사건이 발생할 확률을 이용하여 규칙을 도출하여 방향성을 가진다. 즉, X→Y 확률과 Y→X 확률은 서로 다를 수 있다.

상호연관성 양방향으로 연관성을 가지는 컴포넌트들 간의 관계를 상호연관성이라 정의한다. 주로 양방향으로 높은 연관성을 지는 컴포넌트가 해당한다.

상호연관성의 판단은 자카드 유사도(Jaccard Simi-

larity)를 이용한다. 자카드 유사도는 두 집합간의 교집합과 합집합의 비율을 이용하여 유사도를 계산하여 전체 사용 패턴에서 두 컴포넌트 사용패턴의 유사도를 의미하는 상호연관성의 판단에 적절하다. 수식 4 는 두 컴포넌트 C_i, C_j에 대해 지지도를 이용한 자카드 유사도를 나타낸다.

$$Sim(C_i, C_j) = \frac{Support(C_i \cap C_j)}{Support(C_j)+Support(C_j)-Support(C_i \cap C_j)} \quad (1)$$

2.2 연관규칙 마이닝

상호연관성은 연관성의 판단에 사용되는 메트릭을 이용하여 정의하므로 같은 방법으로 계산될 수 있다. 본 논문에서 각 관계를 도출하기 위해 Apriori 알고리즘[6]을 변형하여 이용하였다. 그림 2 은 변형된 연관규칙 마이닝 과정 중 규칙 생성에 대한 의사코드이다.

```
Input:
Cond: Calculate confidence or similarity from data
   and check condition of metrics
pRules : Previous rules
function Apriori(pRules, Cond):
01: rules = list()
02: unionset = make_unionset(pRules)
03: for antecedent in pRules:
04:    for consequent in unionset:
05:       if consequent in antecedent:
06:          continue
06:       rule = Rule(antecedent, consequent)
07:       if Cond.check(rule)==True:
08:          rules.append(rule)
09: return rules
```

그림 2. 연관규칙 마이닝 의사(pseudo)코드

3. 사례연구

실험은 메이븐 저장소로부터 15,264 개의 컴포넌트와 5,621 개의 종속성 데이터를 받아서 하위 종속성을 제거한 후 7,123 개의 컴포넌트와 4,313 개의 데이터를 이용하여 실험하였다.

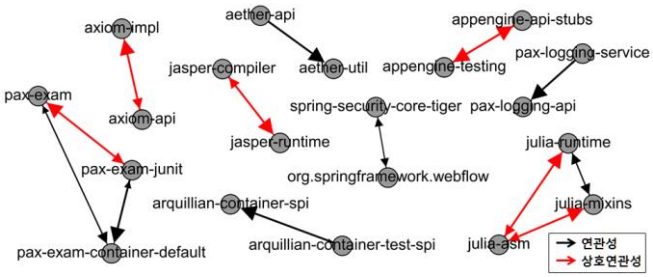


그림 4. 컴포넌트 관계 그래프

그림 4 는 지지도 0.2%이상, 신뢰도 60%이상인 컴포넌트들의 관계를 그래프로 표현한 것이다. 화살표의 크기는 신뢰도의 크기를 나타내고, 유사도 90%이상인 관계에 대해서는 상호연관성으로 나타내었다.

그래프에 나타난 대부분의 컴포넌트들은 이름에서 연관성 및 상호연관성을 유추해 볼 수 있다. {aether-api, aether-util}관계에서 util 은 api 의 보조적인 역할로 프로젝트에 따라 선택적으로 사용되므로 연관성으로 분류되었다. {appengine-api-subs, appengine-testing}에서는 테스팅을 위해 두 컴포넌트가 완전히 동시에 사용되어 상호연관성으로 분류되었다.

4. 결 론

본 논문에서는 컴포넌트 종속성 정보를 기반으로 컴포넌트들 간의 사용 관계인 연관성들을 추출했고 상호 연관성을 정의 및 도출함으로써 개발자 별 맞춤형 재사용 컴포넌트 추천 시 정확도를 높일 수 있는 방법을 제안했다. 또한 상호연관성은 컴포넌트 추천의 랭킹을 결정할 때 자동화된 수치로서 활용이 가능하다. 향상도의 경우 값의 범위가 한정되지 않은 반면 유사도의 값은 [0,1]사이의 값으로 한정된 범위이므로 절대적인 비교가 가능한 장점이 있다.

향후에는 컴포넌트의 소스코드나 문서를 분석한 정보를 추가하여 사용자의 문맥에 맞는 컴포넌트 추천 시스템으로 확장할 계획이다.

Acknowledgement

이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2014M3C4A7030505, No. 2015R1C1A1A01054994)

참고문헌

[1] M. R. J. Qureshi and S.A Hussain, “A Reusable Software Component-Based Development Process Model,” Advances in Engineering Software, Vol. 39, pp. 88-94, 2008.

[2] M. L. Griss, “Software reuse at hewlett-packard,” in Proc. of the 1st International Workshop on Software Reusability, 1991.

[3] I. Sommerville, “Software Engineering,” Pearson, 9th ed., 2010.

[4] M. Ichii, Y. Hayase, R. Yokomori, T. Yamamoto and K. Inoue, “Software component recommendation using collaborative filtering,” in Proc. of the 31st International Conference on Software Engineering Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation, pp.17-20, 2009.

[5] F. Thung, D. Lo, and J. L. Lawall, “Automated library recommendation,” in Proc. of the 20th Working Conference on Reverse Engineering(WCRE), pp.182–191, 2013.

[6] R. Agrawal and R. Srikant, “Fast algorithms for min-ing association rules”, in Proc. of the 20th International Conference on Very Large Data Bases, pp.487-499, 1994.

다양한 이해관계자들 간 정보공유를 위한 산업용 로봇 소프트웨어 설계 기법

박진용, 유철중

전북대학교 소프트웨어공학과
전북 전주시 덕진구 백제대로 567
{jypark_se, cjyoo}@jbnu.ac.kr

요약: 산업용 로봇은 소프트웨어 전문가 외에도 하드웨어 전문가, 산업체 대표, 엔드 유저 등 다양한 이해관계자들과 의사소통을 하며 개발이 진행된다. 산업용 로봇의 기능 개발에 소프트웨어를 설계하는 부분에서 어떤 내용이 설계되고 있는지는 소프트웨어 관련 전문가를 제외하고 다른 이해관계자들은 내용을 파악하기 어려운 실정이다. 본 논문에서는 이런 문제점을 해결하고자 사용자 입장에서의 기능을 정리할 수 있는 모델링 방법 중 유스케이스를 사용하여 가시적으로 현재 설계 내용을 비 전문적인 이해관계자가 이해할 수 있도록 하되, 각 기능들을 개발하는데 있어 보다 산업용 로봇에 적합하도록 유스케이스 명세와 다이어그램을 수정·적용하여 정보공유뿐만 아니라 개발에도 도움 될 수 있는 설계 기법을 제시하고자 한다.

핵심어: 산업용 로봇, 유스케이스, 설계 기법, 가시성

1. 연구배경

산업용 로봇은 반복적이고 정밀한 작업을 사람을 대신하여 지속적으로 수행할 수 있어 다양한 산업현장에서 사용되고 있다. 다양한 산업현장에서 사람이 하는 작업을 대신하는 만큼 산업용 로봇이 수행해야 할 작업들도 다양하며[1] 이를 위해 기능의 관리가 필수적이다. 하지만 이러한 기능들을 관리하는데 있어 소프트웨어 개발자 외에 다양한 이해관계자들과 협력을 하는 경우가 많은데[2], 이 때 서로간에 의사소통을 하는데 있어 공유할 자료가 부족한 실정이다. 즉, 비 전문가도 이해할 수 있는 문서 또는 시각적 자료가 부족한 것이다. 이로 인해 개발되고 있는 실제 기능은 프로토타입 수준까지 개발이 된 다음에 확인이 가능하며 해당 기능에 문제가 있을 경우 수정하는데 비용이 초기에 문제를 해결한 것 보다 더 많이 들어갈 수 밖에 없다[3][4]. 본 논문에서는 비전문가와 문서 또는 시각적으로 산업용 로봇의 기능에 대해 정보공유를 할 수 있도록 유스케이스를 활용하며, 산업용 로봇의 기능을 다루는 점에서 유스케이스 명세 및 표기에 변화를 줘서 보다 산업용 로봇에 적합한 기능 정의 및 관리가 가능한 소프트웨어 설계 기법을 제시한다.

본 논문의 구성은 다음과 같다. 2 장에서는 본 논문의 기본 개념인 유스케이스와 산업용 로봇 소프트웨어의 특징을 설명한다. 3 장에서는 유스케이스의 명세와 표기를 변경하고 이를 통해 실제 산업용 로봇 소프트웨어 설계에 적용하는 방법을 알아본다. 4 장에서는 제시한 기법에 대해 간단히 평가한 후 5 장에서는 결론 및 향후 연구방향에 대해 논의한다.

2. 기본 개념

2.1 유스케이스

유스케이스는 실제 기능을 관리하는데 자주 사용되며 시스템을 사용하는 액터와 시스템 간의 상호작용을 시간적 흐름에 맞게 명세하여 추후 소프트웨어를 개발하는데 기반이 된다. 아래 [표 1]은 유스케이스 명세에 들어가는 내용에 대한 설명이고 [그림 1]은 유스케이스 다이어그램의 예시를 나타내고 있다[5].

표 1 유스케이스 명세

유스케이스 명	유스케이스를 나타내는 명칭
액터	시스템과 상호작용하는데 있어 이벤트를 발생시키는 요소
사전조건	유스케이스가 실행 될 수 있는 조건
메인흐름	정상적인 유스케이스 실행 흐름, 시스템과 액터 간 상호작용을 표현
예외흐름	메인흐름에서 예외적으로 발생할 수 있는 상황의 대체흐름을 표기
사후조건	유스케이스가 실행 된 이후의 상태

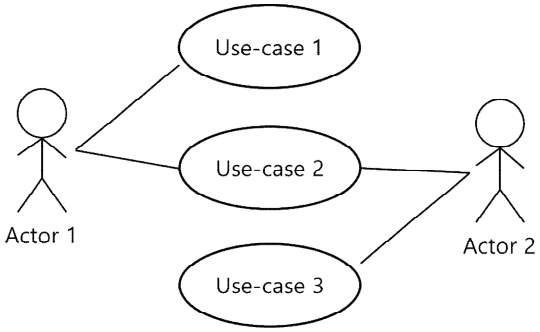


그림 1 유스케이스 다이어그램

다이어그램 상에서 타원은 유스케이스를 나타내고 사람모양의 그림은 액터를 나타낸다. 이들 간 상호작용이 있는 관계일 경우 선으로 이어서 나타낸다. 유스케이스간 관계의 경우 포함과 확장의 개념이 있지만 본 논문에서는 다루지 않으므로 넘어가도록 한다.

2.2 산업용 로봇 소프트웨어의 특징

산업용 로봇은 다양한 구성요소를 이용하여 앞에서 언급한 것처럼 여러 기능들을 수행한다. 이 때 각각의 기능을 수행할 때는 하나의 작업 프로세스가 수행되고, 이 프로세스는 여러 구성요소들 각각의 세부적인 동작들로 이뤄져 있다. 즉, 하나의 작업 프로세스는 여러 세부 동작들이 순서대로 실행된 결과라고 볼 수 있고 다음 [그림 2]는 이를 표현하는 그림이다.

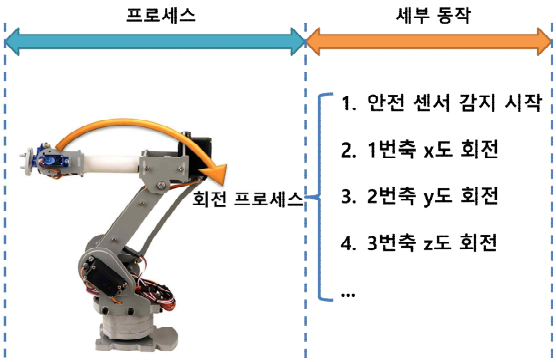


그림 2 산업용 로봇의 프로세스와 세부 동작

이러한 측면을 미루어보아 유스케이스와 유사한 측면이 있음을 알 수 있으며, 다음 [표 2]는 산업용 로봇 소프트웨어의 세부 특징들과 유스케이스의 명세 사이의 유사한 측면을 비교하고 있다.

표 2 산업용 로봇 S/W 세부 특징과 유스케이스 비교

산업용 로봇 소프트웨어 세부 특징	유스케이스
프로세스(동작) 이름	유스케이스 명
프로세스 사용 객체 및 클래스	액터
세부 동작	메인흐름
세부 동작 수행 중 예외 발생 처리부	예외흐름

또한 산업용 로봇 소프트웨어는 앞서 언급한 많은 종류의 세부 동작들을 다양한 구성요소들을 이용하여 수행된다. 이 구성요소들로는 모터, 카메라, 센서, I/O 장치 등 매우 다양하며, 이 구성요소들은 직접 개발하는 경우보단 다른 업체에서 개발한 제품들을 구매해서 조합하는 경우가 많다. 이 때 각각의 구성요소들은 개발업체에서 제공하는 라이브러리가 있으며, 결국 소프트웨어적으로는 라이브러리들의 내용을 조합하여 산업용 로봇 소프트웨어가 개발된다. 즉, 각 구성요소들에게는 이들을 제어할 수 있는 소프트웨어 라이브러리가 존재하며 만약 이들에 대한 변경이 필요할 경우엔 라이브러리의 내용이 변경되기 때문에 소프트웨어적인 수정비용이 발생한다. 이 수정비용을 파악하기 위해선 변경되는 라이브러리의 파급범위를 먼저 파악해야 한다. 파급 범위를 파악하기 위해선 라이브러리의 변경으로 인해 수정해야 하는 클래스나 함수의 범위를 측정할 수 있어야 한다.

3. 유스케이스 활용 기능 관리 기법

앞서 기본 개념에서 설명한 유스케이스를 산업용 로봇 소프트웨어의 기능을 관리하는데 있어 보다 효과적으로 이행하기 위해 산업용 로봇 소프트웨어 특징에 맞게 유스케이스의 명세 내용과 다이어그램을 커스터마이징 한 후 적용하도록 한다.

3.1 유스케이스 명세 커스터마이징

첫 번째로 문서를 통해 다양 한 기능들의 정확한 정보 공유를 위해 유스케이스 명세 부분의 커스터마이징을 알아보며, 2.2 절의 [표 2]를 바탕으로 커스터마이징이 진행된다.

표 3 유스케이스 명세서 커스터마이징

유스케이스	프로세스 명세
유스케이스 명	- 프로세스(동작) 이름
액터	- 프로세스 사용 클래스와 라이브러리를 나눔
메인흐름 및 예외흐름	- 세부 동작의 흐름 명시 - 각 흐름의 하나의 동작들은 가장 작은 단위의 기능 - 각 동작들은 다른 프로세스

	와 공유할 수 있으며 고유한 번호가 할당됨
사전, 사후조건	- 프로세스들 사이에 실행 규칙이 존재할 경우 추가 기입

위의 [표 3]은 커스터마이징한 결과를 요약하고 있으며 자세한 설명은 다음과 같다.

첫 번째, 프로세스 명은 해당하는 프로세스를 나타내는 명칭이기 때문에 기존의 유스케이스 명세상의 이름과 동일하게 유지한다.

두 번째, 프로세스 사용 객체 및 클래스는 [표 2]에서 액터와 비교하고 있는데, 해당 프로세스상에서 사용하는 라이브러리들과 이 프로세스를 사용하는 측의 클래스를 명시하도록 한다. 그러기 위해 기존의 '액터' 한 줄로 표현하던 명세서를 '사용 클래스' 와 '라이브러리' 로 나누도록 수정한다.

세 번째, 메인 흐름과 예외 흐름은 세부 동작들의 순서를 나타내는 것으로 대체하며, 여기서 각 흐름에 한 줄의 내용은 가장 작은 단위의 기능으로 구현될 것이다. 만약 이전에 정의한 기능을 다른 프로세스를 명세하는데 사용하게 되면 해당 기능을 그대로 사용할 수 있도록 한다. 이를 위해 각 기능들은 흐름을 표기할 때 번호를 매기는데 일반적인 유스케이스와 달리 각 기능들 고유의 번호를 할당한다. 즉, 앞의 프로세스 명세에서 정의된 기능이 다시 사용되어야 할 경우 번호를 할당하는데 있어 기존의 번호를 그대로 사용하도록 한다. 또한 기존의 유스케이스 명세의 경우 메인흐름과 예외흐름에는 액터와 시스템간 상호작용을 나타내지만 프로세스 명세에서는 기능 중심으로 명시하기 때문에 해당 기능을 구현하는 관점에서 어떤 사용 클래스와 라이브러리가 연관되어 있는지 표시한다.

다음은 위의 내용을 바탕으로 유스케이스 명세서를 커스터마이징한 프로세스 명세서의 간단한 예시이다. 단, 현재 단계에서는 상위 개념에서의 명세 내용이기 때문에 사용 클래스 및 라이브러리의 명칭을 한글로 표기한다.

표 4 프로세스 명세 예시

프로세스 이름	로봇 회전
사용 클래스	물품 운송 컨트롤러
라이브러리	모터 API, I/O 센서 API
메인흐름	1. 회전 이동 명령 수신 시 각 축 모터의 위치를 원점으로 회전 [모터 API] 2. 각 축 모터를 정해진 위치로 회전 [모터 API] 3. 모터의 위치 값이 목적지에 도달하면 정지 후 완료 메시지 및 사이렌 출력 [모터 API, I/O 센서 API]
예외흐름	4. 모터의 회전 반경에 물체가 있을 경우, 경고 메시지 및 경고음 출력 [I/O 센서 API]

앞서 커스터마이징에 대해 설명한 내용과 마찬가지로 프로세스 이름, 사용 클래스, 라이브러리, 메인흐름, 예외흐름을 명시하였다. 중점적으로 보아야 할 부분은 메인흐름과 예외흐름을 명시하는 부분인데, 각 흐름들은 메인흐름, 예외흐름에 상관없이 고유의 번호를 지니고 있다. 만약 다른 프로세스가 정의될 때 추가적으로 발생하는 기능들은 5 번부터 번호가 할당될 것이며, 위의 '로봇 회전'이라는 프로세스에 정의된 기능을 사용할 경우에는 해당 기능의 고유번호를 그대로 사용하여 표기하게 된다. 또한 각 흐름마다 가장 마지막 부분에 중괄호를 통해 각 기능들이 사용하는 라이브러리를 나타냈으며, 만약 사용 클래스가 많아질 경우 사용 클래스도 같은 방식으로 추가 표기가 가능하다.

3.2 유스케이스 다이어그램 커스터마이징

두 번째로 이해관계자들 중 비 전문가들도 각 기능에 대해 가시적으로 쉽게 정보를 습득할 수 있도록 유스케이스 다이어그램을 커스터마이징한다. 기존의 유스케이스 다이어그램을 크게 분류하면 유스케이스, 액터, 간선 정도로 나타낼 수 있는데, 본 논문에서 제시한 프로세스 명세에서 액터를 두 부분으로 나눴기 때문에 이에 맞도록 다이어그램이 변경되어야 한다. 또한 메인흐름 및 예외흐름에서 각 기능으로 분류되는 흐름들은 각자 매핑되는 사용 클래스와 라이브러리가 존재하기 때문에 이에 대한 정확한 표기도 필요하다. 다음 [그림 3]은 이러한 조건을 충족시키기 위해 기존의 유스케이스 다이어그램을 변경한 [표 4]의 프로세스 다이어그램이다. 이 다이어그램은 기존 유스케이스 다이어그램의 전체적인 액터-유스케이스 간 관계를 확인하는 목적보다 하나의 프로세스에 집중적으로 관계가 어떻게 이뤄져 있는지 확인하는 것을 목적으로 한다.

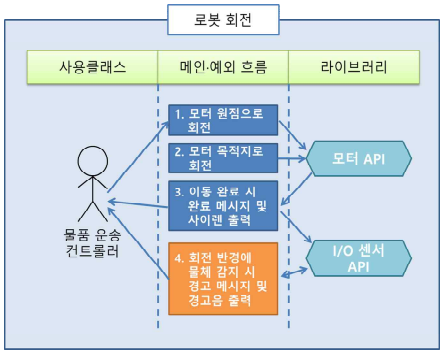


그림 3 프로세스 다이어그램 예시

프로세스 다이어그램의 표기법은 아래와 같다.

- 전체 구조 : 사용클래스, 각 흐름, 라이브러리의 구역을 나눠 표기
- 사용 클래스 : 기존 유스케이스 다이어그램의 액터와 동일
- 메인 흐름 : 푸른 계열의 사각형으로 표기
- 예외 흐름 : 붉은 계열의 사각형으로 표기
- 라이브러리 : 6 각형의 도형에 표기
- 간선 : 방향성을 띄며 메시지 또는 명령의 방향을 나타냄

4. 평 가

프로세스 명세 및 다이어그램을 이용한 산업용 로봇 소프트웨어 설계 기법의 가장 중요한 포인트는 유스케이스를 이용하여 비전문적인 다양한 이해관계자들에게 산업용 로봇 소프트웨어 설계 과정에서의 의사소통을 지원하는 부분이고 그 다음으로 유스케이스를 수정· 적용 하여 앞으로 개발에 도움이 되도록 하는 것이었다. 이에 대해 각각 간단히 살펴보도록 한다.

4.1 원활한 정보 공유

대부분의 산업용 로봇 소프트웨어 개발에서 기능적인 부분의 설계에는 소프트웨어 전문가들 간의 논의를 통해 결과가 도출되며 그 결과는 비전문가들이 확인하여 정보를 습득하기 어렵다. 하지만 본 논문에서 제안한 프로세스 다이어그램은 명확하게 각 기능에 대해 사용자 레벨에서의 표기를 하였고, 가시적으로 어떠한 구성요소가 해당 기능에 사용되는지를 파악할 수 있다. 이를 확장시키면 각 구성요소들의 변경에 대한 비용도 각 프로세스 또는 동작들의 변경 비용을 개발자들이 책정해두면, 소프트웨어 전문가가 아니더라도 도식화된 설계 구조를 보고 가늠할 수 있게 된다

4.2 기능 개발 지원

본 논문에서 제안하고 있는 설계 기법은 하나의 큰 기능을 담당하는 프로세스마다 세부적인 기능들을 명시하고, 이들이 사용하는 라이브러리를 방향성 있는 간선을 통해 연결하고 있기 때문에 향후 해당 기능의 개발을 진행할 때 라이브러리의 호출과 메시지 송수신 등을 가시적으로 확인할 수 있고, 기능 테스트 수행 시 테스트 베이시스로도 활용이 가능하다.

5. 결 론

오랜 시간은 아니었지만 산업용 로봇 소프트웨어 개발자로 실무를 수행하면서 가장 안타까웠던 부분인 소프트웨어에 대해 비전문가들과의 의사소통의 기본적인 자료가 항상 부족한 점을 보안하고자 프로세스 명세와 다이어그램 표기를 유스케이스를 커스터마이징하여 제시해보았다. 평가 부분에서 언급한 부분은 현재 실험을 통한 결과가 아닌 기대 효과를 말하고 있다. 하지만 앞으로 연구를 지속적으로 진행하면서 4 장에서 평가하고 있는 항목들에 대한 적절한 테스트 기법을 선정 또는 도출하여 본 논문에서 제시한 설계 기법의 효율성을 수치적으로 측정할 것이다. 또한 각 기능들이 조합되어 하나의 큰 역할을 수행하는 부분을 미루어보아 디자인패턴 중 커맨드 패턴과의 유사성이 있으므로 커맨드 패턴의 적용도 고려하여 연구를 진행할 것이다.

사사

본 논문은 중소기업청에서 지원하는 2015 년도 산학연협력 기업부설연구소 지원사업(NO.C0267942)의 연구수행으로 인한 결과물임을 밝힙니다.

참고문헌

[1] 이충동, “산업용 로봇 시스템 기술”, Journal of the KSME, vol. 51, no. 2, pp. 26-29, 2011

[2] Jong-In Jang, Jongmoon Baik, “Automated Requirements Prioritiztion using Stakeholder Needs Artifacts and Topic Modeling Technique”, Proceedings of the Korean Information Science Society Conference, pp 564-566, 2015

[3] R. Charette, “Why Software Fails”, IEEE Spectr., vol. 42, no. 9, pp. 42-49, 2005

[4] Jun-Ha Lee, Uije Park, Yong B. Park, Soojin Park and Sooyong Park, "Modularizing Software using Direct and Indirect Class Coupling Metrics", Journal of KIISE, vol.41 no.5, pp 327-339, 2014

[5] Kurt Bittner, Ian Spence: “Use Case Modeling”, Addison-Wesley Professional, 2002

스마트 환경에서 디지털 사이니지와 이기종 디바이스간의 상호운용성을 지원하기 위한 메시지 플로팅 시스템

차민재*, 이동우*, 남태우*, 염근혁**

*부산대학교 전기전자컴퓨터공학과, **부산대학교 정보컴퓨터공학부
부산 금정구 장전동 산 30
{ mj.deff, ldw0706, kaluas, yeom}@pusan.ac.kr

요약: 최근 스마트폰을 비롯한 스마트 기기들의 발전과 IoT 디바이스들의 네트워킹 기술들의 발전에 따라 실내/외를 대상으로 스마트 환경이 등장하고 있다. 특히 실외의 스마트 환경에서 디지털 사이니를 통해 쇼핑몰, 지하철, 관광지, 유동인구가 많은 공공장소 등에서 활용사례가 점차 늘어나고 있다. 하지만 디지털 사이니지에서 제공하는 콘텐츠는 사용자들에게 지속해서 콘텐츠를 제공할 수 없고, 설치된 디지털 사이니지를 제어하는 방법도 제약사항이 있다. 따라서 본 논문에서는 이기종 디바이스간의 콘텐츠 및 이벤트 전달을 지원하는 메시지 플로팅 시스템을 제안한다. 메시지 플로팅 시스템은 사용자가 디지털 사이니지 구역 내에서 제공 받고 있던 콘텐츠를 단절 없이 연계하여 제공받는 방법과 디지털 사이니지를 쉽게 제어하는 방법을 적용한 시스템이다.

핵심어: 웹소켓, 디지털 사이니지(Digital Signage), 아키텍처, 상호운용성

1. 서론

최근 IoT 디바이스들의 네트워킹 기술들이 발전되면서 유비쿼터스의 실현이 더욱 구체화되고 있다. 특히 지능형 IoT 디바이스들이 네트워크로 연결되어 사람이 인지하지 않아도 자연스럽게 상호작용을 통해 인간 중심의 서비스 환경을 지원하는 스마트 환경에 대한 연구가 활발히 이루어지고 있다[1,2,3,4,5].

스마트 환경은 실내/외를 대상으로 연구가 진행되고 있다. 실내로는 스마트 오피스, 스마트 홈 등이 있고, 실외로는 유동 인구가 많은 공공장소나, 번화가 같은 곳에서 스마트한 환경의 연구가 많이 진행되고 있다. 실내의 스마트 환경에서는 NFC, 무선 인터넷과 같은 네트워크 기술을 통해 서비스를 제공받거나, 사용자 인증을 통해 디바이스를 직접 제어할 수 있는 환경이 많이 연구 개발되고 적용이 되어있다. 실외의 스마트 환경에서는 디지털 사이니지[5]를 통해 보행자들에게 정보, 엔터테인먼트, 광고 등과 같은 콘텐츠를 제공하는 환경들이 연구 개발되고 적용되어 있다. 디지털 사이니지 기반의 콘텐츠는 NFC, 블루투스, 영상처리 기술들의 발전되면서 사용자와 양방향 커뮤니케이션이 강화되고 있다. 그러나 콘텐츠를 제공받는 사용자가 디지털 사이니지가 설치된 구역을 벗어났을 때는 더 이상 콘텐츠를 제공받을 수 없다. 또한, 사용자가 디지털 사이니지를 제어하는 데에 있어서 문제점이 있다. 문제점으로는 기존의 디지털 사이니지의 제어 방법으로는 터치 스크린을 이용해 입력을 받아 제어를 했는데, 터치 스크린이 고장이 나면 더 이상 제어가 불가능 해지게 된다.

본 논문에서는 이러한 문제점들을 해결하기 위해서 사용자가 제공받고 있던 콘텐츠를 디지털 사이니지가 설치된 구역을 벗어나더라도 매끄럽게 이어서 모바일 단말기에서 제공받을 수 있고, 사용자의 모바일 디바이스로 별도의 인증 없이 디지털 사이니지 디바이스를 제어할 수 있게 해주는 메시지 플로팅 시스템을 제시한다.

본 논문은 다음과 같이 구성되어 있다. 2 장에서는 시스템에서 사용되는 네트워크 기술인 웹소켓과 기존의 연구들에 대해 설명한다. 3 장에서는 콘텐츠 플로팅을 위한 기본 개념과 프로세스에 대해 설명한다. 4 장에서는 콘텐츠 플로팅 시스템 구현 결과에 대하여 설명하고, 5 장에서는 결론 및 향후 연구에 대하여 설명한다.

2. 관련 연구

2.1 웹소켓

웹소켓은 웹 서버와 웹 브라우저 사이에 하나의

** 교신 저자

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발사업의 일환으로 수행하였음. [2015(1711021663), 실감체험형 콘텐츠 기반 스마트 스트리트 구현 기술 개발]

TCP(Transfer Control Protocol) 포트를 생성하여 실시간 전이중(full-duplex) 통신을 지원하는 프로토콜이다.

웹소켓의 프로토콜은 2011 년 IETF(Internet Engineering Task Force)에서(IETF RFC 6455) 표준화 되었고, API 는 W3C(World Wide Web Consortium)에서 표준화가 완료되었다. 웹 소켓을 사용하여 기존 웹 아키텍처의 반이중(half-duplex) 방식에서 하나의 포트만으로 동시에 양방향 송수신이 가능한 전이중 방식의 구현이 가능하고, 기존의 HTTP 에서 수 KB 에 이르던 불필요한 헤더 오버헤드의 양이 최초의 핸드쉐이크 과정에서만 존재하고 이후에는 2byte 의 웹소켓 프레임을 사용하여 통신의 효율성을 향상시킬 수 있다[6][7].

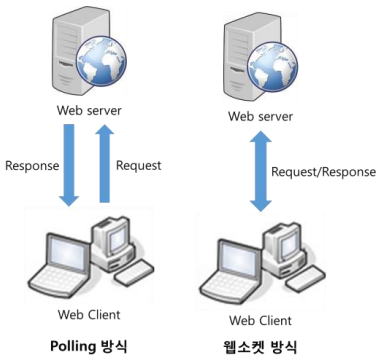


그림 1 Polling 방식과 웹소켓 방식의 비교

본 논문에서 제시하는 메시지 플로팅 시스템에서는 상호운용성을 지원하기 위해 모든 콘텐츠는 웹 기반으로 제공되기 때문에 웹소켓 프로토콜을 채택하여 단일 포트를 통해 디지털 사이니지와 사용자 모바일 단말 등 스마트 환경에 존재하는 이기종 디바이스들로 요청받은 콘텐츠를 전달할 뿐 아니라, 디바이스로부터 상태 정보 및 요청 이벤트를 수신할 수 있다. 또한, 다수의 콘텐츠 요청이 짧은 시간에 집중될 경우에도 오버헤드가 적게 발생하여 일정 수준의 통신 성능을 유지할 수 있다.

2.2 디지털 사이니지 기술 연구

디지털 사이니지란 네트워크를 통해 원격제어가 가능한 디지털 디스플레이를 공공장소나 상업공간에 설치하여 정보, 엔터테인먼트, 광고 등을 제공하는 디지털 미디어를 뜻한다. 단순한 디지털 정보를 보여주는 디지털 정보 디스플레이가 아니라 다양한 IT 및 콘텐츠 관련 기술과 융합하여 양방향 커뮤니케이션이 가능한 정보매체이다[8].

현재 디지털 사이니지 기술들이 다양하게 연구 되고 있으며, 이기종 디바이스를 활용한 디지털 사이니지들은 다양한 콘텐츠를 위해 적용되어 있다.

[9,10,11]은 디지털 사이니지를 사용하기 위해 사용자가 스마트 디바이스로 특정 움직임을 수행하면 주변에 위치한 무선 공유기를 통해 디지털 사이니지에 전송해주는 시스템이다. 이 시스템들을 통하여 다수의 스마트 디바이스에 콘텐츠를 제공할 수 있다.

[12,13]은 디지털 사이니지의 콘텐츠를 주변에 있는 이기종 디바이스로 수신할 때 동기화를 시켜 동일 콘텐츠를 공유하는 시스템을 제시하였다.

[14]는 향후 디지털 사이니지의 발전 방향과 비즈니스적 가치를 분석하였다. 그리고 interactive 디지털 사이니지를 위한 개념적인 프레임워크 및 요구사항을 제시하였다. 디지털 사이니지는 분석한 소비자의 context 를 분석하여 어떤 광고를 표출할 지 결정하게 되고, 광고는 폴링 방식을 통해 서버에서 전달되는 전략을 제시하였다. 하지만 개념적인 수준의 제시였고, 실질적인 사례 연구는 수반되지 않았다. 또한, 데이터의 전달에서 폴링 방식을 채택하여 데이터 트래픽이 많아졌을 때, 심각한 성능 저하를 야기할 수 있다.

이와 같이 디지털 사이니지 기술들은 다양하게 연구되고 발전되어 디지털 사이니지와 사용자간의 커뮤니케이션 할 수 있는 환경들이 등장하고 있다. 하지만 사용자들에게 지속적으로 콘텐츠를 제공하기 위한 방법과, 디지털 사이니지를 효율적으로 제어하는 방법이 필요하다.

본 논문에서는 시스템을 통해서 디지털 사이니지와 사용자를 매핑하여 콘텐츠를 제공하고, 사용자가 별도의 인증 없이 디지털 사이니지를 제어할 수 있는 시스템을 제공한다.

3. 메시지 플로팅 시스템

메시지 플로팅 시스템은 실외 스마트 환경의 디지털 사이니지를 통해 사용자들에게 정보, 엔터테인먼트, 광고 등과 같은 콘텐츠를 웹소켓을 통해 제공하고, 사용자 단말로부터 디지털 사이니지에 표출되는 콘텐츠를 웹소켓 통해 제어할 수 있는 서비스를 제공한다. 즉, 메시지 플로팅이란 콘텐츠를 디지털 사이니지에서 사용자 단말로 전송하는 방법인 콘텐츠 플로팅과 사용자 단말에서 디지털 사이니지의 콘텐츠를 제어하기 위해 이벤트를 전송하는 방법인 이벤트 플로팅을 총칭하여 ‘콘텐츠와 이벤트가 디바이스에서 디바이스로 떠다닌다’ 는 의미이다.

3.1 절에서는 본 논문에서 제안하는 메시지 플로팅 시스템에서 디바이스와 사용자간의 관계 매핑을 위한 데이터베이스 설계에 대해 설명한다. 3.2 절에서는 메시지 플로팅 시스템의 전체적인 프로세스에 대해 설명한다. 그리고 3.3 절에서는 메시지 플로팅 시스템의 아키텍처에 대해 설명한다. 3.4 절에서는 웹소켓을 통해 데이터들을 전달하는 인터페이스에 대해 설명한다.

3.1 데이터베이스 설계

메시지 플로팅 시스템에서 디지털 사이니지와 이를 사용하는 사용자를 시스템 내에서 논리적으로 매핑 시키기 위해 관계형 데이터베이스로 설계하였고, 시스템의 기반이 되는 다양한 정보들을 저장할 수 있도록 설계하였다.

그림 2는 콘텐츠 플로팅 시스템을 위해 설계한 데이터베이스 스키마이다.

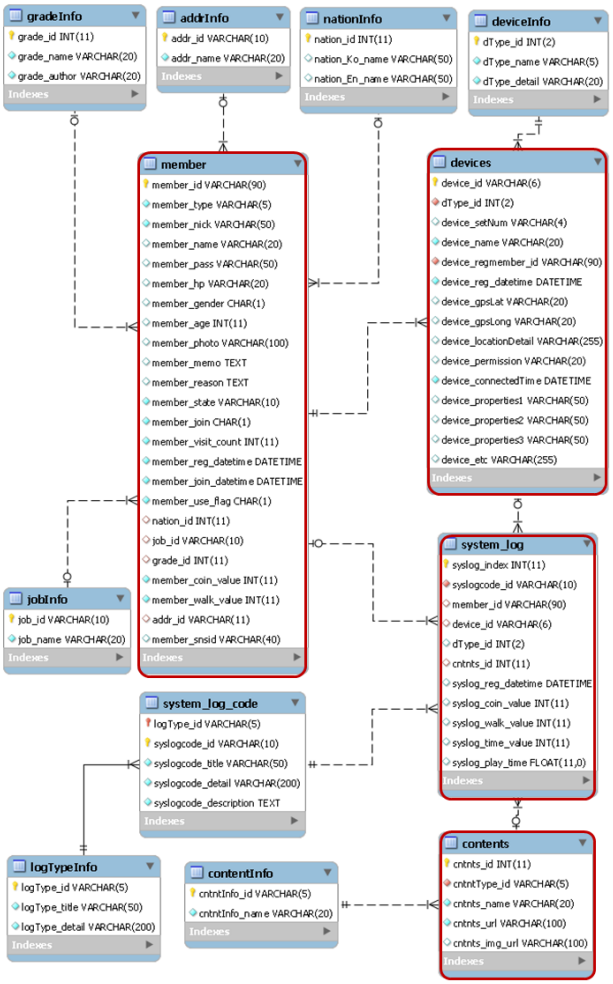


그림 2 메시지 플로팅 시스템 데이터베이스 스키마

테이블은 크게 4가지로 구성되어있고, 나머지 테이블들은 데이터베이스 내에서 같은 정보가 2번 이상 기록되지 않도록 각각의 정보가 저장되어있는 테이블로 구성되어 있다.

4가지 테이블의 구성은 다음과 같다.

- member 테이블: 기본적으로 메시지 플로팅 시스템을 이용하는 사용자 정보를 저장하고 있다.
- devices 테이블: 스마트 환경에 설치된 이기종 디바이스의 디지털 사이니지, 비콘 등 환경에 필요한 디바이스들의 정보를 저장하고 있다.
- contents 테이블: 디지털 사이니지로 사용자들에게 제공될 정보, 엔터테인먼트, 광고 등과 같은 콘텐츠의 정보를 저장하고 있다. 메시지 플로팅 시스템의 모든 콘텐츠들은 상호운용성을 보장하기 위해 웹 기반으로 개발되어 제공되기 때문에 시스템 내에서 콘텐츠 관리 방식은 URL 주소를 저장하여 관리한다.
- system_log 테이블: 사용자가 메시지 플로팅 시스템을 사용했던 모든 이력을 저장하고 있다. 이는 콘텐츠 플로팅에 사용될 테이블이다.

데이터 베이스 설계를 통해 디지털 사이니지 디바이스와 비콘을 매핑하기 위해서 devices 테이블의 속성인 device_setNum 속성의 값을 동일하게 저장하여 하나의 세트로 관리하게 된다. 이를 기반으로 사용자 모바일 단말과 디지털 사이니지를 시스템 내에서 연결할 수 있다. 사용자가 디지털 사이니지로 접근을 했을 때 사용자의 단말에서 비콘 정보를 수신하여 메시지 플로팅 시스템으로 웹소켓을 통해 전달하게 된다. 시스템 내에서는 해당 비콘 정보의 device_setNum 과 동일한 디지털 사이니지 장비를 찾아서 사용자와 웹소켓 연결을 해주는 구조이다.

3.2 메시지 플로팅 프로세스

메시지 플로팅 시스템에서는 웹소켓을 사용하여 디지털 사이니지와 사용자를 연결을 시켜준다. 시스템의 구조는 메시지 플로팅 시스템은 Server 이고, 디지털 사이니지와 사용자의 단말은 Client 이다. 그러나 디지털 사이니지와 사용자의 모바일 단말의 연결이 Client-to-Client 가 아닌 각자의 Session 을 가진 Server-to-Client 의 연결을 통해 시스템 내에서 연결된 Session 을 매핑하여 전송된 데이터를 해당 Clinet 로 전송해주게 되는 구조이다.

그림 3은 사용자 단말과 디지털 사이니지의 웹소켓 매핑 관계 및 데이터 전달 과정을 나타낸 구조이다.

그림 3 사용자 단말과 디지털 사이니지의 웹소켓 매핑 관계 및 데이터 전달 과정

디지털 사이니지는 메시지 플로팅 시스템이 구동됨과 동시에 웹소켓을 통해 고유의 Session 을 가지고 지속적으로 연결되어 있는 상태이다. 그리고 그림 3 과 같이 사용자 모바일 단말이 시스템 내에서 식별되었을 때 웹소켓을 통해 연결되어 고유의 Session 을 가지게 되고, 시스템에서 해당 디지털 사이니지의 Session 을 찾아 사용자의 Session 과 매핑을 시켜주는 구조이다. 메시지 플로팅 시스템에서 디지털 사이니지 또는 사용자 단말로 데이터를 전송할 때는 JSON 구조로 전달하고, 시스템 내에서는 Parsing 된 데이터로 전달한다.

3.1.1 절에서는 콘텐츠 플로팅에 대한 프로세스를 설명하고, 3.1.2 절에서는 이벤트 플로팅에 대한 프로세스를 설명한다.

3.2.1 콘텐츠 플로팅

콘텐츠 플로팅은 메시지 플로팅 시스템의 하나의 방법으로 디지털 사이니지에 콘텐츠를 웹소켓을 통해 전달하거나, 사용자가 디지털 사이니지에서 콘텐츠를 이용하다가 그 구역을 벗어났을 때 사용자의 모바일 단말에서 이어서 콘텐츠를 제공받을 수 있는 방법이다.

그림 4 는 콘텐츠 플로팅 과정을 가시적으로 표현한 것이다.

그림 4 콘텐츠 플로팅

1. 메시지 플로팅 시스템에서 'A' 디지털 사이니지를 통해 사용자에게 콘텐츠를 제공 중이다.
2. 콘텐츠를 제공받던 사용자는 'A' 디지털 사이니지를 벗어나면 사용자가 제공받던 콘텐츠의 기록을 메시지 플로팅 시스템에 전송하여 저장한다.
3. 사용자가 'A' 디지털 사이니지에서 제공받던 콘텐츠를 자신의 모바일 단말로 이어서 제공받을 수 있다. 예를 들어 동영상 콘텐츠를 'A' 디지털 사이니지에서 5 분간 시청하다가 자리를 벗어났을 때 사용자 자신의 모바일 단말로 이어서 5 분부터 동영상을 시청할 수 있다.
4. 모바일 단말로 지속적으로 제공받던 콘텐츠를 다시 이어서 디지털 사이니지에서 제공 받기 위해 'B' 디지털 사이니지로 접근한다. 사용자가 접근을 하면 디지털 사이니지에 부착되어있는 비콘 정보를 모바일 단말에서 수신하여 메시지 플로팅 시스템으로 전송한다.
5. 전송받은 데이터를 분석하여 사용자의 이력 정보를 메시지 플로팅 시스템에서 파악하여 'B' 디지털 사이니지로 콘텐츠를 실행시켜준다.

3.2.2 이벤트 플로팅

이벤트 플로팅은 메시지 플로팅 시스템의 하나의 방법으로, 기존의 터치 스크린으로 디지털 사이니지를 제어하는 방법을 대신해 모바일 단말로 디지털 사이니지를 제어를 하는 방법이다.

사용자가 디지털 사이니지를 제어하기 위해 모바일 단말에서 이벤트를 메시지 플로팅 시스템으로 전송하여 해당 디지털 사이니지에서 입력 받은 이벤트를 실행하게 된다.

그림 3 과 같은 데이터 전달 과정으로 사용자로부터 입력 받은 이벤트를 디지털 사이니지로 전달하게 된다.

그림 5 는 이벤트 플로팅 과정을 가시적으로 표현한 것이다.

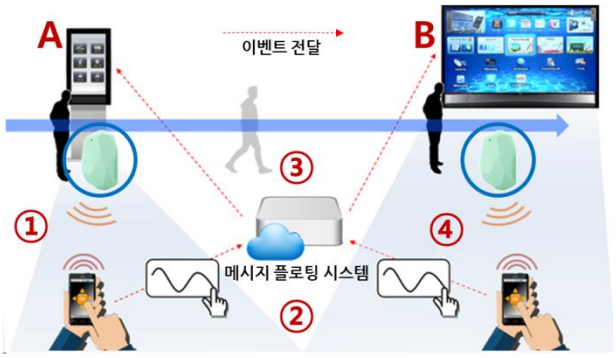


그림 5 이벤트 플로팅

1. 사용자는 디지털 사이니지에서 제공되는 콘텐츠를 이용하기 위해 'A' 디지털 사이니지로 접근하면 'A' 디지털 사이니지와 매핑되어 있는 비콘의 정보를 모바일 단말기에서 수신한다.
2. 모바일 단말로 수신 받은 비콘 정보를 메시지 플로팅 시스템과 웹소켓 연결을 통해 해당 디지털 사이니지와 연결되고, 사용자 모바일 단말에는 제어 기능을 가진 애플리케이션이 실행이 된다.
3. 사용자 모바일 단말에서 입력되는 이벤트를 매핑된 디지털 사이니지로 이벤트를 전달해서 실행하게 된다.
4. 사용자는 'B' 디지털 사이니지와 같은 다른 종류의 디지털 사이니지라도 위와 같은 이벤트 플로팅 방법으로 제어가 가능하다. 즉, 디지털 사이니지의 플

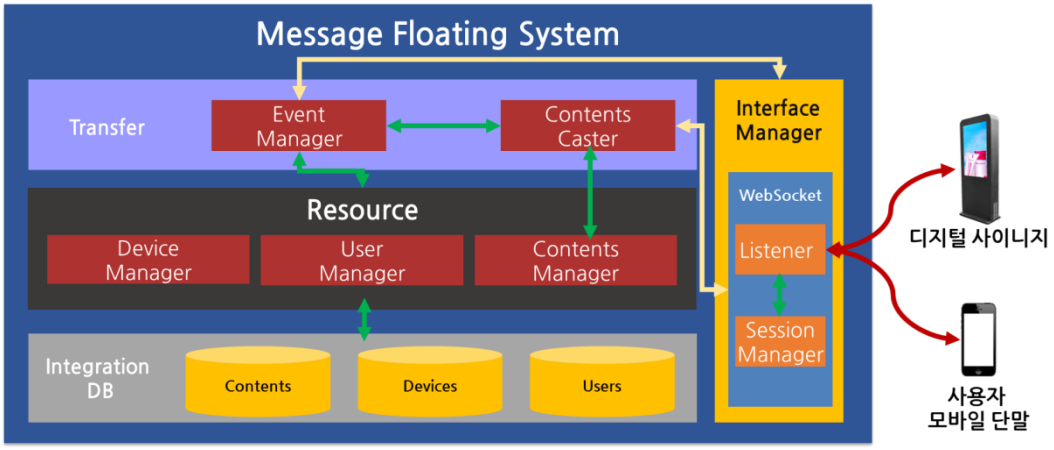


그림 6 메시지 플로팅 시스템 아키텍처

랫폼과 디바이스의 종류가 다르더라도 제안하는 이벤트 플로팅 방법을 통해 제어가 가능하다.

3.3 메시지 플로팅 시스템 아키텍처

메시지 플로팅 시스템을 구현하기 위해 개발 단계에서 발생하는 요구 사항의 변경 등의 위험을 최소화하기 위해 아키텍처를 설계하였다. 그림 6 과 같이 총 4 개의 레이어로 구성되어있고, 각 레이어 별 기능을 담당하는 모듈단위로 구성하였다. 4 가지 레이어의 구성은 다음과 같다.

- **Transfer 레이어** : Event Manager 모듈과 Contents Caster 모듈을 포함하는 이벤트와 콘텐츠 전송에 관련된 것이다.
  Event Manager 는 사용자 모바일 단말로부터 전송되어 오는 이벤트를 식별하고 제어해야 할 디지털 사이니지 찾아주는 모듈이다.
  Contents Caster 는 디지털 사이니지에 실행할 콘텐츠를 찾거나, 사용자에게 지속적인 콘텐츠 서비스를 제공할 때 사용되는 모듈이다.
- **Resource 레이어** : Resource 레이어에 포함된 각각의 Manager 모듈은 IntegrationDB 레이어에 있는 데이터베이스에 접근해서 데이터를 관리하는 모듈을 모아둔 레이어이다.
- **IntegrationDB 레이어** : 메시지 플로팅 시스템에서 존재하는 다양한 디바이스(디지털 사이니지, 비콘 등)의 정보와 디지털 사이니지로 통해 제공되는 콘텐츠의 정보, 메시지 플로팅을 사용하는 사용자들의 정보를 저장하는 레이어이다.
- **Interface 레이어** : Listener 모듈은 Client 가 웹소켓을 통해 시스템으로 연결을 할 수 있게 수신하고 있는 모듈이다.
  Session Manager 는 Client 들이 웹소켓을 통해 접속을 했을 때 모바일 단말과 디지털 사이니지를 Session 매핑을 해주는 모듈이다.

3.4 메시지 인터페이스

디지털 사이니지와 모바일 단말간의 웹소켓을 통해 콘텐츠 및 이벤트 전달을 위한 메시지 인터페이스를 표 1 과 같이 정의하였다.

표 1 메시지 인터페이스 정의

구분	메시지 코드	메시지 구조(JSON)	
사용자 식별 (비콘)	BC0001	Device->Server	Server->Device
		{ "Code": "", "memId": "", "UUID": "", "majorID": "", "minorID": "" }	{ "Suc": "" }
콘텐츠 플로팅	CT0001	Device->Server	Server->Device
		{ "Code": "", "memId": "" }	{ "Suc": "", "cntURL": "", "syslog": "" }
이벤트 플로팅	CT0002	Device->Server	Server->Device
		{ "Code": "", "Input": "" }	{ "Ctrl": "" }

메시지 플로팅 시스템에서 메시지의 타입을 식별하기 위해 메시지 코드를 정의하였고, 시스템에서는 메시지 코드를 식별하여 정의된 코드에 따라 사용자 식별(비콘 식별), 콘텐츠 플로팅, 이벤트 플로팅의 기능을 수행한다.

사용자 식별(비콘)은 사용자 단말에서 수신된 비콘의 정보(UUID, MajorID, MinorID)로 같은 구역에 있는 디지털 사이니지와 사용자 단말의 웹소켓 세션 매핑을 시켜준다.

콘텐츠 플로팅은 디지털 사이니지에서 제공받던 콘텐츠를 사용자 단말로 전달하거나, 사용자 단말에서 제공받던 콘텐츠를 디지털 사이니지로 전달해 준다.

메시지 플로팅은 사용자 단말로 디지털 사이니지를 조작하기 위해 입력된 이벤트를 전달해준다.

메시지 플로팅 시스템에서는 정의된 인터페이스에 따라 모든 데이터는 JSON 구조로 전달된다.

4. 메시지 플로팅 시스템 구현

3.2 절에서는 콘텐츠 플로팅 방법과 이벤트 플로팅 방법을 설명하였고, 3.1 절, 3.3 절에서 메시지 플로팅 시스템의 설계에 대해서 설명하였다. 본 장에서는 이를 기반으로 실제 시스템을 구현하고 실행 결과에 대하여 설명한다.

4.1 절에서는 시스템의 핵심이 되는 메시지 플로팅 서버 구현에 대해 설명한다. 4.2 절에서는 디지털 사이니지를 제어하기 위한 사용자 모바일 단말의 애플리케이션에 대해 설명한다. 4.3 절에서는 시스템 내에서 사용될 메인 콘텐츠에 대해 설명한다. 4.4 절에서는 메시지 플로팅 결과에 대해 설명한다.

4.1 메시지 플로팅 서버 구현

시스템의 핵심이 되는 서버는 3.3 절에서 설명한 아키텍처를 기반으로 자바의 웹 서비스로 구현해서 Glassfish 서버에서 실행되도록 구현하였다.

서버를 구동시키면 데이터 베이스에 등록된 모든 디지털 사이니지는 웹소켓을 통해 서버와 연결되고 각각의 고유의 Session ID 를 가진다.

그림 7 은 메시지 플로팅 서버 구동 화면이다.

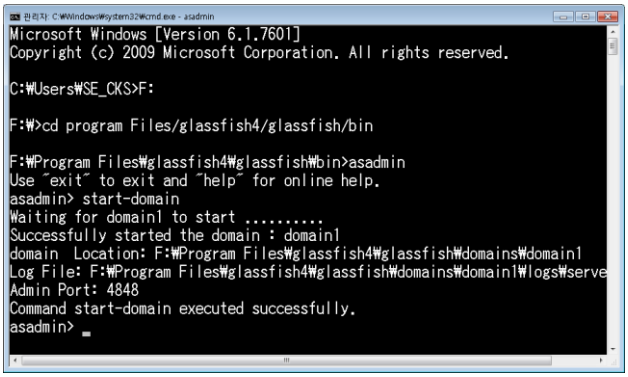


그림 7 메시지 플로팅 서버 구동 화면

4.2 모바일 애플리케이션

모바일 애플리케이션은 사용자가 디지털 사이니지에서 제공하는 콘텐츠를 제어하기 위한 애플리케이션이고, 상호운용성을 지원하기 위해 플랫폼, OS 에 제약이 없는 Hybrid-App 로 구현하였다. 비콘 정보를 수신하기 위해서는 백그라운드 서비스로 안드로이드 젤리빈(4.3) 버전으로 구현하였다.

사용자가 콘텐츠를 이용하기 디지털 사이니지로 접근하게 되면 설치되어 있는 비콘의 정보를 모바일 애플리케이션이 수신한다. 비콘의 정보가 수신되면 메시지 플로팅 서버에 웹소켓으로 연결을 하고 제어할 수 있는 컨트롤러 화면이 실행된다.

그림 8 은 모바일 애플리케이션에서 수신한 비콘의 정보(UUID, major, minor)이고, 그림 9 은 사용자 모바일 애플리케이션 실행 화면이다.

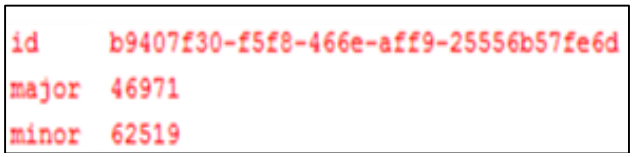


그림 8 비콘 수신 정보(UUID, major, minor)

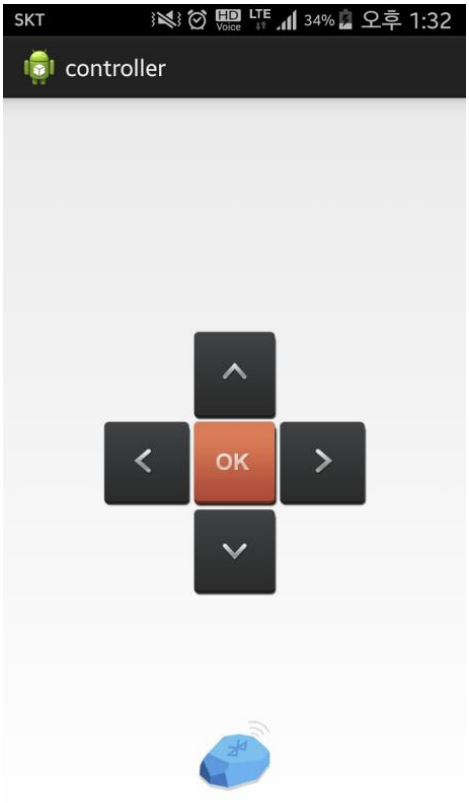


그림 9 사용자 모바일 애플리케이션 실행 화면

그림 9 와 같이 상, 하, 좌, 우, 확인 기능을 가진 버튼들로 구성되어있다. 사용자의 입력에 따라 해당 이벤트가 메시지 플로팅 서버로 전달하여 매핑되어 있는 디지털 사이니지의 Session 을 찾아 이벤트를 전달하여 동작하게 한다.

모바일 애플리케이션을 사용하면 기존의 터치 스

크린 디바이스를 사용하지 않아도 사용자는 자신의 모바일 단말로 제어를 할 수 있다.

4.3 메인 콘텐츠

메인 콘텐츠는 디지털 사이니지에 기본으로 실행되는 콘텐츠이다. 사용자는 모바일 애플리케이션으로 메인 콘텐츠를 제어할 수 있다.

그림 10 에 하단의 디지털 사이니지에 기본적으로 실행되는 메인 콘텐츠 구현하여 실행한 화면이다. 기본적으로 가운데 콘텐츠가 실행되고 모바일 애플리케이션을 통해 좌, 우 이벤트를 전달받아 이전 콘텐츠, 다음 콘텐츠로 이동할 수 있다.

그림 10 의 하단에 실행되어 있는 메인 콘텐츠의 실제 화면 구성은 상단에 있는 ①, ②, ③, ④와 같이 콘텐츠가 여러 개 연결되어 있는 방식이다. 그러나 화면상에는 하나의 콘텐츠만 출력이 되고, 이벤트를 전달받았을 때 ①->②->③->④->①과 같이 슬라이드 방식으로 좌, 우로 넘어가는 방법으로 구현하였다.

메인 콘텐츠는 자바스크립트를 사용하여 구현하였다.

그림 10 메인 콘텐츠 실행 화면과 실제 화면 구성

4.4 메시지 플로팅 결과

그림 11, 그림 12, 그림 13 는 4.1 절, 4.2 절, 4.3 절에서 구현한 결과로 데이터의 전달 과정을 가시적으로 나타낸 그림이다.

사용자가 디지털 사이니지 영역 내로 진입했을 때 메시지 플로팅 서버에서 사용자를 식별해야만 한다. 이때 사용자 단말에서 비콘 정보를 수신하고 디지털 사이니지 서버로 정의된 인터페이스에 따라 메시지를 전송하게 되고, 서버는 사용자를 식별하게 된다. 서버에서는 해당하는 디지털 사이니지와 사용자 모바일단말을 서로 매핑 시켜주고 그림 11 과 같이 결과 값을 전송해준다.

그리고 사용자가 디지털 사이니지 영역을 벗어났을 때 서버에서 현재 실행중인 콘텐츠를 그림 12 와 같이 사용자 모바일 단말로 콘텐츠를 전송한다.

그림 12 에서 cntURL 은 제공받던 콘텐츠 URL 을나타내고, syslogplayTime 은 제공받던 콘텐츠가 동영상일 경우 시간을 전송해준다.

```
{"Code":"CT0001","devId":"MW0001"}
[object MessageEvent]
*MW0001 접속 1a5e6dda-95f6-4ea9-86ee-66232ec72de6
[object MessageEvent]
*apppnuselab 접속 33f235ab-1bab-492c-9a15-9dd5b6926151
[object MessageEvent]
{"Suc":"Success"}
```

그림 11 사용자 식별 결과 화면

(디지털 사이니지와 모바일 단말의 매핑 결과)

```
{"Suc":"Success","cntURL":"http:₩/₩/app.smartstreet.co.kr₩/display₩/","syslogplayTime":0.0}
```

그림 12 콘텐츠 플로팅의 메시지 결과 화면

그림 13 은 모바일 애플리케이션에서 사용자가 입력한 이벤트를 서버로 전달하여 디지털 사이니지를 제어하는 결과 화면이다. 이벤트를 입력할 때 마다 정의된 인터페스에 따라서 메시지가 서버를 통해 디지털 사이니지로 전달된다.

```
정보 : WebSocket 오픈!
[object MessageEvent]
*MW0001 Open! 4c8ff264-6418-48c2-b637-108f81336766
[object MessageEvent]
*apppnuselab Open! 54913dba-bdb1-4a28-9ac0-957656918598
[object MessageEvent]
{"Ctrl":"Left"}
[object MessageEvent]
{"Ctrl":"Right"}
```

그림 13 이벤트 플로팅의 메시지 결과 화면

5. 결론 및 향후 연구

본 논문에서는 실외 스마트 환경의 디지털 사이니지를 통해 사용자들에게 콘텐츠를 끊김 없이 지속적으로 제공하는 방법과 디지털 사이니지에 표출되는 콘텐츠를 사용자 단말로 제어할 수 있는 방법을 적용한 메시지 플로팅 시스템을 제안하였다.

실외의 스마트 환경은 컨벤션 센터, 박물관, 유동 인구가 많은 거리, 관광지 등에 구성될 수 있다.

이러한 환경이 구성되고 메시지 플로팅 시스템을 적용하면 단순 정보만을 보여주는 디지털 정보 디스플레이가 아니라 다양한 IT 기술 및 콘텐츠들을 접목하여 실외 환경에서도 많은 사람들에게 유용한 정보, 재미, 나아가 편리함까지 줄 수 있는 환경을 구성 할 수 있다.

향후 연구로는 사용자와 디지털 사이니지간의 웹소켓 연결을 1:1 관계로 개인이 점유하는 형태로 제공하지만, 1:N 관계로 다수의 사용자가 함께 사용할 수 있는 방법에 대해 연구할 것이고, 또한 다양한 콘텐츠를 적용하여 메시지 플로팅 시스템에 대한 효용성에 대해서 검증할 것이다.

참고문헌

[1] 박호진, 박준희, “스마트홈 국제표준화 동향 및 추진 전략”, 한국통신학회지, 2014, pp. 72-79

[2] 김항영, 김승곤, 이창건, “스마트홈의 메소드 오프로딩을 통한 리소스 쉐어링”, 한국정보과학회, 2014, pp. 6-7

[3] H Rodrigues, R José, “System Implications of Context-Driven Interaction in Smart Environments,” Interacting with Computers, 2013

[4] I Chun, J Park, H Lee, W Kim, S Park, E Lee, “An agent-based self-adaptation architecture for implementing smart devices in smart space,” Telecommunication Systems, 2013, pp. 2335-2346

[5] 김찬, “디지털 사이니지 기술 현황 및 전망”, 2013, pp. 61-68

[6] Five Signs You Need HTML5 WebSockets – Peter Lubbers, “http://peterlubbers.sys-con.com/node/1551694”

[7] 남궁란, “비동기 웹 스크립트 기반의 통신 처리 속도 성능분석”, 중앙대학교, 컴퓨터 소프트웨어학과 인터넷전공, 석사학위논문, 2012

[8] 채송화, “디지털 사이니지 기반 콘텐츠산업의 현황과 전망”, 코카포커스 6 호(통권 54 호), 2012

[9] J. She, J. Crowcroft, H. Fu and P. H. Ho, "Smart Signage: A Draggable Cyber-Physical Broadcast/Multicast Media System, " Green Computing and Communications (GreenCom), IEEE International Conference on, 2012, pp.468-476

[10] Y. G. Park, Y. G. Nam, “Determining a Minimum Initial Delay Time for Download & Seamless Playback of Multimedia Contents on Network Digital Signage,” Journal of the Korea Industrial Information System Society, 2012, Vol. 12, No. 2, pp. 33-43

[11] K. H. Ro, H. Y. Hwang and S. C. Kim, "A Research on Personalized Mobile Advertising Service using the Linkage between Digital Signage and Smartphones," The Journal of the Institute of Webcasting, Internet and Telecommunication, 2014, Vol. 14, No. 1, pp.139-146

[12] J. S. Lee, J. W. Lee and K. S. Yoon, “A Novel Digital Signage Content Propagation Method using a Smart-phone Application,” Journal of Advanced Information Technology and Convergence, 2013, Vol. 3, No. 2, pp.13-23

[13] S. H. Lee, S. Y. Kim, “Design and Implementation of an Intelligent System for Personalized Contents Recommendation on Smart TVs,” Journal of the Korea Industrial Information System Society, 2013, Vol. 18, No. 4, pp. 73-79,

[14] C Bauer, P Dohmen, C Strauss, “Interactive Digital Signage – an Innovative Service and its Future Strategies”, International Conference on Emerging Intelligent Data and Web Technologies, 2011

Automotive 소프트웨어 Security 향상을 위한 Coding Guideline 에 대한 연구

우충기*, 정지현**, 민상윤***

*㈜ 삼성전자, KAIST 소프트웨어대학원
**㈜ 만도, KAIST 소프트웨어대학원
***KAIST 전산학과
{chungki.woo, jihyun0235, symin}@kaist.ac.kr

요약: 수백, 수천만 라인의 소스코드로 동작하며 원거리 무선통신기술이 탑재되는 Connected Car 의 등장 및 시장의 확대로 인해 Automotive 소프트웨어의 Security 중요성이 크게 증가하고 있다. Automotive 와 관련된 최근까지의 이슈는 주로 Safety 쪽에 집중되어 있어 이에 대한 많은 연구들이 주로 수행되었던 반면 소프트웨어 Security 에 관련된 연구는 부족한 실정이고 이에 대한 관심도 적어 많은 업계 소프트웨어 개발자들이 Security 에 대한 중요성을 간과한 채 개발을 진행 하고 있다. 본 논문에서는 Automotive 소프트웨어 개발 시 업계에서 사용하던 Coding Guideline 을 알아보고 이를 기존의 Secure Coding Guideline 과 비교분석 후 Security 측면에서의 취약점을 찾아 개선 할 수 있는 방향을 제시 하도록 한다. 이를 통해 Automotive 소프트웨어 Security 를 향상 시키고 개발자의 Automotive Security 에 대한 인식변화에 기여하고자 한다.

핵심어: Automotive, Connected Car, Security, Coding Guideline, CERT, MISRA

1. 서론

소프트웨어와 관련된 Security 이슈들은 오래 전부터 있어왔고 현재도 진행 중이다. 최근에 발생한 Heartbleed 버그[1]의 경우를 보면 그 심각성을 알 수 있다. 인기 있는 암호화 관련 소프트웨어 라이브러리인 OpenSSL 의 Security 취약점을 이용해 이를 사용하는 시스템(보안이 중요한 이메일 서비스, 거대 포털 사이트, 금융권 등)의 메모리를 누구나 읽어 들여 그곳에 저장된 중요한 정보들, 예를 들어 개인 암호화 관련 Key, 패스워드 등을 쉽게 획득 할 수 있는 치명적인 소프트웨어 Security 결함 이었다.

최근의 Automotive 는 내부에 Engine Control Unit, Airbag Control Unit, Infotainment System 등 많은 ECU(Electronic Control Unit) 를 가지고 있으며 이들은 수백, 수천만 라인의 소프트웨어 코드에 의해 구동 된다. 이렇게 Automotive 시스템에서 소프트웨어가 차지 하는 비중이 점점 더 커지고 있으며 최근에는 Connected Car 의 등장과 시장의 확대[2]로 인해 Automotive 에서의 소프트웨어 Security 는 더 이상 무시 할 수 없는 큰 이슈가 되고 있다.

Connected Car 란 대체적으로 무선통신기술을 이용해 차 외부와의 접속 및 통신 기능을 갖춘 것을 의미한다. 원래부터 Automotive 에 대한 다양한 공격 루트가 있어 왔지만(Figure 1), Connected Car 가 특히나 Hacking 공격의 대상이 되기 쉬워진 이유는 이런 원거리 무선통신 기술을 통해 익명의 공격자로부터 차량으로의 접근이 쉽게 가능해 졌기 때문이다. 실제로 최근 Jeep 해킹 사례[4]를 통해 Connected Car 에 대한 Security 취약점을 누군가 악의적으로 이용했을 때의 심각성을 쉽게 확인 할 수 있다. 원격으로 공격을 수행하여 차량의 Air conditioner, Audio system, Wiper, Door Lock 등 에서부터 탑승자의 Safety 를 직접적으로 위협 할 수 있는 Engine, Steering Wheel, Break System 에 대한 공격을 쉽게 하는 모습을 통해 Automotive 에서의 Security 이슈는 결국 심각한 Safety 이슈로 이어지게 되고 최악의 경우엔 사람의 목숨을 앗아 갈 가능성이 있음을 눈으로 확인 할 수 있다. 하지만, 이런 Security 의 중요성에도 불구하고 Automotive 업계에서 개발 시 적용되어 사용중인 ISO 26262(자동차 기능 안정성 국제 표준 프로세스)[5]는 에서는 별도의 Security 확보 관련 프로세스를 제공하고 있지 않다.

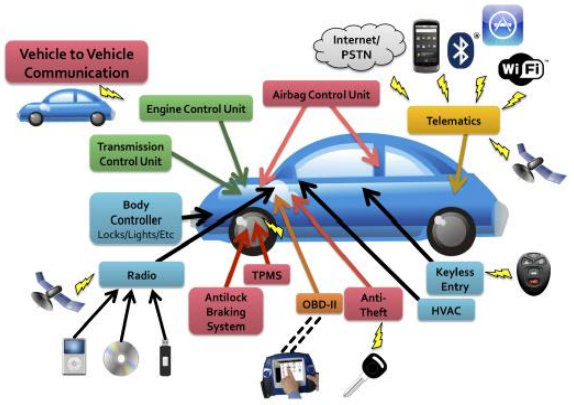


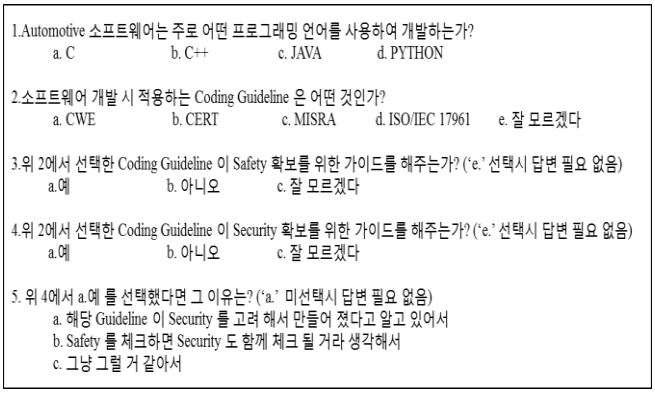
Figure 1. 자동차에 대한 공격 루트[3]

이런 상황에서 소프트웨어를 개발하는 개발자 입장에서는 Security 를 필히 잘 고려하여 개발하여야 할 필요가 있다. 이를 쉽게 하기 위해 본 논문에서는 우선적으로 Coding Guideline 에 대한 Security 향상을 위한 연구를 수행하였다. Automotive 업계에서 사용중인 Coding Guideline 과 기존의 다른 Coding Guideline 들을 비교하여 Security 취약점을 발견하고 이를 위한 Coding Guideline 개선 방향을 제시한다.

본 논문은 다음 순서로 기술된다. 2 장에서는 우선적으로 현재 Automotive 업계에서 소프트웨어 개발 시 적용하고 있는 Coding Guideline 을 확인하고 소프트웨어 Security 와의 연관성에 대한 개발자의 인식을 확인한다. 3 장에서는 현존하고 있는 Secure Coding Guideline 들의 종류를 확인하고 4 장에서는 2 장에서 확인한 Automotive 업계에서 사용중인 Coding Guideline 과 기존 Secure Coding Guideline 들을 비교한 연구결과들을 소개한다. 5 장에서는 Security 향상을 위한 Coding Guideline 을 제시하고 6 장에서는 본 논문에 대한 결론을 내린다.

2. Automotive Coding Guideline

Automotive 업계 에서 현재 사용중인 Coding Guideline 를 확인하고 이것과 소프트웨어 Security 와의 연관성에 대한 개발자의 인식을 국내 자동차 관련 업체인 A 사에서 간단한 설문조사를 통해 확인해 보았다. Figure 2 는 설문지의 내용이다.


1.Automotive 소프트웨어는 주로 어떤 프로그래밍 언어를 사용하여 개발하는가?
a. C b. C++ c. JAVA d. PYTHON

2.소프트웨어 개발 시 적용하는 Coding Guideline 은 어떤 것인가?
a. CWE b. CERT c. MISRA d. ISO/IEC 17961 e. 잘 모르겠다

3.위 2에서 선택한 Coding Guideline 이 Safety 확보를 위한 가이드를 해주는가? ('e.' 선택시 답변 필요 없음)
a.예 b. 아니오 c. 잘 모르겠다

4.위 2에서 선택한 Coding Guideline 이 Security 확보를 위한 가이드를 해주는가? ('e.' 선택시 답변 필요 없음)
a.예 b. 아니오 c. 잘 모르겠다

5. 위 4에서 a.예 를 선택했다면 그 이유는? ('a.' 미선택시 답변 필요 없음)
a. 해당 Guideline 이 Security 를 고려 해서 만들어 졌다고 알고 있어서
b. Safety 를 체크하면 Security 도 함께 체크 될 거라 생각해서
c. 그냥 그럴 거 같아서

Figure 2. 설문지

본 설문 조사 결과를 통해 확인 할 수 있는 사실은 다음과 같다.

1. Automotive 소프트웨어 개발 시 주로 사용하는 언어 : C Language
2. Automotive 소프트웨어를 개발 시 사용하는 Coding Guideline : MISRA(Motor Industry Software Reliability Association) C
3. 현재 사용되는 Coding Guideline 으로 Security 확보가 가능하다고 개발자들은 믿고 있음

이들 중 3 번 Automotive 소프트웨어 개발 시 MISRA C Coding Guideline 이 소프트웨어 Security 확보를 위한 가이드를 해준다는 개발자의 인식에 문제가 없는지를 확인하기 위해 본 논문에서는 우선적으로 MISRA C 와 더불어 기존의 잘 알려진 Coding Guideline 들에 대해 알아보고 이들 간의 비교연구를 통해 개발자 인식의 옳고 그름을 판별하고자 하였다.

3. MISRA C 와 다른 Coding Guideline

여기서는 MISRA C 를 포함하여 기존 Coding Guideline 들 중 몇몇 대표적인 것들을 3 가지를 소개 한다.

3.1 MISRA C

자동차 산업 신뢰형 협회(Motor Industry Software Reliability Association)에서 발표한 C 프로그래밍언어에 대한 Coding 가이드 라인으로써 ISO C 로 프로그래밍된 시스템의 Safety, Reliability 확보를 위한 Coding Guideline 이다. 특히 Safety 에 민감한 우주항공, 철도, 국방 등 다양한 분야에서 소프트웨어의 결함을 줄이고자 사용되고 있다. MISRA C 는 무조건 지켜야 하는 것(Mandatory)과, 지켜지길 요구 받는 것(Required)과 지켜지면 좋은(Advisory) Guideline 들로 이루어져 있다. 최근 버전은 2013 년 3 월에 나온 MISRA C 2012 이며 총 159 개의(Mandatory : 10 개, Required 110 개, Advisory 39 개) Guideline 을 제공한다.[5]

3.2 Common Weakness Enumeration(CWE)

CWE 는 미국 국토안보부와 National Institute of Standards and Technology(NIST) 산하 비영리 기관인 MITRE 가 유지 관리하는 전세계 소프트웨어 보안 취약점을 표준화하고 공개한 목록이다. 소프트웨어 보안 및 품질 강화를 위해 개발 시에 이를 참고 할 수 있도록 관련 정보를 제공한다.[6]

3.3 ISO/IEC TS 17961

C 프로그래밍 언어의 Secure Coding 을 위한 Rule 들과 코드 예제들을 제공한다. Rule 들이 적용되는 메카니즘이나 적용이 강제되는 Coding Style 들은 별도로 명시되지 않으며 각 Rule 마다 'noncompliant', 'compliant' 코드를 제공하여 Rule 을 설명한다. 총 46 개의 Rule 들이 제공된다. [7]

3.4 SEI COMMUNITY EMERGENCY RESPONSE TEAM(CERT)

CWE, ISO/IEC TS 17961 등을 포함하는 Safety, Security 에 대한 Coding Guideline 이다. CERT Secure Coding wiki[8] 에서 커뮤니티 기반 개발 프로세스로 개발되었고 해당 커뮤니티에 WG(Working Group)14 표준 위원회 멤버들이 전문가로 초대되어 컨트리뷰션 및 에디터 권한을 가지게 되었고 커뮤니티 멤버들이 자유롭게 Coding Rule 에 대해 코멘트를 하고 있으며 이를 반영해 자주 업데이트 된다. 지키지 않으면 Defect 이 발생하는 'Rule' Guideline 개수는 98 개이며 지키지 않았을 때 Defect 은 없으나 지키면 Security 향상을 위해 좋은 'Recommendation' Guideline 개수는 192 개로 구성되어 있다.

4. MISRA C 와 다른 Coding Guideline 비교연구들

본 세션은 앞에서 설명된 MISRA C 와 다른 Coding Guideline (들)을 비교 분석한 2 가지 연구 결과를 소개한다.

Coding Standard	C Standard	Security Standard	Safety Standard	International Standardization
CWE	None/all	Yes	No	No
CERT C99	C99	Yes	No	No
CERT C11	C11	Yes	Yes	No
ISO/IEC TS 17961	C11	Yes	No	Yes
MISRA C2	C89	No	Yes	No
MISRA C3	C99	No	Yes	No

Figure 3. MISRA C, Coding Guideline 들 비교결과[9]

Figure 3 는 MISRA C 와 다른 Coding Guideline 들 간의 비교 결과를 보여준다. MISRA C 의 경우 Security 확보 측면에서는 Security Standard 를 지원하지 않는 등 다른 Coding Guideline 들에 비해 부족한 점이 있음을 해당 연구 결과를 통해 알 수 있다.

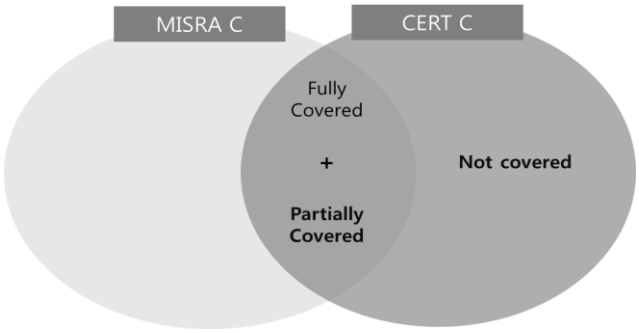


Figure 4. MISRA C, CERT C Guideline 비교결과[10]

Figure 4 는 MISRA C 와 Coding Guideline 들 중 하나인 CERT C 를 집중적으로 비교 분석한 연구 결과 이다. CERT C Guideline 들 중 MISRA C Guideline 으로 Cover 가 가능한지(=대체가능한지) 여부를 확인하여 Cover 가 안 되는 부분(Not Covered), Cover 가 부분적으로 되는 부분(Partially Covered), 완전히 Cover 되는 부분(Fully Covered) 로 나누었다. Guideline 들 간의 매핑에 대한 자세한 설명은 없으나 Cover 가 안되 는 부분(Not Covered), 부분적으로 Cover 되는 부분(Partially Covered) 등 MISRA C 가 CERT C 를 완벽히 Cover 하지 못함을 해당 연구 결과를 통해 알 수 있다.

5. Security 향상을 위한 Coding Guideline 개선 방법

앞 세션에서 소개한 연구들을 통해 현재 Automotive 업계에서 사용중인 MISRA C Coding Guideline 이 Security 확보 측면에서 취약할 수도 있음을 알 수 있다. 본 세션에서는 별도의 기준, 정책을 세워 실제로 MISRA C 가 Security 확보에 취약한지를 CERT C Coding Guideline 과의 비교 분석을 통해 확인하고 이를 바탕으로 Automotive 소프트웨어 Security 향상을 위한 Coding Guideline 개선 방향을 제시 하고자 한다. 여러 Coding Guideline 들 중 CERT C 를 비교 대상으로 선정한 이유는 CWE, ISO/IEC TS 17961 를 아우르며 보다 명확한 설명 및 예제, Coding Style 등을 제공하기 때문이다.

MISRA C 와 CERT C 의 비교는 아래 Figure 5 에 기술된 것 것과 같은 순서로 이루어진다.

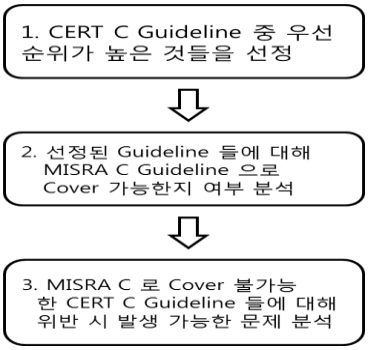


Figure 5. MISRA C, CERT C 비교 분석 과정

다음 세션부터는 각 단계에 대한 자세한 내용을 기술한다.

5.1 CERT C Guideline 선정

CERT C Guideline 은 지키지 않으면 Defect 이 발생하는 'Rule' Guideline 과 지키지 않았을 때 Defect 이 발생할 가능성은 없으나 지키면 Security 향상을 위해 좋은 'Recommendation' Guideline 으로 나뉜다. 이중 상대적으로 중요한 'Rule' Guideline(98 개) 을 우선 비교대상으로 삼았고 이 'Rule' Guideline 들 중

에서도 문제 발생 가능성과 발생시의 심각성이 크고 개선 시 비용이 낮은, 다시 말해 Priority Level 이 가장 높은 Level 1 에 속하는 것들을 선정하였다.

CERT C 에서는 Guideline 미 준수로 인한 문제 발생시의 심각성 정도를 나타내는 Sevirity(1~3) 와 Guideline 미 준수로 인한 문제 발생 가능성 정도를 나타내는 Likelihood(1~3), 그리고 Rule 위반검출 및 수정용이성을 나타내는 Remediation Cost(1~3) 를 기반으로 각 Guideline 의 Priority 값이 결정된다. 그 공식은 다음과 같다.

- Priority (1~27) = Severity(1~3) x Likelihood(1~3) x Remediation Cost(1~3)

Figure 6 은 위 공식으로 구해진 Priority 값을 기반으로 Guideline 들을 3 개의 Priority Level 로 분류한 것이다. 각 Level 별로 존재하는 Guideline 들의 개수는 Level 1 이 17 개, Level 2 는 32 개, Level 3 는 49 개 이다.

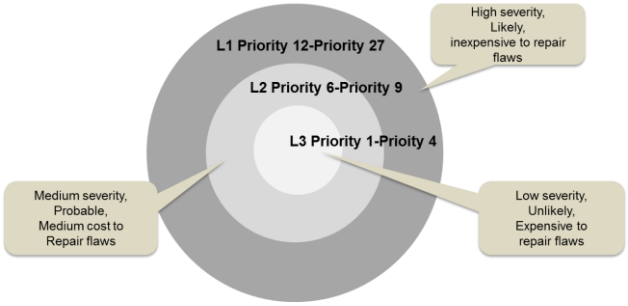


Figure 6. CERT C Guideline Priority Level[9]

Priority Level 1 에 속하는 CERT C Guideline 들 17 개와 Priority 를 결정하는 속성 값들은 다음 Figure 7 과 같다.

Rules	Severity	Likelihood	Remediation Cost	Priority	Level
EXP33-C	High	Probable	Medium	P12	L1
EXP34-C	High	Likely	Medium	P18	L1
ARR38-C	High	Likely	Medium	P18	L1
STR31-C	High	Likely	Medium	P18	L1
STR32-C	High	Probable	Medium	P12	L1
STR38-C	High	Likely	Low	P27	L1
MEM30-C	High	Likely	Medium	P18	L1
MEM34-C	High	Likely	Medium	P18	L1
FIO30-C	High	Likely	Medium	P18	L1
FIO34-C	High	Probable	Medium	P12	L1
FIO37-C	High	Probable	Medium	P12	L1
ENV32-C	Medium	Likely	Medium	P12	L1
ENV33-C	High	Probable	Medium	P12	L1
SIG30-C	High	Likely	Medium	P18	L1
ERR33-C	High	Likely	Medium	P18	L1
MSC32-C	Medium	Likely	Low	P18	L1
MSC33-C	High	Likely	Low	P27	L1

Figure 7. CERT C Rule 선정 결과

5.2 MISRA C 와 선정된 CERT C 비교 분석

선정된 CERT C Guideline 들에 대한 설명과 해당 Guideline 에 대한 MISRA C Guideline 의 Cover 여부(동일한 효과를 가지는 Rule 의 존재 할 경우 'Covered', 존재하지 않으면 'Not Covered')및 설명을 표(Table 1)로 기술 하였다. 둘간의 비교는 Guideline 내에 명시된 Keyword 비교 및 MISRA C Guideline 에 대한 line by line 분석을 통해 이루어졌다.

Table 1. CERT C Guideline 의 MISRA C Cover 여부 확인 결과

Guideline No.	Guideline Description	MISRA C 의 Cover 여부 MISRA C Guideline
EXP33-C	Do not read uninitialized memory	Covered
	초기화 안된 지역 변수(automatic variables)를 읽을 경우 문제 발생을 대비함.	* Rule 9.1 (Mandatory): The value of an object with automatic storage duration shall not be read before it has been set
EXP34-C	Do not dereference null pointers	Covered
	NULL 포인터를 참조 할 경우의 문제 가능성. 특히 동적 할당 실패 후 NULL 을 반환 했을 때 이를 그대로 사용하여 문제 발생하는 경우를 대비함.	* Dir 4.1 (Required): Run-time failures shall be minimized - Pointer dereferencing. * Rule 11.9 (Required): The macro NULL shall be the only permitted from of integer null pointer constant.
ARR38-C	Guarantee that library functions do not form invalid pointers	Not covered
	라이브러리 함수(memset 같은..)가 invalid 주소에 접근할 가능성. 배열 메모리의 주소와 그 사이즈를 인자로 받는 함수에 사이즈를 잘못(더 크게) 주었을 경우 문제 발생하는 것을 대비함.	No Guideline
STR31-C	Guarantee that storage for strings has sufficient space for character data and the null terminator	Not covered
	문자열 복사 등을 수행할 때 문자열+NULL 문자 의 크기를 고려하지 못해 문제 발생 하는 것 대비함.	No Guideline

STR32-C	Do not pass a non-null-terminated character sequence to a library function that expects a string	Not covered
	NULL 로 끝나지 않은 문자열을 라이브러리 함수의 인자로 사용하여 문제 발생하는 것을 대비함.	No Guideline
STR38-C	Do not confuse narrow and wide character strings and functions	Not covered
	Wide character 문자열을 별도의 명시 없이(문자열 앞에 L 붙이지 않고) 일반 문자열 조작 함수의 인자로 넘길 때 문제 발생하는 것을 대비함.	No Guideline
MEM30-C	Do not access freed memory	Covered
	이미 free() 로 해제된 포인터(dangling pointer)에 접근하여 이를 사용할 경우 발생하는 문제를 대비함.	* Rule 18.6 (required): The address of an object with automatic storage duration shall not be copied to another objet that persist after the first object has ceased to exist * Rule 21.3 (Required): The memory allocation and deallocation functions of <stdlib.h> shall not be used
MEM34-C	Only free memory allocated dynamically	Covered
	동적으로 할당되지 않은 메모리가 free() 로 해제할 때의 문제 발생하는 경우를 대비함.	* Rule 21.3 (Required): The memory allocation and deallocation functions of <stdlib.h> shall not be used.
FIO30-C	Exclude user input from format strings	Covered
	User input 에 따라 동적으로 할당할 메모리의 크기가 바뀔 경우의 문제 발생을 대비함.	* Rule 21.3 (Required): The memory allocation and deallocation functions of <stdlib.h> shall not be used
FIO34-C	Distinguish between characters read from a file and EOF or WEOF	Covered
	파일입력을 통해 들어온 문자열과 EOF,WEOF 를 구분하지 못할 경우의 문제 발생을 대비함.	* Rule 21.6: The Standard Library input/output functions shall not be used
FIO37-C	Do not assume that fgets() or fgetws() returns a non empty string when successful	Covered
	fgets, fgetws 등의 함수가 호출 성공 시 항상 비어있지 않은 문자열을 반환할 것이라고 가정하여 발생하는 문제를 대비함.	* Rule 21.6: The Standard Library input/output functions shall not be used
ENV32-C	All exit handlers must return normally	Covered
	프로그램 종료 시 exit 류의 함수를 사용해 호출되도록 등록된 함수들이 정상적으로 return 하지 않아 발생하는 문제를 대비함.	* Rule 21.8 The library functions abort, exit, getenv and system of <stdlib.h> shall not be used
ENV33-C	Do not call system()	Covered
	system () 함수를 사용할 경우 발생 가능한 다양한 security 문제를 대비함.	* Rule 21.8 The library functions abort, exit, getenv and system of <stdlib.h> shall not be used
SIG30-C	Call only asynchronous-safe functions within signal handlers	Covered
	Signal handler 내에서 asynchronous safe 하지 않은 함수를 호출하여 발생하는 문제를 대비함.	* Rule 21.5 The standard header file <signal.h> shall not be used
ERR33-C	Detect and handle standard library errors	Covered
	모든 표준 라이브러리에 함수들 호출 시 리턴 값들에 대해 error 체크를 수행하고 이를 핸들링 해주지 않아 발생하는 문제를 대비함.	* Dir 4.7 (Required): If a function returns error information, then that error information shall be tested
MSC32-C	Properly seed pseudorandom number generators	Not covered
	random() 함수 사용하여 Random 수 생성시 srandom() 을 호출 하지 않아 발생하는 문제를 대비함.	No Guideline
MSC33-C	Do not pass invalid data to asctime() function	Covered / * Rule 21.10(Required) : The Standard Library time and data functions shall not be used
	Asctime 함수로 invalid data 를 전달하여 문제 발생.	

5.3 위반 시 취약점 분석

17 개의 CERT C Guideline 들 중 5 개가 MISRA C 에 의해 'Not Covered' 되는 것을 확인하였다. 해당 CERT C Guideline 에 대해 각각 위반시의 발생 가능한 Risk 를 Table 2 와 같이 분석하였다.

Table 2. CERT C Guideline 위반 시 Risk 분석 결과

Guideline No.	Guideline	Risk 분석 결과
ARR38-C	Guarantee that library functions do not form invalid pointers	라이브러리 함수가 invalid 주소에 접근할 가능성. 메모리의 주소와 그 크기를 인자로 필요로 하는 함수에 크기를 잘못 주었을 경우 문제 발생 가능. **→ Risk : Buffer Overflow 를 이용한 공격 가능**
STR31-C	Guarantee that storage for strings has sufficient space for character data and the null terminator	문자열 복사 등을 수행 할 때 문자열 크기를 잘못 고려하였을 때 문제 발생 가능. **→ Risk : Buffer Overflow 를 이용한 공격 가능**
STR32-C	Do not pass a non-null-terminated character sequence to a library function that expects a string	NULL 로 끝나지 않은 문자열을 라이브러리 함수인자로 사용하여 문제 발생 가능. **→ Risk : Buffer Overflow 를 이용한 공격 가능**
STR38-C	Do not confuse narrow and wide character strings and functions	Wide character 문자열을 별도의 명시 없이(문자열 앞에 L 붙이지 않고) 일반 문자열 조작 함수의 인자로 넘길 때 문제 발생가능. **→ Risk : Buffer Overflow 를 이용한 공격가능**
MSC32-C	Properly seed pseudorandom number generators	매번 같은 수의 random 수가 발생 가능. **→ Risk : 암호화 관련해 해당 함수가 사용될 가능성이 있고 매번 같은 수가 발생함으로써 공격자에게 '예측성' 을 제공함**

위 Guideline 들 중 STR32-C 의 Guideline 에 대한 Noncompliant(Table 3) Code 코드를 보면 String 복사 함수 내에서 Array 의 크기와 NULL('/0') 문자에 대한 고려를 잘 하지 못해 Line 9 에서 결국 'Off-By-One' Buffer Overflow[11] 가 발생함을 알 수 있다.

Table 3. Noncompliant Code Example of STR31-C[9]

Noncompliant Code Example

1: #include <stddef.h>
2:
3: void copy(size_t n, char src[n], char dest[n]) {
4: size_t i;
5:
6: for (i = 0; src[i] && (i < n); ++i) {
7: dest[i] = src[i];
8: }
9: dest[i] = '\0';
10: }

Table 4 는 이 문제를 회피한 Compliant Code 를 보여준다. Line 6 에서 배열의 크기와 NULL 문자를 고려한 프로그래밍을 하여 문제발생을 회피하였다. 간단해 보이지만 개발자가 실수하여 빠뜨리기 쉽고 세심하게 챙기지 않으면 문제가 발생할 가능성이 크다.

Table 4. Compliant Code Example of STR31-C[9]

Compliant Code Example
1: #include <stddef.h> 2: 3: void copy(size_t n, char src[n], char dest[n]) { 4: size_t i; 5: **6: for (i = 0; src[i] && (i < n - 1); ++i) {** 7: dest[i] = src[i]; 8: } 9: dest[i] = '\0'; 10:}

이와 같이 STR32-C 를 포함한 문자 Manipulation 및 Array 관련 Guideline 들(ARR38-C, STR31-C, STR32-C, STR38-C) 에 대해서 MISRA C 가 Cover 하지 못하는 것으로 확인되었으며 이는 공격자가 Arbitrary Code 를 실행하여 관리자 권한을 취득하거나 Buffer Overflow 를 일으켜 서론에 소개된 Heart Bleed 이슈와 같이 시스템에 심각한 Security 문제를 일으킬 수 있는 가능성이 있다. 또한 Random 수 발생 함수와 관련된 MSC32-C 의 경우도 Automotive 시스템에서의 그 Likelihood 가 커지고 있고 Severity 가 크므로 무시 할 수 없는 Guideline 이다. 그러므로 여기서 확인된 5 가지 CERT C Guideline 들에 대해서는 Automotive 시스템 개발 시 필수적으로 고려가 필요하며 이 Guideline 들 적용을 통해 심각한 Security 문제 발생 가능성을 줄 일 수 있다.

5.4 개선 방향

기존에 사용하던 Guideline 을 당장 한번에 바꿀 수는 없으므로 점진적인 개선을 수행한다. 아래 Figure 8 와 같이 일단 CERT C Guideline 에서 Priority Level 이 높은 것부터 MISRA C 와 비교 분석하여 개선하도록 하며 추후에 다른 Guideline 들과도 비교 분석을 통해 개선하도록 한다.

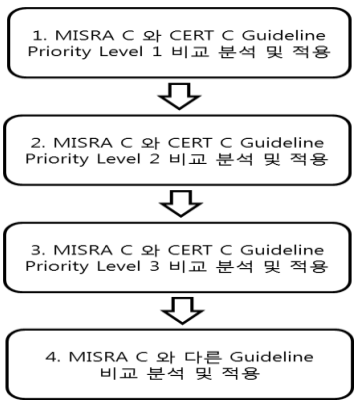


Figure 8. Guideline 개선 방향

6. 결론

CERT C Coding Guideline 과 현재 Automotive 업계에서 사용중인 MISRA C Coding Guideline 과의 비교 분석을 수행하였다. 이를 통해 MISRA C Guideline 만으로는 Security 확보에 취약함을 확인하였고 이에 대한 개선 방향을 제시하였다.

Automotive 업계에서 MISRA C Coding Guideline 과 Security 확보와의 연관성 및 Security 의 중요성에 대한 인식이 변화되어야 하며 개선이 반드시 필요함을 알 수 있다.

향후에는 CERT C Coding Guideline 에서 Priority Level 2, 3 로 분류되는 것들도 추가적인 비교 분석을 통해 MISCRA C 에 의해 'Not covered' 되는 Guideline 들에 대한 미 준수로 인한 취약점을 확인하고 해당 Guideline 들에 대한 적용도 고려하여야 겠다. 또한 이런 Coding Guideline 에 집중된 접근뿐만 아니라 Automotive Security 확보를 위한 전반적인 개발 프로세스 개선에 대한 연구도 필요 하다.

참고문헌

[1] Ghafoor, Imran, et al. "Analysis of OpenSSL Heartbleed Vulnerability for Embedded Systems."

[2] Allied Market Research. "Connected Car Market - Opportunities and Forecasts, 2013–2020"

[3] Checkoway, Stephen, et al. "Comprehensive Experimental Analyses of Automotive Attack Surfaces." USENIX Security Symposium. 2011.

[4] Greenberg, Andy. "Hackers remotely kill a jeep on the highway—with me in it." Wired, 21July (2015).

[5] Misra C+. "Guidelines for the Use of the C Language in Critical Systems." ISBN 978-1-906400-11-8 (PDF), March 2013.

[6] Christey, S. M., and R. P. Glenn. "Common weakness enumeration." (2013).

[7] ISO/IEC TS 17961:2013 Information technology -- Programming languages, their environments and system software interfaces -- C secure coding rules

[8] SEI CERT Coding Standards https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards

[9] Seacord, Robert C. The CERT C Coding Standard: 98 Rules for Developing Safe, Reliable, and Secure Systems. Pearson Education, 2014.

[10] Secure Coding and MISRA C/C++ in ECU Development - Jone Holle, Priyamvadha Vembar

[11] Haugh, Eric, and Matt Bishop. "Testing C Programs for Buffer Overflow Vulnerabilities." NDSS. 2003.

소프트웨어 품질관리를 위한 이종 코드 변환 프레임워크 연구

손현승*, 김영철*

* 홍익대학교 소프트웨어공학연구실
세종특별자치시 조치원읍 세종로 2639
{son, bob}@selab.hongik.ac.kr

요약: MDA(Model Driven Architecture)는 플랫폼 독립적인 모델을 종속적인 모델로 변환해 코드를 생성하는 순공학 기법이고 ADM(Architecture Driven Modernization)은 코드로부터 모델로 만드는 역공학 기법이다. MDA 와 ADM 의 병합은 이종 플랫폼에서의 모델과 코드 동기화 가능해 소프트웨어 변경에 의한 모델과 코드 불일치 문제를 해결할 수 있다. 그러나 현재 MDA 와 ADM 은 호환되지 않아 모델변환에 많은 규칙과 절차가 요구된다. 본 논문에서는 이종 플랫폼에서 양방향으로 모델과 코드를 변환 가능한 이종 코드 변환 프레임워크를 제안한다. 제안한 프레임워크는 양방향 변형이 가능하도록 MDA 와 ADM 병합한 개발 프로세스, 모델 변환 언어, 엔진으로 구성되어 있다. 이 프레임워크는 모델에서 코드생성까지 서로 다른 언어와 방법으로 적용된 기존 모델변환의 개선으로 한 번의 모델 변환으로 모델에서 코드, 코드에서 모델로 변환 가능하다. 또한 이종 모델과 코드를 양방향으로 변환이 가능하다.

핵심어: MDA(Model Driven Architecture), 메타모델, ADM(Architecture Driven Modernization), 모델변환, 역공학

1. 서 론

소프트웨어의 중요성이 높아지면서, 소프트웨어는 광범위하게 사용되었고 크기가 켜졌고 복잡해졌다. 이러한 상항에서 소프트웨어의 개발비 절감과 신뢰성 확보 등의 품질관리가 중요한 현안으로 대두되고 있다.

고품질의 소프트웨어를 만들기 위해서는 SP(Software Process) [1], CMMi(Capability Maturity Model Integration) [2], TMMi(Test Maturity Model integration) [3] 등과 같은 인증 프로세스를 도입하여 개발과정의 성숙도를 높이거나 테스팅이나 GS(Good Software) [4] 같은 제품의 품질을 높이는 방법이 있다. 그러나 인증 프로세스는 평가 기준, 절차, 방법, 도구 활용, 문서 등과 같은 다양한 것들이 필요하여 도입 비용이 높다 [5]. 또한 테스팅은 전문 인력과 기술이 추가적으로 필요하다.

이러한 소프트웨어 개발 실정을 볼 때, 고품질 소프트웨어를 위한 다른 방안으로 기존 소프트웨어 자산 축적과 재사용 활용한 기술도 있다 [6]. 소프트웨어는 본질적인 면에서 축적된 지식과 경험을 반영한 상품으로 특이한 소유, 거래의 개념을 갖고 이를 재사용과 공유로 사회적 가치 증대가 가능하다.

이러한 소프트웨어의 특성 때문에 많은 회사들은 기존 소프트웨어로부터 자산을 구축하거나 재사용할 수 있는 플랫폼을 선호한다. 이렇다 보니 소프트웨어 기술이 발전할 수록 많은 수의 플랫폼이 생겨나고 있다. 이런 다양한 소프트웨어 플랫폼 등장으로 다중 플랫폼에서의 소프트웨어의 품질을 높일 수 있는 방법도 필요해졌다.

MDA(Model Driven Architecture) [7-8]는 닷넷(.Net), 코바(CORBA), 자바(Java)와 같은 플랫폼 기반 소프트웨어의 플랫폼간 종속 문제를 모델을 활용해 없애고 모델을 코드로 생성해 이종 플랫폼의 코드까지 생성할 수 있다. MDA 는 플랫폼의 독립과 종속 모델로 분리하고 두 모델간의 차이를 모델 변환 언어와 엔진을 사용해 자동화 한다. 두 모델간의 추상화 수준 차이로 생기는 정보는 프로파일이나 모델 변환 규칙을 통해서 극복한다.

ADM(Architecture Driven Modernization) [9]은 역공학 기법으로 코드로부터 모델을 모델변환으로 만드는 자동화 기법이다. 코드를 모델로 만들기 위해서 텍스트로 된 코드를 파싱한 ASTM(Abstract Syntax Tree Metamodel) [10]와 코드와 모델의 정보를 모두 저장하는 KDM (Knowledge Discovery Metamodel) [11]을 사용한다.

* 본 논문은 “손현승, MDA/ADM 통합 모델 기반의 이종 코드 변환 프레임워크, 홍익대학교 대학원, 2015”박사학위 논문을 재편집하였음.

† 이 논문은 2015 년 교육부와 한국연구재단의 지역혁신창의인력양성사업(NRF-2015H1C1A1035548)과 2015 년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2013R1A1A2011601).

MDA 와 ADM 의 병합은 모델과 코드의 동기화가 가능하여 소프트웨어 변경에 따른 모델과 코드의 불일치 문제를 해결할 수 있다. 그러나 현재 MDA 와 ADM 기법의 수행은 두 기법 사이에 호환되지 않아 모델 변환시 많은 규칙과 절차가 요구된다.

본 논문에서는 이종 플랫폼 상에서 양방향으로 모델과 코드 변환을 위한 이종 코드 변환 프레임워크를 제안한다. 제안한 프레임워크는 양방향으로 모델과 코드를 변환하기 위해서 개발 프로세스, 모델 변환 언어, 엔진으로 구성되어 있다. 개발 프로세스에서는 MDA 와 ADM 를 병합하기 위해서 두 프로세스는 같은 모델을 사용하도록 하고 양방향 모델 변환 언어를 통해서 양방향 변형을 가능하게 한다. 또한 코드와 모델은 ASTM 을 공유하여 별도의 언어와 도구 없이 양방향으로 모델과 코드를 변환 할 수 있도록 하였다. 이를 위해 양방향 모델 변환 언어와 엔진은 여러 종류의 모델 변환 방법(Model to Model, Model to Text, Text to Model)을 하나의 언어로 처리할 수 있다.

제안한 프레임워크는 기존 소프트웨어 코드를 모델을 경유하여 이종 코드로 변환하거나, 코드를 모델로, 모델을 코드로 생성 가능하다. 특히, MDA 와 ADM 의 접목을 통해서 순공학 뿐만 아니라 역공학 기법 포함으로 코드를 이용한 개발에서 그래프를 통한 시각화나 문서화가 용이하여 소프트웨어 품질관리에 도움을 준다.

2. 이종 코드 변환 프레임워크

이종 코드 변환 프레임워크는 이종 플랫폼 개발을 위해 필요한 모델과 코드를 양방향으로 모델 변환할 수 있는 프로세스, 방법, 도구를 포함하는 프레임워크이다. 이 코드 변환 프로세스는 그림 1 과 같이 MDA 와 ADM 을 병합한 프로세스로 플랫폼 독립 모델로부터 양방향 모델 변환 언어로 종속 모델로 만들고 이를 ASTM(Abstract Syntax Tree Metamodel)으로 변환하여 코드를 생성하는 순방향 프로세스와 생성된 코드를 역으로 ASTM 으로 만들고 ASTM 을 종속 모델로, 종속 모델을 독립 모델로 변환하는 역방향 프로세스로 구성되어 있다. 그러므로 이 프로세스는 순방향과 역방향에 대한 변환을 모두 수행할 수 있는 순환구조이다.

MDA 과 ADM 을 병합만 한 경우 MDA 와 ADM 의 모든 과정을 수행해야 하므로 변환 절차가 복잡해지고 비효율적인 문제가 있다. 이러한 문제를 해결하기 위해서 제안한 이종 코드 변환 프로세스는 기존의 MDA 와 ADM 에서 꼭 필요한 단계만 수행하도록 최적화한다. 모델에서 모델의 변환은 양방향의 모델 변환 언어를 개발하여 변환의 수행을 간단하게 하였고 코드생성과 파싱은 ASTM 을 사용하여 표준화된 하나의 AST(Abstract Syntax Tree)에서 처리 할 수 있도록 하였다.

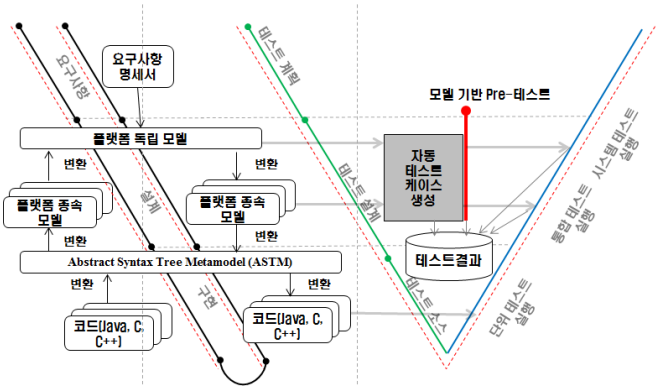


그림 1 이종 플랫폼을 위한 코드 변환 프로세스

제안한 프로세스의 실현을 위한 이종 코드 변환 구조는 그림 2 와 같다. 이 코드 변환 구조는 모델에서 모델로 변환하기 위한 변환 언어와 엔진과 ASTM 을 기반으로 코드 생성 및 파싱할 수 있는 파서로 구성한다. 모델 변환 언어는 모델에서 모델로 변환을 수행하기 위해서 변환되는 규칙을 메타모델을 사용하여 작성하고 양방향 모델 변환을 보다 효과적으로 수행하기 위해서 제안한 모델 변환 언어는 트리를 기반으로 한다. 파서는 ASTM 을 코드로 코드를 ASTM 으로 변환해 주는 역할을 수행한다. 이러한 두 가지 부분이 병합되어 모델에서 코드, 코드에서 모델로 자유롭게 자동변환 할 수 있다.

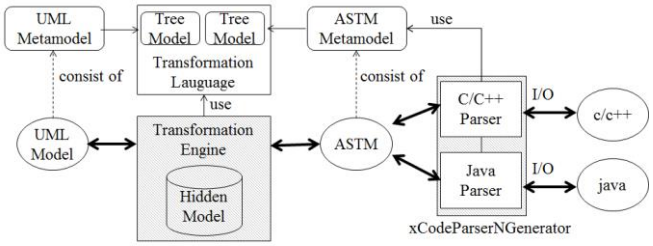


그림 2 이종 코드 변환의 구조

양방향 모델 변환에서는 모델간의 추상화 수준의 차이가 있거나 모델과 모델이 1:1 로 완전 매치되지 않는 경우 정보가 손실될 수 있다. 이 문제를 해결하기 위해서는 모델 변환 수행시 모델 변환 규칙이 1:1 로 대응되지 않을 때, 두 모델 사이의 관계를 1:1 대응 관계가 되도록 해줌으로 정보 손실의 문제점을 해결 할 수 있다.

3. 결 론

단일 플랫폼 기반 개발은 소프트웨어의 재사용으로 빠르게 개발이 가능하지만 이종 시스템 개발에

적합하지 않다. 또한 고품질 소프트웨어를 위해서는 코드의 분석을 통해 모델로 변환하거나 결함을 찾을 수 있는 방법이 필요한데, 기존 방법은 한 번에 수행할 수 없었다. 본 논문은 이러한 이종 시스템의 개발 환경과 소프트웨어 품질의 문제를 해결하고자 이종 코드 변환 프레임워크를 제안하였다.

제안한 이종 코드 변환 프레임워크는 양방향 모델과 코드를 변환 가능하여 모델에서 코드, 코드에서 모델의 과정을 일괄적으로 통합한 자동화 방법이다. 이를 위해서 MDA 와 ADM 을 병합한 프로세스를 제안하고, 또한 모델에서 모델 변환을 위한 양방향 모델과 코드 변환 언어와 엔진을 개발했다.

이종 코드 변환 프레임워크는 이종의 플랫폼을 사용하는 모바일, 스마트폰 환경, 임베디드 소프트웨어 등에 활용과 코드 가시화가 가능하여 소프트웨어 품질관리에 도움이 될 것으로 기대한다.

향후 연구로 양방향 모델 변환의 효율성을 높이기 위한 알고리즘과 언어 체계 개선 연구이다. 또한 제안한 프레임워크를 테스트 프로세스를 접목한다면 테스트 프로세스에 설계 모델을 그대로 사용하면 자동 테스트에 적용하여 더 높은 품질향상이 이루어질 수 있다.

참고문헌

[1] 임수빈, 이승주, 남성욱, 소프트웨어프로세스 품질인증 적용 가이드북 2 등급, 긍정 미디어, 2011.

[2] M.B. Chrissis, M. Konrad, S. Shrum, CMMI Guidlines for Process Integration and Product Improvement, Addison-Wesley, 2003.

[3] E. Van Veenendaal, J.J. Cannegieter, Test Maturity Model integration (TMMi), TMMi Foundation, 2010.

[4] 김현정, 장형진, 김장경, 신석규, “TTA 소프트웨어 시험 · 인증 서비스 소개”, 한국통신학회지(정보와 통신), Vol. 31, No. 7, pp. 24-31, 2014.

[5] 양해술, 배두환, “소프트웨어 품질표준화와 시험 · 인증기술의 동향”, 정보과학회지, Vol. 23, No. 3, pp. 45-55, 2005.

[6] I. Jacobson, M. Griss, P. Jonsson, Software reuse: architecture, process and organization for business success, ACM Press/Addison-Wesley, 1997.

[7] OMG, “Technical Guide to Model Driven Architecture: The MDA Guide v1.0.1”, Document Number: omg/2003-06-01

[8] S.J. Mellor, A.N. Clark, T. Futagami, “Model-Driven Development”, IEEE Software, no. 5, pp. 14-18, 2003.

[9] V. Khusidman, “Architecture-Driven Modernization and Transformation”, ADM Transformation White Paper, pp. 1-5, 2008.

[10] OMG, “Architecture-driven Modernization: Abstract Syntax Tree Metamodel (ASTM) Version 1.0”, Document Number: formal/2011-01-05.

[11] OMG, “Architecture-Driven Modernization: Knowledge Discovery Meta-Model (KDM) Version 1.3”, Document Number: formal/2011-08-04.

역공학을 이용한 오픈소스의 소스코드 요구사항 도출 및 Best Practices 와의 비교 분석을 통한 소스코드의 품질평가

위대한

아주대학교 소프트웨어특성화학과
경기도 수원시 영통구 원천동 산 5 번지
wdhwdh1@ajou.ac.kr

이석원

아주대학교 소프트웨어융합학과
경기도 수원시 영통구 원천동 산 5 번지
leesw@ajou.ac.kr

요약: 오픈소스 프로젝트는 많은 사람들이 참여하는 프로젝트의 한 형태이다. 하지만 이러한 오픈소스 프로젝트는 전 세계의 다양한 사람들이 개발에 참여하게 되면서 시간적, 공간적 제약으로 소프트웨어 공학 프로세스를 따르기 힘든 경우가 발생한다. 이는 소프트웨어의 품질에 영향을 줄 수 있다. 따라서 본 연구에서는 일반적인 소프트웨어 공학의 프로세스를 적용하기 어려운 오픈소스의 특징을 고려하며 품질을 평가하기 위해 필요한 프로젝트의 요구사항이 무엇인지를 도출한다. 요구사항을 도출한 이 후, 해당 요구사항을 구현하기 위한 Best Practices 가 존재하는지 지식베이스를 탐색한다. 해당 요구사항에 대한 Best Practices 가 존재함에도 이에 따라 구현되지 않은 코드가 많은 경우 이는 현재 소스코드의 품질을 보장할 수 없음을 나타낸다. 따라서 본 연구에서는 역공학으로 추출한 코드의 요구사항과 Best Practices 와의 매핑정도를 품질측정의 척도로 이용함으로써 작성된 소스코드의 품질을 얼마나 보장할 수 있는지를 나타내고자 한다. 이러한 과정을 통해서 오픈소스에 작성된 소스코드의 요구사항을 정확하게 파악하고 검증의 관점에서 정확하게 소스코드가 요구사항에 맞도록 작성이 된 것인지 평가하는 프레임워크를 제안한다.

핵심어: 오픈소스, 요구공학, 코드품질, Abstract Syntax Tree, 온톨로지, CheckStyle

1. 연구배경 및 문제정의

오픈소스란 소스코드가 공개되어 모든 사람들이 열람 할 수 있도록 개방된 자료들을 이야기한다. 오픈소스 프로젝트는 다양한 사람들이 개발에 참여할 수 있도록 개방되어 있다는 점 때문에 일반적인 소프트웨어 공학 프로세스를 따르기 어려운 프로젝트 중 하나이다. [1]

이러한 오픈소스의 개방성을 고려하여 프로젝트를 통제하기 위해 오픈소스 프로젝트에 참여하는 사람들의 역할은 크게 4 가지로 구분된다. 이미 만들어져 있는 소스를 그냥 가져다 쓰는 사용자, 사용하면서 발생한 버그나 문제점에 대하여 수정된 소스코드를 업로드 하는 컨트리뷰터, 컨트리뷰터들이 올린 패치파일들을 보고 오픈소스 프로젝트의 목적과 부합되는 소스코드를 커밋할 수 있는 권한을 가진 커미터, 그리고 가장 처음 오픈소스의 프로젝트를 만든 오픈소스 프로젝트 커미티 총 4 가지의 역할이 존재한다.

처음 오픈소스 프로젝트가 만들어질 시점에서는 개발하고자 하는 내용에 대한 구체적이고 일관된 목표가 있다. 오픈소스 프로젝트에서는 앞서 언급한 4 가지 역할을 통해 본래의 목적을 잃지 않고 프로젝트를 진행하고자 한다.

하지만 대중적으로 많이 알려진 오픈소스 프로젝트를 제외하면 많은 오픈소스 프로젝트들은 아직 소프트웨어 공학 프로세스를 따기 힘든 환경에 있다. 이는 오픈소스 프로젝트가 개발과정에서 변화하는 요구사항에 대처하기 어렵도록 하며 소프트웨어를 검증 하기 위해 필요한 요구사항이 부재하는 상황을 초래한다. 오픈소스 프로젝트는 전문지식을 가진 개발자들을 수용하고 빠른 개발이 가능하도록 한다. 하지만 여전히 오픈소스에서는 테스팅이나 문서화 작업과 같은 중요한 프로세스를 간과하는 경우가 존재한다. [2] 새로운 개발자들의 참여가 늘어날수록 본래의 목적을 유지하기는 어려워지고 이는 오픈소스 프로젝트를 통해서 생성된 소스코드의 품질을 낮출 가능성이 있다.

이러한 문제 하에서 존재하는 산출물은 소스코드이다. 따라서, 검증을 위해 소스코드를 통해서 역으로 어떤 요구사항을 만족시키기 위해 만들어진 코드인지, 그리고 어떻게 기능을 구현하기 위해 작성된 코드인지 파악하는 작업이 필요하다. 또한, 단순히 구문적 관점에서가 아닌 의미적 관점에서 코드를 이해하고 파악하는 것이 중요하다.

2. 기본 개념

2.1 추상구문트리 (Abstract Syntax Tree: AST)

AST 는 추상구문트리이다. 기본적으로 소스코드는 AST 를 이용하여 트리모양의 형태로 분해된다. AST 는 소스코드를 읽은 후 이를 문법요소단위로 파싱한다. 그 후 구조체별로 트리를 추상적으로 작성한다. 보통 이는 컴파일러에서 소스코드를 컴퓨터가 이해할 수 있는 언어로 변형하는데 사용되며, 한 언어에서 다른 언어로 변환하는 경우에 주로 사용된다. AST 는 내용을 추상화시키기는 작업이기 때문에 소스코드의 구체적인 내용은 알 수 없다. 하지만 이는 소스코드를 가장 기본적인 단위로 분해한 상태에서 분석을 시작할 수 있도록 하기 때문에 기본단위에서부터 체계적으로 소스코드의 문맥을 파악하는데 도움을 준다.

2.2 온톨로지 지식베이스

온톨로지란 개념화에 대한 정확한 명세를 말한다 [3]. 여기서 개념화라는 것은 실제 세상의 특정 부분을 추상적인

표 1 오프소스의 품질평가를 위한 과정

Phase1	지식베이스 구축	Step1	지식 체계 구축을 위한 자료 수집
		Step2	수집한 Best Practices 들을 AST 수준으로 분해
		Step3	Step2 의 내용을 온톨로지 지식베이스에 저장
		Step4	소스코드의 역공학 과정에서 코드의 이해를 위해 필요한 구성요소로써 코드의 흐름, 변수의 개수, 그 외의 다른 문법요소들을 특정 목적과 매핑
		Step5	Step4 를 통해 나온 내용을 온토롤지 지식베이스에 저장
Phase2	Target Source Code 분석	Step6	분석하려는 소스코드를 AST 로 분해
		Step7	Checkstyle 을 통해 분석 소스코드를 입력
		Step8	분석한 소스코드의 요구사항과 일치하는 요구사항을 지식베이스에서 확인
		Step9	Target Source Code 의 요구사항 도출
		Step10	Target Source Code 의 요구사항에 해당하는 Best Practices 가 존재하는지 확인

모델로 만들어 컴퓨터 상에서 표현하는 것을 말한다. 이 때, 중요한 개념과 개념들 사이에서 맺어진 관계들 사이의 속성 그리고 그 속성의 정의들을 이용하여 모델을 만들고 이용한다. 실제 세상의 특정 부분을 모델로 만듦으로써 그 개념들을 표현하기 때문에 온톨로지를 구축할 때에는 도메인과 범위를 구축 가능한 수준으로 설정한다. 하지만 정해진 범위 안에서 구축된 온톨로지는 해당하는 도메인에 대하여 명확한 지식을 가지고 있어야 하며 일관된 추론을 할 수 있도록 구축되어야 한다.

온톨로지의 가장 큰 특징중의 하나는 온톨로지 지식베이스에 저장되어 있는 정보들의 속성을 통해 겉으로 드러나지 않는 문맥을 도출해 낸다는 점이다. 이를 추론이라 하며 일반적인 데이터 베이스에서 제공하지 못하는 가장 큰 차이점이다.

2.3 Checkstyle

Checkstyle 은 공동으로 작업하는 협업 시 사용되는 코딩 컨벤션을 소스상에서 잡아 주는 툴로써 Eclipse 에서 Plug-in 형식으로 사용할 수 있다. 기본적으로 제공하는 스타일로써는 Google 스타일, Sun 그리고 Sun(Eclipse) 버전의 스타일이 있으며 이는 변경이 불가능하다. 기본적으로 제공되는 스타일 이외에도 스스로 스타일을 추가 할 수 있으며 이를 통해 소스코드의 작성시 따라야 하는 규칙을 정의할 수 있다. 규칙을 위반하는 코드는 소스코드 라인의 가장 왼쪽에 돋보기 모양으로 표시되며, 위반되는 내용이 수정되면 이는 사라진다.

3. 제안방법

본 연구에서는 소스코드를 검증 관점으로 분석하는데 필요한 요구사항을 추론하기 위해 온톨로지를 이용하며 두 가지 지식베이스를 구축한다. 첫 번째 지식베이스는 AST 에서 소스코드를 표현하는 구성요소들이 어떠한 조합으로 되어 있는 경우 어떤 요구사항과 부합하는지를 나타내는 지식베이스다. 두 번째 지식베이스는 어떤 지식과 기준에 근거하여 소소코드의 품질을 평가 할 것인지에 대한 온톨로지이다.

대중적으로 이미 품질이 검증된 소스코드를 앞서 언급한 두 번째 지식베이스로 구축하기 위하여, 특정 요구사항과 그 요구사항을 구현하기에 가장 적합한 방법으로 알려진 Best Practices, 함수 혹은 method 라이브러리, API, 알고리즘 그리고 디자인 패턴 등의 정보는 온톨로지에 포함되어야 한다.

제시하는 접근법에 대한 개괄적인 내용은 표 1 에 나타난다. 총 2 개의 phase 로 나누어 진행되며 이 안에서 다시 여러 개의 Step 으로 나누어서 설명하고자 한다.

Phase1 은 지식베이스를 구축하는 일이다. 여기서 Step1 은 지식 체계를 구축할 소스들을 모으는 작업으로 적정 수준 이상의 품질을 보장하는 소스코드들을 온톨로지의 지식베이스로 표현하기 위한 데이터 전처리 과정이다. Step2 에서는 찾은 Best Practices, 함수 혹은 method 라이브러리, API, 알고리즘 그리고 디자인 패턴들을 AST 를 이용하여 소소코드에서 가장 Low-level 의 문법수준으로 나눈다. 이후 Step3 에서 이를 소스코드에서 표현된 flow 에 따라서 온톨로지에 저장한다. 온톨로지에 저장할 때에는 하나의 클래스 아래에 인스턴스로 저장하되 클래스의 이름은 그 소스코드의 의도를 가장 잘 나타낼 수 있는 이름으로 저장한다. Step4 에서는 분석하려는 소스코드를 입력 값으로 넣었을 때 이에 해당하는 목적을 찾아주기 위한 지식베이스 구축을 위하여 추상구문트리의 요소들의 다양한 결합 방법을 특정한 목적으로 매칭 하는데 이용될 지식을 모은다. Step5 에서는 이전 단계에서 축적된 지식을 온톨로지 지식베이스에 구축하는 부분을 나타낸다.

Phase1 을 수행하는 과정은 상당히 많은 시간을 소요하게 된다. 또한 특정 기능을 구현하기 위한 Best Practices 들을 모두 찾는다는 것은 불가능한 일이다. 따라서 Phase1 은 문제를 해결하기 위한 특정 도메인에 한정되어 진행된다. 또한 확장성이 좋은 온톨로지의 특징을 이용하여 지속적으로 지식베이스를 확장하고 수정하는 것이 가능하다.

Phase2 는 품질을 평가하고자 하는 소스코드에 대해서 분석하는 작업이다. Step6 에서는 분석하려는 소스코드의 흐름을 AST 를 이용해서 가장 Low-level 의 문법수준으로 나타낸다. Step7 는 CheckStyle 을 이용하여 분석하려는 소스코드를 읽어 들이는 역할을 한다. Step8 에서는 읽어 들인 소스코드의 내용들을 이용해 온톨로지의 지식베이스에 일치하는 내용이 있는지를 탐색한다. Step9 에서는 일치하는 내용들을 토대로 분석한 소스코드가 어떠한 기능을 구현하기 위하여

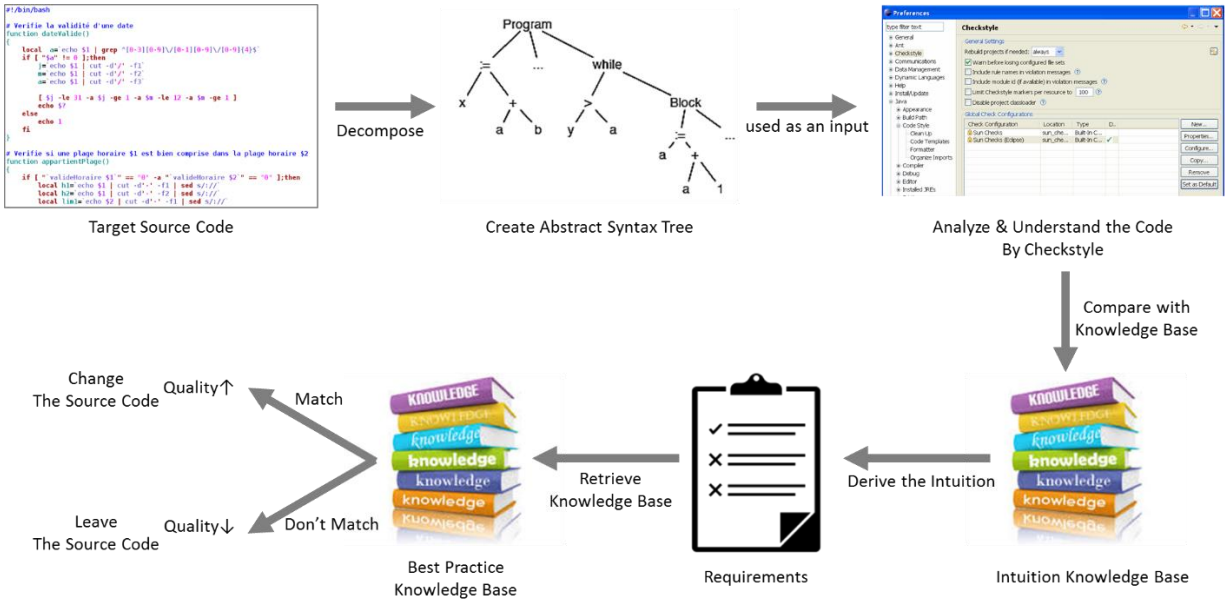


그림 1 소스코드의 품질 평가를 위한 시스템 프레임워크

작성된 코드인지를 파악하여 역으로 요구사항을 도출해낸다. Step10 에서는 도출해낸 요구사항을 다시 입력 값으로 이용하여 이를 구현하기에 가장 알맞은 Best Practices 가 지식베이스에 있는지 검색한다. 만약 매칭되는 Best Practices 가 존재하는 경우 이는 현재 소스코드가 품질을 보장할 수 있는 방향으로 구현되지 않았음을 개발자에게 제시한다. 최종적으로 Best Practices 가 존재함에도 이를 적용하지 않는 코드를 측정하여 현재 소스코드의 품질보증 수준을 제시한다.

제시하는 방법은 분석시점의 소스코드의 요구사항이 무엇인지 역공학적 방법으로 추론을 가능하게 한다. 이 요구사항을 구현하기 위한 Best Practices 가 있는 경우, 이는 검증된 방법으로 같은 요구사항을 구현할 수 있는 방법이 있었음을 사용자에게 알려줌으로써 현재 소스코드의 품질상태가 검증되지 않았음을 나타낸다.

4. 사례연구

본 내용에서는 실제 소스코드를 통해서 제안하는 프레임워크가 어떻게 작동하는지를 나타내고자 한다. 예시로 제시하는 소스코드는 사용자로부터 숫자를 입력 값을 받은 후 그 입력 값에 해당하는 구구단을 나타내는 것이다. 먼저 Best Practices 를 나타내는 지식베이스에는 for 문을 이용하여 작성한 간단한 소스코드를 통해서 사용자가 원하는 구구단을 콘솔 창에 출력할 수 있는 소스코드가 포함되어 있으며 Target 소스는 System.out.println 구문을 9 번 적은 코드로 작성되어 있다.

Target 소스코드가 Best Practices 와 매핑되는 과정은 그림 1 에 자세하게 나타나 있다. 역공학적 방법을 이용해 요구사항을 파악하기 위한 지식들의 내용을 저장하고 있는 온톨로지 지식베이스에서는 System.out.println 소스코드와 이 안에 사용되는 변수가 같지만 출력하는 내용이 약간씩의 차이점이 있다는 것을 AST 를 통해서 파악한다. 이를 통해서 사용자가 의도한 내용은 '같은 내용의 소스코드를 조금 다른 형태로 여러 번 표현하고자 하는 경우' 라는 것을 온톨로지 추론을 통해서 얻어낸다. 즉, High-level 의 사용자 의도를 파악한다. 그 후 다시 이 의도를 매칭되는 Best Practices 를 찾기 위한 입력 값으로 사용한다. 입력되는 요구사항은 '같은 내용의 소스코드를 조금 다른 형태로 여러 번 표현하고자 하는 경우' 이고 이와 매칭되는 소스코드의 Best Practices 를 찾아서 이를 사용자에게 추천하도록 한다. 즉, 본 장의 사례연구에서는 가장 적절한 Best Practices 로 for 문을 이용하여 구구단을 나타내는 소스코드를 찾게 된다. 또한 이와 동시에 이러한 Best Practices 를 따라서 구현하지 않았으므로 현재 소스코드의 품질을 보장할 수 없다는 것을 사용자에게 알려주게 된다. 이러한 프레임워크를 통해 나온 결과를 보고 개발자는 자신이 구현하고자 한 내용이 이미 기존의 다른 사람들에 의해 검증된 방법으로 구현하는 방법이 있음을 알 수 있게 된다. 최종적으로 이를 이용함으로써 구구단 기능의 코드는 품질을 보장하게 된다. 이를 요구사항이 문서가 없는 오픈소스 프로젝트에 이용하면 저장된 소스코드의 품질이 향상되는 결과를 가져올 수 있다.

5. 관련연구

기존의 관련 연구들을 살펴보면, 소프트웨어의 품질을 평가하기 위한 대부분의 연구들이 단순한 Testing 과 관련된 부분이 많다는 것을 알 수 있다. Tianyi Zhang 의 연구 [4] 에서는 코드리뷰를 효과적으로 수행하기 위한 툴로 CRITICS 라는 툴을 제공했다. 이 툴에서는 diff patch 를 기반으로 하여 코드에 변화가 생긴 부분을 탐지한다. 이에 따라 코드의 일관성 유지를 위해 함께 변해야 하는 부분을 탐색해주는 역할을 하여 코드리뷰의 시간을 줄이고 효율을 높였다. 이 연구에서는 비슷한 내용의 소스코드를 가진 부분을 찾아주기 위해 AST 를 이용했지만 단순히 Code Clone 을 탐색하기 위한 방법과 유사하게 진행된다. 이는 코드의 의

미를 파악하지 못하고 단순히 오류가 발생할 수 있는 부분만을 찾아주는데 한정되어 있다.

소스코드를 분석하고 비교하기 위한 방법으로 다른 연구에서도 AST 를 많이 이용하고 있다. Baojiang Cui 의 연구 [5] 에서는 AST 를 이용하여 코드를 비교하는 시스템을 제안하였다. Darryl Owens 의 연구 [6] 에서는 Software Model 품질을 보장하기 위한 프레임워크를 제안하였고 이를 수행하기 위한 도구에 AST 를 이용한 어플리케이션을 추가하였다. 하지만 기존의 연구들은 소스코드의 수준에서 의미를 도출하는 과정이 포함되지 않았다는 점에서 시사할 부분이 있다. 단순한 형식과 구문의 검사는 같은 코드를 적더라도 서로 다른 의미에서 사용될 수도 있다는 점을 배제하고 있다. 소스코드는 단순히 기계어가 아닌 사람에 의해 서비스를 제공하기 위하여 작성된다. 따라서 그 의미를 찾지 못하고 분석하는 것으로는 품질을 평가하기에 한계가 존재한다. 이러한 점에서 본 연구는 소스코드를 모두가 객관적으로 볼 수 있는 Syntactic 한 수준으로 분해하여 Semantic 을 이끌어 낸다는 점에서 연구의 의미가 있다.

소스코드를 Syntactic 한 관점으로 분석하기 보다 Semantic 한 관점에서 분석해보려는 시도는 정보검색 분야에서 Latent Semantic Analysis 라는 방법론이 대표적인 방법으로 거론되고 있다. Jonathan I. Maletic 의 연구 [7] 를 보면 소스코드의 이해를 돕기 위해서 Latent Semantic Analysis 를 소스코드 내부의 주석을 포함하여 자연어로 기술된 부분에 적용하여 의미를 도출하려 하였다. 하지만 이 방법의 가장 큰 단점은 오픈소스의 특징을 고려하였을 때, 오픈소스 프로젝트에는 적용하기가 힘들다는 점이다. 다양한 사람들이 모여 코드가 작성되는 오픈소스 프로젝트의 경우 코드를 작성하는 사람들이 코딩 표준을 따르지 않을 경우, 혹은 그들이 자연어로 작성하는 어투나 단어의 선택도 일관되지 않다는 점을 고려하였을 때 코드의 작성 방법이 모두 다양하기 때문에 제대로 된 의미를 도출해낼 수 없다.

6. 결 론 및 향후 연구

본 논문에서는 다양한 실력과 서로 다른 특징을 가지는 개발자들이 모여서 소프트웨어를 개발하는 환경인 오픈소스라는 특별한 환경에서 만들어지는 소스코드의 품질을 분석하기 위한 방법을 제시한다. 제시하는 방법에서는 이미 만들어진 소프트웨어의 소스코드로부터 그 원래의 목적 및 요구사항을 역공학을 이용하여 추출한다. 그 후 최종적으로는 추출한 요구사항을 바탕으로 그 요구사항을 구현하기에 적절한 Best Practices 가 적용되었는지의 분석을 통해 소스코드의 품질을 검증 관점에서 평가하기 위한 프레임워크를 제시하였다.

특정 요구사항에 구현하기 위한 Best Practices 가 존재함에도 내용이 적용되지 않은 경우, 이는 전체 소프트웨어의 품질에 영향을 줄 수 있다. Best Practices 로 참고 할 수 있는 자료는 많지만 이 모든 자료를 개발자가 숙지한다는 것은 불가능하다. 따라서 이러한 문제점을 해결하고 오픈소스에서의 품질을 평가하는 기준으로 Best Practices 를 이용하기 위한 프레임워크를 제시하는데 본 논문은 집중하였다.

현재 본 논문의 한계점은 구체화된 검증방법과 검증 내용을 뒷받침 하기 위한 결과 분석 내용이 부족하다는 것이다. 또한, Semantic 을 추출하기 위하여 구축된 온톨로지를 설명하기 위한 내용의 보완이 필요하다. 이는 추후 컴파일러 수준인 low-level 에서 소스코드를 파싱하는 방법과 파싱된 소스코드를 어떻게 읽도록 설정할 것인지 결정하고, 그 내용에 따라 파싱되는 소스코드 단위를 이용하여 온톨로지의 구축 예를 사용한 concept 과 property 와 함께 소개함으로써 개선하고자 한다.

또한, 소스코드로부터 요구사항을 추출하는데 이용될 온톨로지를 구축하는데 있어서 발생하는 한계점은 특정한 요구사항을 구현할 때, 사람마다 이를 구현하는 방식이 다르기 때문에 다양한 방식의 소스코드로 구현될 수 있다는 점이다. 따라서 하나의 요구사항에 매핑될 수 있는 그 다양한 코드들을 모두 고려하여 온톨로지를 구축하는 것은 불가능하다. 따라서 이를 극복하기 위해, 확장성이 좋은 온톨로지의 특징을 통해 지속적으로 지식베이스를 확장하여 본 프레임워크의 정확도를 높이고 구현 가능성과 타당 가능성을 제시하고자 한다.

Acknowledgement

이 논문은 2013 년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보・컴퓨팅기술개발사업의 지원을 받아 수행된 연구임 (2013M3C4A7056233)

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW 특성화 대학원 지원 사업의 연구결과로 수행되었음(R0346-15-1017)

참고 문헌

[1] J. Kuriakose and J. Parsons, “How Do Open Source Software (OSS) Developers Practice and Perceive Requirements Engineering? An Empirical Study,” *International Workshop on Empirical Requirements Engineering*, Ottawa, 2015.

[2] I. Stamelos, L. Angelis, A. Oikonomou and G. L. Bleris, “Code quality analysis in open source software development,” *Information systems journal*, 2002.

[3] T. Gruber, “Toward principles for the design of ontologies used for knowledge sharing,” *Padua Workshop on Formal Ontology*, 1993.

[4] T. Zhang, M. Song, J. Pinedo and M. Kim, “Interactive Code Review for Systematic Changes,” *International Conference on Software Engineering*, Firenze, 2015.

[5] B. Cui, J. Li, T. Guo, J. Wang and D. Ma, “CODE COMPARISON SYSTEM BASED ON ABSTRACT SYNTAX TREE,” *International Conference on Broadband Network and Multimedia Technology*, Beijing, 2010.

[6] D. Owens and M. Anderson, “A Generic Framework for Automated Quality Assurance of Software Models - Application of an Abstract Syntax Tree,” *International Conference on Broadband Network and Multimedia Technology*, London, 2013.

[7] J. I. Maletic and A. Marcus, “Using Latent Semantic Analysis to Identify Similarities in Source Code to Support Program Understanding,” *International Conference on Tools with Artificial Intelligence*, Vancouver, 2000.

소프트웨어 프로세스 인증제도의 국방 적용방향

김영봉 유천수

한국국방연구원 국방획득연구센터
서울 동대문구 회기로 37
ybkim@kida.re.kr, cheonsoo@kida.re.kr

요약: 고품질의 SW 개발 보장을 위하여 SW 프로세스 인증제도의 국방 적용방안을 적용사업, 적용사업자, 평가기준, 제도화 방안의 측면에서 수립하였다. 무기체계와 전력지원체계를 포괄하여 일정 규모 이상의 사업비로 개발되며, SW 의 비중이 큰 사업을 대상으로, 참여하는 주사업자와 핵심 임무기능을 개발하는 부사업자에게 SW 프로세스 인증 요구조건을 적용한다. 또한, SW 프로세스 인증제도의 인증 기준 간 관련성을 고려하여 3 개의 평가기준안을 마련하였으며, 현재 제도화되어 있는 제안서 평가 시 적용하는 형태 외에 실제 국방 SW 개발 시 적용하는 형태를 추가 제안하였다.

핵심어: 국방, SW 프로세스 인증, CMMI, SPICE, SP 품질인증

1. 연구배경

SW 의 규모가 커지고 복잡해지면서 SW 의 품질에 영향을 미치는 프로세스의 중요성이 증대되었다. 이에 SW 개발 기업의 입장에서는 조직의 SW 프로세스를 개선하고자 하는 노력이, 사업 발주 기관의 입장에서는 사업에 참여하는 기업의 SW 프로세스 능력을 객관적으로 평가하고자 하는 시도가 나타났다.

방사청은 SW 프로세스의 중요성을 인식하고 제안서 평가 시 조직의 SW 프로세스 인증 등급에 따라 가점을 부여하고 있으나, 무기체계로 그 적용이 제한되어 국방 SW 전체 차원의 정책은 미흡한 실정이다.

본 논문의 목적은 전력지원체계를 포함한 국방 SW 전체 차원에서 SW 프로세스 인증제도의 국방 적용방향을 제시하는 것이다.

이를 위하여 먼저 SW 프로세스 인증제도와 도입 효과에 대해 살펴보고, 다음으로 국방의 관련 현 실태를 분석한 후, 마지막으로 국방 분야에 SW 프로세스 인증제도를 적용하기 위한 방향을 제시한다.

2. 기본 개념

국제전기전자기술자협회(IEEE: Institute of Electrical and Electronics Engineers)는 프로세스란 목적을 달성하기 위해 수행하는 일의 순서로 정의하고 있다[1]. 카네기멜론대학의 SW 공학연구소에 따르면 SW 프로세스란 소프트웨어 및 관련 산출물을 개발하고 유지보수하기 위해 사용하는 일련의 활동, 방법, 실무 및 변형을 의미한다. SW 프로세스 인증이란 SW 프로세스 모델·기준(베스트프랙티스)에 대한 프로젝트 또는 조직의 만족도를 심사·평가하는 활동을 말하며, 이는 조직의 SW 프로세스 능력 평가에 활용될 뿐만 아니라, 조직이 내부의 프로세스를 개선하고자 할 때에도 활용될 수 있다.

주요 SW 프로세스 인증제도로는 산업표준이라 할 수 있는 CMMI (Capability Maturity Model Integration), 국제표준인 SPICE(Software Process Improvement and Capability dEtermination), 국내 인증제도인 SP(Software Process) 품질인증이 있다.

CMMI 는 조직의 프로세스 개선을 돕는 베스트프랙티스의 집합체이며, 현재 시장표준으로써 산업계에 상당한 영향력을 가지고 있다. CMMI 모델은 주요 내용 따라 CMMI-ACQ(CMMI-Acquisition: 프로세스/서비스 획득), CMMI-DEV(CMMI-Development: 프로세스/서비스 개발), CMMI-SVC(CMMI-Service: 서비스 확립/관리)의 3 개 모델로 구분된다. CMMI-ACQ 모델은 프로세스 관리, 프로젝트 관리, 획득, 지원의 4 개 영역, CMMI-DEV 모델은 프로세스 관리, 프로젝트 관리, 공학, 지원의 4 개 영역, CMMI-SVC 는 프로세스 관리, 프로젝트/워크 관리, 서비스 수립 및 제공, 지원의 4 개 영역으로 구성되어 있다. CMMI 기반의 평가 유형에는 클래스 A, B, C 가 있다. 클래스 A 는 조직의 공식 단계(Rating)를 획득하기 위해 사용되며 반드시 공인된 SCAMPI 선임심사원에 의해 수행되어야 한다. 클래스 B 는 조직의 현황을 상세히 진단하고, 클래스 A 의 심사 준비 점검을 목적으로 수행하는 심사를, 클래스 C 심사는 주로 조직의 갭 분석 목적으로 수행하는 간단한 심사를 의미한다[2, 3, 4, 5].

SPICE 란 여러 프로세스 개선 모형을 국제 표준으로 통합한 것으로 프로세스를 개선하고 능력을 결정하기 위한 프로세스 평가 모델이다. SPICE 의 프로세스 모델은 고객-공급자, 공학, 지원, 관리, 조직의 5

개 분야로 구성되어 있다. SPICE 와 관련하여 단일의 평가 모델이 존재하는 것이 아니라 평가 대상에 따라 시스템 수명주기 프로세스 평가 모델, SW 수명주기 프로세스 평가 모델, 자동차 분야 SPICE 평가 모델 등 다양한 모델이 존재한다[6, 7, 8, 9]

SP 품질인증은 국내 기준으로써, SW 기업의 SW 개발 프로세스 품질 역량을 심사하여 등급을 부여하는 제도를 말한다. SP 품질인증의 인증 기준은 조직관리, 프로세스 개선, 프로젝트 관리, 개발, 지원의 5 개 영역으로 이루어져 있다[10].

SW 공학연구소는 2006 년 CMMI 인증 등급을 획득한 35 개 기업을 대상으로 CMMI 기반 프로세스 개선 성과를 조사 분석하여 발표하였다. 분석 결과 비용, 일정, 생산성, 품질, 고객 만족도, 투자대비효과 측면에서 <표 1>과 같이 개선 효과를 거둔 것으로 나타났다[11].

표 1 프로세스 인증 도입을 통한 개선 효과

	개선율			자료 수
	평균(중간값)	최소	최대	
비용	34%	3%	87%	29
일정	50%	2%	95%	22
생산성	61%	11%	329%	20
품질	48%	2%	132%	34
고객 만족도	14%	-4%	55%	7
투자대비효과	4.0 : 1	1.7 : 1	27.7 :1	22

3. 국방분야 현 실태 분석

국방분야의 SW 프로세스 품질인증 관련 현 실태를 적용사업, 적용사업자, 평가기준, 제도화 현황의 측면에서 분석하면 다음과 같다.

먼저 적용사업 측면에서 살펴보면 무기체계 뿐만 아니라 전력지원체계도 고품질의 SW 확보가 중요함에도 현재 방사청의 무기체계 제안서 평가 규정에 따라 무기체계 사업에 한하여 SW 프로세스 인증제도를 적용하고 있어, 국방부 차원에서 무기체계와 전력지원체계를 통합적으로 고려한 정책 방향 수립이 요구된다[12].

SW 프로세스 평가를 적용 받는 사업자 측면에서 살펴보면 방사청 규정에 명시적으로 규정되어 있지는 않으나 사업에 참여하는 주 사업자에 대해서만 인증결과를 제안서 평가 배점에 반영하는 것으로 이해할 수 있다. 국방에서 수행하고 있는 대규모 사업의 경우 주사업자는 통합, 프로젝트 관리 등의 역할을 수행하고, 부사업자가 중요 SW 기능을 개발하여 고품질의 SW 개발 보장에 부사업자의 역량이 중요함에도 불구하고 부사업자에 대한 평가는 부재한 상황이다.

평가기준의 측면에서 살펴보면 CMMI·SPICE 와 SP 품질인증의 인증 기준이 세부적으로 다름에도 이에 대한 고려가 미흡하다. SP 품질인증 2 등급은 CMMI 3 단계 4 개의 프로세스 영역 중 프로젝트 관리와 지원 영역은 포함하지 않음에도 불구하고 <표 2>와 같이 SP 품질인증 2 등급이 CMMI 3 단계와 동일한 환산비율을, SP 품질인증 3 등급은 CMMI 4 단계의 프로젝트 관리 영역과 CMMI 5 단계의 프로세스 관리 영역을 포함하지 않음에도 불구하고 CMMI 4~5 단계와 동일한 환산비율을 얻도록 되어있다.

표 2 인증등급에 따른 가산점 환산비율

인증등급	SP 3/ CMMI·SPICE 4~5	SP 2 / CMMI·SPICE 2~3	미부여
환산비율(%)	100	50	0

마지막으로 제도화 현황을 살펴보면 무기체계 사업의 제안서 평가 시 기업의 SW 프로세스 인증 등급에 따라 배점을 부여하는 형태로 국방에 적용하고 있다. 현재의 제안서 평가 시 배점을 부여하는 형태는 기업의 과거 사업 수행실적에 대한 평가 결과가 기업의 국방 사업 참여 여부에 영향을 미치는 구조이다. 실질적으로는 기업의 우수한 SW 프로세스 역량을 과거 프로젝트가 아닌 국방 사업 수행 시 적용하는 것이 중요함에도 불구하고 이에 대한 제도적 차원의 장려는 제한되는 실정이다.

4. 국방분야 적용방향

본 논문에서는 국방분야에 SW 프로세스 인증제도 적용방향을 앞서 현 실태 분석 시 살펴본, 적용사업, 적용사업자, 평가기준, 제도화 방안의 측면에서 수립하였다.

4.1 적용사업

무기체계 뿐만 아니라 전력지원체계에 들어가는 SW 에 대해서도 고품질의 SW 개발이 중요하다. 이에 기존에 무기체계 사업에만 적용하고 있던 SW 프로세스 인증 제도를 전력지원체계 사업에도 적용하도록 한다. 단, 체계 특성에 따라 SW 의 역할 범위에 차이가 있으므로 모든 사업에 적용하는 것이 아니라 사업 예산이 일정금액 이상인 경우 적용하는 것이 적절하다. 이때에도 SW 의 비중, 체계 중요성 등을 고려하여 최종적으로 적용여부를 판단하고 사업추진 관련 위원회 등을 통해 적용여부 타당성을 승인하도록 한다.

4.2 적용사업자

고품질의 국방 SW 개발을 보장하기 위해서는 주사

업자의 역량뿐만 아니라 부사업자의 역량 또한 중요하므로 부사업자에 대해서도 SW 프로세스 인증 요구조건을 적용하도록 제안한다. 다만, 부사업자마다 개발하는 SW 의 중요도가 차이가 있으므로 모든 부사업자에 대해 인증 요구조건을 적용하는 것이 아니라 핵심 임무기능을 개발하는 부사업자에 대해 적용하도록 한다.

4.3 평가기준

본 논문에서는 각 인증제도의 등급 간 관계성을 고려하여 <표 3>과 같이 개선 대안을 마련하고 장단점을 분석하였다.

대안 1 은 현행의 기준을 유지하는 것으로 이는 SP 등급이 타 SW 프로세스 인증제도에 비해 적은 영역의 인증 기준을 만족함에도 동등한 수준의 평가 배점을 보장함으로써 우리 정부가 추진하고 있는 SP 품질인증에 대한 장려 효과는 있으나 제도간 인증 기준의 차이점을 정확히 반영하지 못하는 한계점이 상존한다.

대안 2 는 CMMI·SPICE 4 단계~5 단계, SP 3 등급, CMMI·SPICE 3 단계, SP 2 등급, CMMI·SPICE 2 단계 순으로 평가 배점을 부여하는 방안이다. 이는 제도간 인증 기준의 차이점을 대안 1 에 비하여 명확히 반영하나 업체로써는 최고 배점을 확보하기 위해 고비용이 소요되는 CMMI, SPICE 심사를 받아야 하는 경제적 부담이 발생한다.

대안 3 은 미국의 사례와 같이 CMMI·SPICE 3 단계와 SP 2 등급에 대해 동일하게 100%의 환산 비율을 부여하는 방안이다. 이는 기업의 과도한 인증 획득 부담을 방지하는 효과가 있으나 대안 1 과 마찬가지로 제도간 인증 기준의 차이점을 정확히 반영하지 못하는 한계점이 있다.

표 3 평가기준 개선방안

대안	세부 내용
1안	현행 유지
2안	CMMI·SPICE 4~5 | SP 3 | CMMI·SPICE 3 | SP 2 | CMMI·SPICE 2
3안	CMMI·SPICE 3, SP 2 이상 등급에 대해 동일하게 100% 부여

4.4 제도화 방안

우수한 SW 프로세스 역량을 갖춘 기업이 국방 사업에 참여하도록 할 뿐만 아니라, 국방 사업에 참여하는 기업이 SW 프로세스 인증제도에서 제시하는 베스트 프랙티스를 따라 국방 사업을 수행하도록 제안서 평가 단계와 국방 SW 개발 단계에 SW 프로세스 인증제도를 적용하도록 한다.

먼저 국방 SW 업체의 SW 프로세스 역량 제고에 대한 관심을 높이고 고품질의 국방 SW 를 확보할 있도록 현 제안서 평가 시 배점을 부여하는 형태에서 특정 등급 이상의 업체만 국방 사업에 참여하도록 의무화하는 방안을 제안한다. 이때, 의무화에 앞서 기업이 인증을 획득할 수 있도록 충분한 유예기간 부여하는 것이 필요하다. 의무화 이후에는 국방 SW 시장의 성숙도를 분석하여 기업의 역량이 어느 정도 성숙되었다고 판단 될 시 의무화 요구 조건을 해지하도록 한다.

또한 참여기업이 수행하는 국방 사업에 대해서도 프로세스 인증을 수행하고, 이를 데이터베이스화하여 관리함으로써 향후 해당 업체가 타 국방 사업 참여하는 경우 업체가 이전 국방 사업에서 받은 평가 결과를 활용하여 참여 여부를 판단할 수 있도록 한다.

5. 결론

소프트웨어 품질의 아버지라 불리는 왓츠 험프리가 “SW 시스템의 품질은 그것을 개발하고 유지보수하는데 사용한 프로세스의 품질에 의해 좌우된다.”고 언급한 것처럼, SW 프로세스는 고품질의 SW 의 개발에 중요 요소로 작용한다.

이에 본 논문에서는 국방 SW 전체 차원에서 SW 프로세스 인증제도의 국방 적용을 위한 정책적 방향 및 제도 마련의 기반을 제시하였다. 이러한 제안이 관계 부서와 기관의 통합적 검토 및 노력을 통해 우리 군의 정책으로 실현된다면 고품질의 국방 SW 획득 제고에 기여할 것으로 기대된다

참고문헌

[1] IEEE, IEEE Standard Glossary of Software Engineering Terminology(IEEE Std 610.12), 1990

[2] CMU/SEI, CMMI for Acquisition, Version 1.3, 2010

[3] CMU/SEI, CMMI for Development, Version 1.3, 2010

[4] CMU/SEI, CMMI for Service, Version 1.3, 2010

[5] CMU/SEI, SCAMPI A, Version 1.3, 2011

[6] ISO/IEC, Information Technology-Process assessment-Process measurement framework for assessment of process capability, 2015

[7] ISO/IEC, Information Technology-Process assessment-Part5: An exemplar system life cycle process assessment model, 2013

[8] ISO/IEC, Information Technology-Process

assessment-Part5: An exemplar software life cycle process assessment model, 2012

[9] VDA QMC, Automotive SPICE Process Assessment/Reference Model, Version 3.0, 2015

[10] NIPA SW 공학센터 홈페이지, http://www.sw-eng.kr

[11] CMU/SEI, Performance Results of CMMI-Based Process Improvement, 2006

[12] 방사청, 무기체계 연구개발사업 제안서평가 및 협상지침, 2014

원자력 계측제어 소프트웨어의 안전성 분석을 위한 Safety Case 의 Arguments 개발 절차

이동아, 유준범

건국대학교 컴퓨터정보통신공학과
서울시 광진구 능동로 120
{ldalove, jbyoo}@konkuk.ac.kr

이장수

한국원자력연구원
대전광역시 유성구 대덕대로 989 번길 111
jslee@kaeri.re.kr

요약: 원자력 계측제어 소프트웨어는 안전 필수 시스템에서 핵심적인 부분이기 때문에 안전성에 대한 분석이 반드시 요구된다. Safety Case 는 안전성 분석 기법으로서 다양한 안전 필수 분야에서 사용되지만, 분석가에 따라 분석의 수준이 결정된다. 특히 소프트웨어의 안전성은 하드웨어와 달리 마모나 확률적 계산에 근거하지 않기 때문에 기존과는 다른 소프트웨어의 안전성을 위한 분석 체계가 요구된다. 본 논문에서는 Safety Case 를 활용한 원자력 계측제어 소프트웨어의 안전성 분석을 위한 Safety Case 의 전략(Strategy) 수립 과정 및 결과를 소개한다.

핵심어: 안전성 분석, 원자력 계측제어 소프트웨어, Safety Case

1. 서론

안전 필수 시스템(Safety Critical System)이란 시스템의 사고로 인해 인명피해나 환경오염과 같이 돌이킬 수 없는 결과를 초래할 수 있는 중대한 시스템을 의미한다. 이런 특징 때문에 시스템이 가진 안전성 또는 위험성 대한 분석을 통해 시스템의 안전성을 높이는 활동이 반드시 필요하다.

본 논문에서는 대표적인 안전 필수 시스템인 원자력 발전소의 계측제어(I&C: Instrumentation & Control) 시스템의 소프트웨어에 대한 안전성 분석(safety analysis)을 수행하는 절차에 대하여 소개한다. 다양한 안전 필수 분야에서 시스템의 안전성 분석 기법으로 주목 받고 있는 Safety Case[1] 기법을 활용하여 원자력 계측제어 시스템의 소프트웨어 안전성 분석을 수행하는 절차에 대하여 소개한다. 이를 위해 원자로 보호 시스템(RPS: Reactor Protection System)의 BP (Bistable Processor) 소프트웨어를 대상으로 안전성 분석을 수행한 결과를 보인다.

Safety Case 는 안전성 분석 기법의 한 종류로서, 시스템이 용인되는 수준의 안전성(acceptably safe)을 갖췄는지 보이기 위한 논증을 명시적이고 구조적으로 표현하는 기법이다. GSN (Goal Structuring Notation) [2]은 이러한 논리적 구조를 표현하는 대표적인 방법이다. <그림 1>은 GSN 의 표기법을 나타낸다. 본 논문에서는 소프트웨어의 안전성 분석을 위해 Safety Case 작성 시 고려해야 할 전략(Strategy)을 도출하는 과정 및 결과에 대해 소개한다.

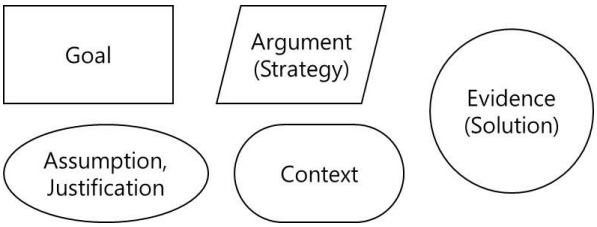


그림 1 GSN 표기법

2. Safety Case 와 GSN 을 이용한 RPS SW 안전성 분석

Safety Case 를 이용한 안전성 분석은 대상의 안전성에 대한 최상위 목표를 설정하는 것으로부터 시작된다. 일반적으로 Safety Case 의 안전성 분석은 '허용 가능한 안전성(acceptably safe)'에 대하여 다룬다. 따라서 Safety case 를 이용한 안전성 분석의 최상위 목표는 '○○○ system is acceptably safe' 형태를 따른다. 이 때, 해당 Goal 에 대한 배경지식과 제약사항이 필요한 경우 Context 와 Assumption 을 명시한다. BP 를 대상으로 최상위 Goal 을 설정한 결과는 다음과 같다.

Goal	G1: *BP is acceptably safe to operate within in PLC*
Context	C1: *BP (Bistable Processor) is a software* *C15: PLC is POSAFE-Q*
Assumption	A1: *Safety demonstration of PLC hardware is already finished by hardware engineers.* A4: *"Safe" means that BP is functionally and non-functionally correct.*

다음으로 설정된 최상위 Goal 을 해결하기 위한 전략을 수립한다. 소프트웨어는 논리적 연산 집합이기 때문에 안전성 분석 수행은 하드웨어와 달리 마

모나 확률적 분석을 대상으로 할 수 없다. BP 안전성 분석의 최상위 Goal 인 G1 의 제약사항 A4 에서 밝혔듯이 소프트웨어의 기능적/비기능적 정확성에 의해 안전성을 분석을 수행한다. BP 의 안전성 분석을 위해서 다음과 같은 4 가지 전략을 제시하였다.

Strategy (Argument)	S1: *Argument over V&V to demonstrate functional correctness*
	S7: *Argument over elimination or mitigation of hazards*
	S11: *Argument over reliability demonstration activities*
	S12: *Argument over software development process*

BP 의 기능적/비기능적 정확성을 보이기 위해 두 가지 전략을 각각 제시하였다. 기능적 정확성을 보이기 위해서는 V&V (Verification & Validation) 활동 및 소프트웨어 내/외부의 위험요소를 제거하는 전략을 수립하였다. 반면, 비기능적 정확성을 보이기 위해서는 소프트웨어의 신뢰성(Reliability)과 개발 절차에 대한 타당성을 보이는 전략을 수립하였다—본 논문에서는 지면의 한계로 인해 S1 에 대해서만 자세히 언급한다.

S1 은 BP 의 기능적 정확성 확인을 통해 안전성을 확보하기 위한 전략이다. BP 의 기능적 정확성을 달성하기 위해서는 BP 내부에 논리적 결함(fault)이 존재하지 않아야 한다. 이를 위해 새로운 하위 목표를 설정하였고, 새로운 하위 목표를 달성하기 위한 전략을 수립하였다.

Sub-goal	G2: *There is no logical fault in the BP*
Strategy	S2: *Formal proof that the software requirement satisfies safety properties*

G2 달성을 위해 정형 기법을 통한 요구사항 검증을 수행하는 전략을 취하고, 해당 전략 달성을 위해 BP 의 기능별 모델체킹 수행(G3– 생략)을 새로운 하위 목표로 설정한다. 최종적으로 G3 달성을 위한 해결책을 아래와 같이 제시하였다.

Evidence (Solution)	Sn1: *A V&V plan of software requirement* Sn2: *A V&V report of software requirement*

위에 소개한 절차를 통해 도출된 Safety Case 는 많은 양의 자료를 텍스트 형태로 만들어 낸다. Goal 과 Strategy, Evidence 라는 구조를 가지고 있으나 그 구조를 한 눈에 파악하기 쉽지 않다. GSN 은 Safety Case 의 구조를 쉽게 파악하기 위해, 그 구조를 도식화할 수 있도록 제안된 표현 기법이다. 본 장에서 도출한 BP 의 안전성 분석을 위한 최상위 Goal 은 그림 3 과 같이 표현할 수 있다(GSN 작성 도구: Adelard 社의 ASCE (The Assurance and Safety Case Environment)[3]).

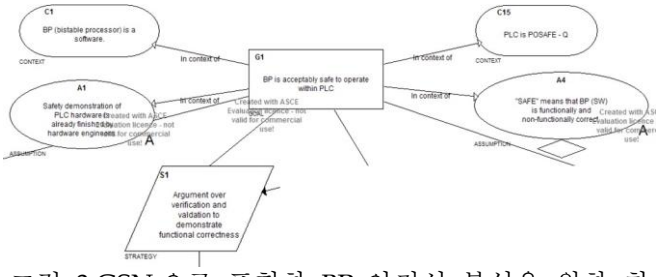


그림 3 GSN 으로 표현한 BP 안전성 분석을 위한 최상위 Goal

최상위 Goal 인 G1 을 중심으로 G1 을 설명하기 위한 배경지식(C1, C15)과 제약사항(A1, A4)이 연결된 것을 볼 수 있다. G1 해결을 위한 Strategy 중 하나인 S1 이 그 아래쪽에 연결 되어 있으며, S1 을 해결하기 위한 Sub-goal 및 Evidence 는 그림 4 와 같다. 이처럼 GSN 을 사용하면 Safety Case 구조를 보다 효과적으로 나타낼 수 있으며, 쉽게 파악할 수 있다.

3. 결 론

본 논문에서는 원자력 계측제어 소프트웨어의 안전성 분석을 위해 Safety Case 작성을 수행하는 절차에 대하여 소개하였다. Safety Case 및 GSN 을 활용한 안전성 분석은 분석가의 역량에 따라 큰 차이를 보인다. 이러한 차이를 좁히기 위하여 소프트웨어의 안전성 분석을 위해 사용되어야 할 네 가지 전략을 도출하는 과정 및 결과에 대해 소개하였고, 하드웨어 기반 시스템의 안전성 분석과의 차이를 보여주었다.

Safety Case 를 작성에서 가장 어렵고 중요한 부분이 바로 Goal 해결을 위한 타당한 Strategy 를 도출하는 부분이다. 본 연구팀은 원자력 계측제어 소프트웨어에 대한 안전성 분석 시 타당하고 효과적인 Strategy 를 사용할 수 있도록 패턴을 개발하는 연구를 향후에 수행할 예정이다.

사 사

본 연구는 한국원자력연구원의 "원자력 계측제어 계통 안전 적합성 평가체계" 사업의 지원으로 연구한 결과 입니다.

참고문헌

[1] “The purpose, scope, and content of safety cases”, Office for Nuclear Regulation, An agency of HSE
[2] http://www.goalstructuringnotation.info/
[3] http://www.adelard.com/asce/