The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 15, No. 4, pp.9-18, Aug. 31, 2015. pISSN 2289-0238, eISSN 2289-0246

http://dx.doi.org/10.7236/JIIBC.2015.15.4.9
JIIBC 2015-4-2

레거시 소프트웨어 시스템을 위한 문맥 독립적 행위 기반 실시간 오작동 탐지 기법

Runtime Fault Detection Method based on Context Insensitive Behavioral Model for Legacy Software Systems

김순태*

Suntae Kim*

요 약 최근에는 스마트폰과 같이 임베디드 형태로 다양한 장소에서 서비스를 제공하는 어플리케이션의 수가 늘어나는 추세이다. 기존의 고정된 장소에서의 실행 환경보다 서비스 실행 중 상태가 동적으로 변할 수 있다는 점으로 인해 실행 중 오작동이 발생할 수 있다. 이 문제를 다루기 위하여 본 연구에서는 레거시 소프트웨어 시스템을 대상으로 메서드 수준의 오작동 탐지 기능의 구축기법을 제안한다. 기존의 문맥 의존적 행위 모델 기반으로 비정상 행위를 탐지하는 방식 메서드 수준의 탐지에 적용 시 거짓 양성의 발생 비율 증가, 모니터링 오버헤드 증가 등의 문제가 발생 가능하다. 이를 향상하기 위해 본 연구에서는 문맥 독립적 행위 모델 기반 오작동 탐지(Context-Insensitive Behavior Model-based Failure Detection, CIBFD) 기법을 제안한다. 사례 연구를 통해 기존 연구 대비 탐지 결과를 비교 분석하고, 어플리케이션 도메인 별 기법의 효용성을 분석한다.

Abstract In recent years, the number of applications embedded in the various devices such as a smart phone is getting larger. Due to the frequent changes of states in the execution environment, various malfunctions may occur. In order to handle the issue, this paper suggests an approach to detecting method-level failures in the legacy software systems. We can determine if the software executes the abnormal behavior based on the behavior model. However, when we apply the context-sensitive behavior model to the method-level, several problems happen such as false alarms and monitoring overhead. To tackle those issues, we propose CIBFD (Context-Insensitive Behavior Model-based Failure Detection) method. Through the case studies, we compare CIBFD method with the existing method. In addition, we analyze the effectiveness of the method for each application domains.

Key Words : Legacy Software System, Fault Detection, self-adaptive software

Ⅰ. 서 론

최근에 소프트웨어는 다양한 오픈 컴포넌트를 기반으로 하고, 오픈 응용 프로그램(OTS: Off-The-Shelf)을 사용하여 개발한다. 하지만, 많은 OTS 컴포넌트 혹은 응용 프로그램은 소스 코드와 자세한 개발 산출물을 제공하는 경우보다, 사용법에 대한 간략한 정보만 제공하

*정회원, 전북대학교 소프트웨어공학과
접수일자 2015년 7월 28일, 수정완료 2015년 8월 7일
게재확정일자 2015년 8월 7일

Received: 28 July, 2015 / Revised: 7 August, 2015 /
Accepted: 7 August, 2015
*Corresponding Author: stkim@jbnu.ac.kr
Dept of Software Engineering, Chunbuk National University, Korea

는 경우가 많다[1]. OTS 소프트웨어를 사용하여 시스템을 개발 시 충분치 않은 정보로 인하여 실시간에 오작동 발생 시 이에 원인과 상세한 정보를 파악하기가 쉽지 않다[2]. 실시간 오작동 탐지기법은 이런 이슈를 해결하기 위하여 사용 할 수 있는 방법중 하나이다.

실시간 오작동 탐지 기법이란 실시간에 대상 시스템을 모니터링하여 정상동작과 다른 행위를 감지하여 오작동을 탐지하는 방법이다[3]. 실시간 오작동 탐지를 위해서는 다음의 사항이 고려되어야 한다. 첫째, 탐지된 오작동에 대한 가능한 상세한 위치 정보를 제공해야한다[4]. 오작동에 대한 정보의 제공은 추후, Self-Healing, 진보된 시스템 관리, OTS를 차용하는 시스템 개선등 다양한 의사 결정을 취할 수 있도록 도와준다. 둘째, 오작동의 발생 원인을 제공해야한다[5]. 최근 소프트웨어는 재사용성을 높이기 위하여 하나의 소프트웨어 모듈이 여러 기능 구현에 참여한다. 이 부분에 오작동발생시 발생 위치만으로는 그 발생 원인을 파악하기 어렵다[6]. 마지막으로, 소프트웨어 수행중에 오작동을 탐지해야하므로 성능상의 오버해드를 가능한 최소화 해야한다[7].

오작동 탐지를 위한 많은 기존 연구가 있어왔다. 일반적으로, 오작동을 탐지하기 위하여 정상 동작에 대한 모델을 구축하고, 현재 시스템을 모니터링 한 후 이 결과와 정상 동작 모델을 비교하여 오작동을 탐지한다. 정상 동작 모델을 구축하는 방법에 따라 접근 방안은 명세기반 탐지 기법과 히스토리 탐지 기법으로 분류할 수 있다. 첫째, 명세기반 탐지는 사용자 요구사항, 아키텍처, 세부 설계 등의 산출물이 정상 동작 모델링의 기반으로 사용된다[8][9]. 이 기법은 명세 문서가 부족할 경우 적용하기 어렵고, 명세와 구현 간의 연계성을 찾기 위한 비용이 크다는 문제점을 가진다. 둘째, 히스토리 기반 탐지는 소프트웨어 실행 로그 히스토리에서 특정 기능 수행 시 일관적으로 발생하는 로그 패턴을 찾아 모델을 구축한다[10][11]. 그리고, 모니터링된 값과, 구축된 모델의 비교를 통하여 오작동을 탐지한다. 이 기법은 로그 기반으로 귀납적 추출한 패턴이 소프트웨어가 갖는 본래의 기능성을 적절히 반영하지 못하는 문제로 인해 거짓 양성(False Alarm)이 발생할 수 있다는 문제점을 갖는다. 이는 실행 로그의 복잡도가 커질수록 추론의 어려움 증가로 인해 거짓 양성 문제가 더 커질 수 있다.

위의 이슈를 다루기 위해서 본 논문에서는 히스토리 기반 탐지 기법을 확장한 문맥 독립적 행위 모델 기반 오작동 탐지 기법을 제안한다. 문맥 독립적 행위 모델이란 실행 히스토리로그를 통해 구축된 행위 모델이라는 점에서는 기존 연구와 동일하나, 실행시 현재 실행 상태 정보 유지 하지 않기 때문에 문맥 독립적이다. 이를 구축하기 위하여 시스템의 사용 기능 별로 행위 모델을 추출하고, 기능의 행위 모델 간 차이를 분석하여 기능에 특화된 행위 모델을 추출하여 모니터링의 기준으로 채용한다. 이렇게 잘게 나뉘어진(Fine-grained) 행위 모델을 통하여 오작동 탐지 소프트웨어는 대상 소프트웨어에 대한 현재 실행 상태를 유지 않아도 실행 문맥을 파악 가능하다.

논문에서는 Paint App의 오작동을 탐지하는 사례를 통해 접근 방안의 타당성을 보이고자 한다. 본 연구의 기여는 크게 세가지로 정리할 수 있다. 첫째, 탐지된 오작동의 발생 위치에 대하여 메소드 수준으로 상세한 정보를 제공한다. 둘째, 오작동의 발생 원인에 대한 정보를 제공한다. 마지막으로, 실행시간에 오류를 탐지하기 위하여 오버헤드를 줄이기 위한 방법을 제안한다.

본 연구는 아래와 같이 구성된다. 2장에서는 히스토리 기반 오작동 탐지 기법에 대한 관련 연구를 설명하고, 3장에서는 문맥독립적인 행위 모델 기반 오작동 탐지 기법을 제안한다. 4장에서는 사례 연구를 통해 연구 내용을 검증하며, 5장에서는 결론을 소개하며 본 논문을 마무리한다.

II. 관련 연구

이번 장에서는 오작동을 탐지를 위한 선행 연구에 대하여 소개한다.

1. 명세기반 오작동 탐지 기법

명세기반 탐지는 사용자 요구사항, 아키텍처, 세부 설계 등의 산출물이 정상 동작 모델링의 기반으로 사용된다[8][9]. 이중 가장 핵심적으로 사용되는 모델 중 하나가 상태 머신이다. 상태 머신은 실행 소프트웨어 내의 상태 전이들을 표현하기 위해 상태를 표현하는 노드와, 상태 전이를 표현한 엣지로 구성되는 UML모델이다. 상태 머신을 기반으로 한 접근 방법은 현재 상태 정보(노드)를 보유하고 추후 발생하는 이벤트에 따라 어떤 상태로 전이 될지 결정하는 방식으로, 항상 실행 문맥(Context)을 파악하고 있다. 즉, 이는 문맥 의존적 상태 노드들로 구

성된 행위 모델로, 본 논문에서 이를 문맥 의존적 행위 모델(Context-Sensitive Behavior Model)이라고 부른다. 기존의 문맥 의존적 행위 모델 기반의 상태 추적 기법들은 모니터링 대상 메서드의 수가 증가할 경우 크게 두 가지 문제점이 발생한다.

첫 번째, 오작동 탐지의 거짓 양성 문제의 발생이 증가할 수 있다. 모니터링 대상 메서드의 수가 증가할수록 발생 가능한 상태 전이의 경우의 수는 증가한다. 로그 상에 있는 특정 실행 시퀀스들을 기반으로 행위 모델을 추출했기 때문에 해당 행위 모델과 다른 상태 전이가 감지되면 오작동이 발생된 것으로 잘못 판별하여 거짓 양성이 발생할 수 있다. 두 번째로, 모니터링 오버헤드가 증가할 수 있다. 문맥 의존적 행위 모델 기반의 상태 추적 방식은 대상 메서드들의 실행 이벤트들을 지속적으로 지켜봐야 하기 때문이다.

기존 연구들은 컴포넌트 인터페이스에 해당하는 메서드만을 모니터링하는 것을 가정하였기 때문에 위의 두 문제가 발생되지 않았을 수 있다. 그러나, 레거시 소프트웨어의 경우 설계 내부를 정확하게 파악하지 못하는 상황에서 주요한 인터페이스를 파악하기 어렵기 때문에 현실적으로 오작동 탐지를 위해서는 많은 수의 메서드들을 대상으로 모니터링 해야 한다.

2. 히스토리 기반 오작동 탐지 기법

히스토리 기반 오작동 탐지 기법은 사용 히스토리 로그 분석을 통하여 동적 불변 속성을 추출하고 이를 기준으로 모니터링하여 실행 상태가 동적 불변 속성을 준수하는지 분석함으로써 오작동을 탐지하는 방법이다[10][11]. 동적 불변 속성(Dynamic Invariant) 이란 실행 중인 소프트웨어의 실행 내역을 동적 분석(Dynamic Analysis)하여 식별한 일관된 행위 속성을 의미한다[14]. 예를 들어, 이미지 파일들을 보여주는 어플리케이션의 시작 화면은 썸네일 이미지들이 우측 방향으로 순차적으로 로딩되는 순차적인 행위 속성이 이에 해당될 수 있다. 즉, 구현한 코드에 의해 순차적으로 관찰되는 행위 속성을 가리킨다.

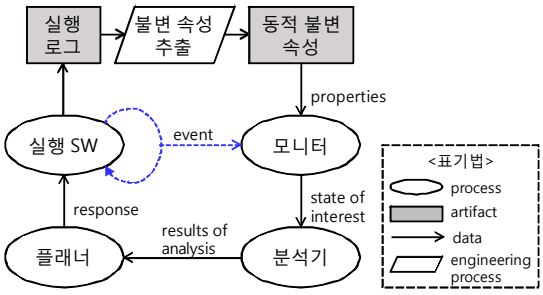

그림 1. 히스토리 기반 오작동 탐지
Fig. 1. Fault Detection based on History

그림 1에서는 히스토리 기반 오작동 탐지기법의 구성요소를 도식화하였다. 이때, 실행 SW의 성공적인 실행 로그를 기반으로 동적 불변 속성을 추출하여 런타임 모니터링의 기준으로 이용한다. 이러한 정상 실행 로그를 기반으로 추출된 동적 불변 속성을 이용해야 정상/비정상 여부를 구분할 수 있기 때문이다. 히스토리 기반 탐지 기법은 SW의 동적 불변 속성을 한정된 양의 실행 로그로부터 귀납적으로 추출하기 때문에 추출한 동적 불변 속성은 정확도 및 종류에 한계가 있을 수 밖에 없다. 동적 불변 속성의 정확도 문제는 오작동 탐지 정확도 문제를 발생시키기 때문에 특히 중요하다.

오작동 탐지 정확도 문제에는 거짓 양성, 거짓 음성 유형이 있다. 거짓 양성(False Positive, 또는 False Alarm)은 실제로 오작동이 발생하지 않았으나 오작동이 발생한 것으로 탐지되는 경우를 뜻하고, 거짓 음성(False Negative)은 실제로 오작동이 발생하였으나 정상적으로 실행 중인 것으로 탐지되는 경우를 뜻한다. 오작동 탐지 기능에서 거짓 양성이 발생할 경우, 잘못된 오작동 탐지 보고 출력으로 인해 불필요한 런타임 오버헤드가 발생할 수 있다. 거짓 음성이 발생할 경우, 오작동 발생에 대처하기 어렵다는 문제가 발생한다. 히스토리 기반 탐지 방식은 탐지 정확도 문제를 가지고 있음에도 불구하고, 현실적으로 OTS 소프트웨어에 적용하기 용이하기 때문에 본 연구에서는 이 방식을 채용하고자 한다.

동적 불변 속성에는 데이터 불변 속성(Data Invariant), 순차적 불변 속성(Sequential Invariant)이 있으며[14], 본 연구는 순차적 불변 속성을 다룬다. 순차적 불변 속성을 기반으로 오작동을 탐지하는 기존 연구들은 실행 로그를 기반으로 K-tail, Markov, RNet 등의 기

법들을 이용하여 상태 머신(State Machine) 형태의 행위 모델을 추출한 후, 실행 중인 소프트웨어가 해당 행위 모델에 따라서 작동하는지 검사하여 오작동을 탐지한다[10-12].

Ⅲ. 문맥 독립적 행위 모델 기반 오작동 탐지 기법

1. 접근 방법 개요

기존의 연구의 문제점을 다루기 위해 본 연구는 문맥 독립적 행위 모델 기반의 탐지 기법(CIBFD : Context-Insensitive Behavior model-based Failure Detection)을 제안한다. 문맥 독립적 행위 모델(CIB Model: Context-Insensitive Behavior Model)이란 소프트웨어의 실행 문맥에 관계 없이 특정 이벤트 이후에 반복적으로 발생한 일련의 이벤트 패턴을 표현한 행위 모델을 의미한다. 여기에서 일련의 이벤트 패턴을 순차적 행위 패턴이라고 부른다. 실행 상태와 관계 없이 반복적으로 관찰되는 일련의 이벤트 시퀀스를 의미한다. 제안하는 탐지 기법은 CIB 모델에 포함된 순차적 행위 패턴과 모니터링으로 관찰된 이벤트 패턴을 비교하여 문제를 탐지한다.

제안 기법은 크게 세가지 단계로 구성된 프로세스에 의해 구현될 수 있다(그림 2 참조)

- CIB 모델 추출: 소프트웨어의 기능 별로 CIB 모델을 추출한다. (작업 1~4, 산출물 A, B)
- 각 기능의 특화 CIB 모델 추출: 각 기능의 CIB 모델 간 차이를 분석하여 기능에 특화된 CIB 모델을 추출한다. (작업 5, 산출물 C)
- 순차적 행위 패턴 선정 및 분석기 구현: 기능 특화 행위 모델 내에 포함된 순차적 행위 패턴을 추출하여 모니터링 기준으로 채용한다. (작업 6~7, 산출물 D)

2. 문맥 독립적 행위 모델 추출

SW의 실행 히스토리 로그를 추출하기 위하여 프로브(Probe)를 사용한다. 프로브 삽입용 도구로서 본 연구에서는 TPTP(Test and Performance Tools Platform)가 제공하는 Probe Kit을 활용하였다. 이를 이용하여 모니터링 대상 메서드들의 진입 부위, 리턴 부위에 프로브를 삽입하여 메서드 실행 정보를 로그로 출력하도록 한다. 이 로그는 <Thread Id, Call Depth, Class Name, Method Name, Time Stamp>으로 구성 된다.

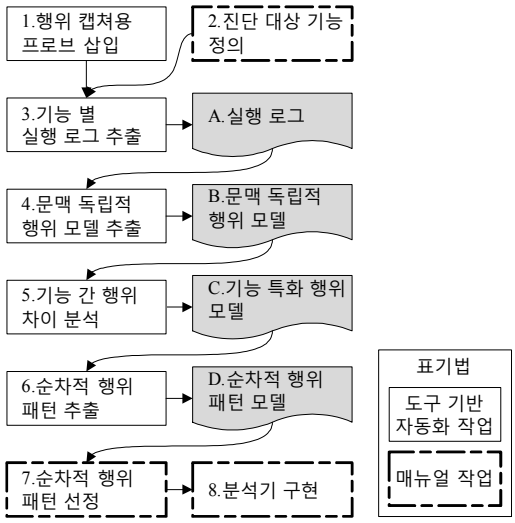

그림 2. CIB 모델 기반 탐지 기법 구현 프로세스
Fig. 2. A Process for Implementing Fault Detection based the CIB model

다음으로 진단 대상 기능의 집합을 정의한다. 진단 대상으로 선정 가능한 기능은 실행 SW가 사용자의 요청에 대한 명확한 수행 완료를 확인할 수 있어야 한다. 이는 실행 중에 어떤 기능의 실행 중에 문제가 발생했는지 파악하기 위한 단위로서 활용한다. 시스템의 가능한 모든 기능을 실행하여 로그 파일을 저장 후, 각 기능 별로 CIB 모델을 추출한다.

그림 3은 실행 로그로부터 CIB 모델을 추출하는 과정을 예를 들어 보여준다. 이해를 돕기 위하여 텍스트 형식의 실행 로그를 그림 3(a)의 UML Sequence Diagram으로 도식하였다. 이 흐름에서 시스템은 사용자의 요청을 완성하기 위하여 f1, f2을 호출하고, f2의 실행 결과에 따라서 f3의 호출 횟수가 달라진다. 이를 CIB 모델로 표현하면 그림 3(b)와 같다. 이를 추출하기 위하여 실행 이벤트(호출 메소드)를 노드로 추출하고, 이벤트의 선후 발생 관계를 엣지로 표현한다. 이는 외부에서 관찰되는 메서드 실행 이벤트를 근거로 캡쳐한 모델이 된다.

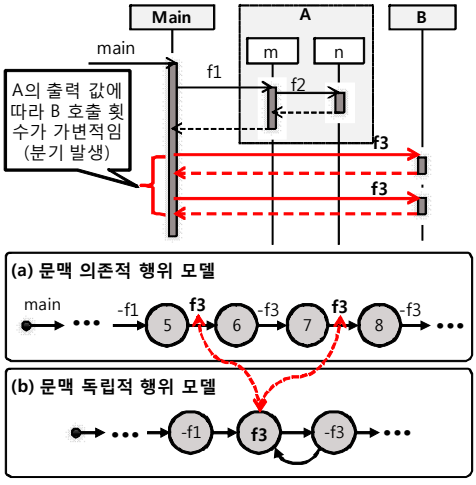

그림 3. 문맥 의존적 vs. 문맥 독립적 행위 모델
Fig. 3. Context Sensitive Vs. Context Insensitive Behavioral Model

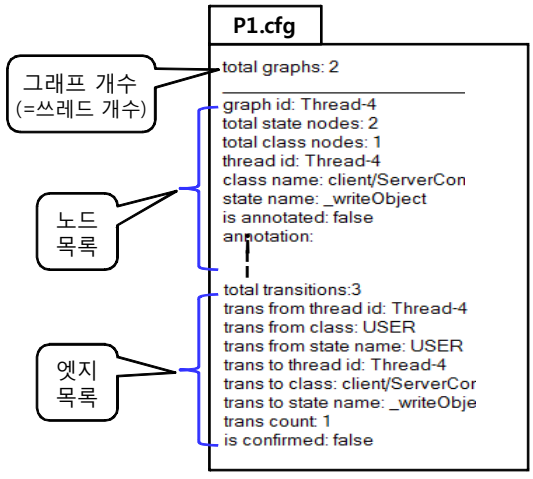

그림 4. 문맥 독립적 행위 모델 파일 내부 구성
Fig. 4. A internal structure of a file for a context insensitive behavioral model

CIB모델은 다음과 같이 정의할 수 있다.

- CIB(s, x, N, E): 시작 노드 s, 종료 노드 x, 노드 집합 N, 엣지 집합 E ⊆ N x N

a) 노드 m ∈ N, 메서드 m의 시작/완료 상태 (시작 상태 값: m, 완료 상태 값: -m)

b) 엣지 e ∈ E, 방향성을 가지는 상태 전이 (속성 값: 상태 전이의 발생 횟수).

로그 기반 CIB 모델 추출 알고리즘은 표 1과 같으며, 그림 4는 결과물 파일 내부 구성을 보여준다.

표 1. CIB 추출 함수
Table 1. A Function for Building the CIB model

function CIB *makeCIB*(L)
Input: SW의 임의 기능 p의 실행 로그 L = <L_1,…,L_n> 임의 L_i는 메서드의 실행 시작 또는 완료 상태의 기록임 L_i=<ThreadId,ClassName,MethodName>
1: create empty CIB 'M_p'
2: create empty node 'previousState' and 'currentState'
3: create empty edge 'e', e = <node a, node b>
4: **for each** $L_i$in**L do**
5: currentState ← L_i
6: **if** currentState is in M_p**then**
7: edge←<previousState,currentState>'s frequency ++
8: **else**
9: add currentState to M_p
10: e ← <previousState, currentState>
11: add e to to M_p
12: **end if**
13: previousState ← currentState
14: **end for**
15: **return** M_p

3. 각 기능의 특화 행위 모델 추출

각 기능의 특화 CIB 모델은 각 기능의 행위 모델 간 차이를 분석함으로써 추출할 수 있다. 그림 5는 기능 특화 CIB 모델을 추출하는 방법을 보여준다.

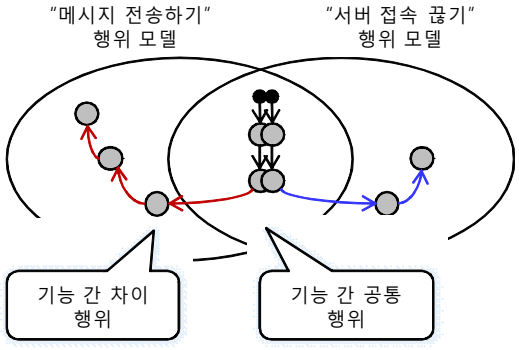

그림 5. 기능 특화 행위 추출
Fig. 5. Extracting a behavioral model for a specific function

CIB 모델 A와 CIB 모델 B가 있을 때, 두 모델간의 차를 의미하는 Diff(A, B)는 A를 기준으로 B에 공통적으로 포함된 행위를 제외한 CIB 모델을 말한다. 즉, Diff(A, B)는 A, B의 엣지 집합 간의 여집합(Edge(A) - Edge(B))을 의미한다(그림 6 참조). Diff 함수는 표 2와 같다.

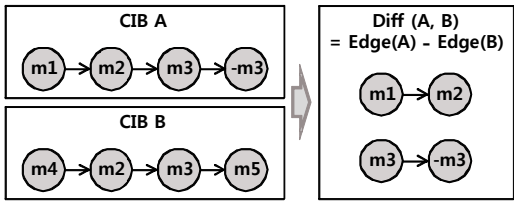

그림 6. CIB 모델 간 차이 추출 함수(Diff(A, B))
Fig. 6. A function for extracting difference between two CIB models(Diff(A, B))

표 2. Diff 함수 Pseudo Code
Table 2. A Pseudo Code for the Diff Function

function CIB *diff*(M_a,M_b)

Input: SW의 기능 a의 CIB M_a,기능 b의 CIB M_b
create empty CIB 'M_{diff}'
create empty flag 'isInvalidNode'
M_{diff}ß M_a
for each edge $e_i$inM_{diff}'sEdgeSet**do**
if $e_i$is inM_b'sEdgeSet**then**
remove $e_i$fromM_{diff}'sEdgeSet
end if
end for
for each node $n_j$inM_{diff}'sNodeSet**do**
isInvalidNode ß true
for each edge $e_k$inM_{diff}'sEdgeSet**do**
if $n_j$ise_k'sfromNodeortoNode**then**
isInvalidNode ß false
end if
end for
if isInvalideNode is true **then**
remove $n_j$fromM_{diff}'sNodeSet
end if
end for
return M_{diff}

3개 이상의 기능 별 CIB 모델이 있을 경우, 일련의 Diff 함수를 통해서 기능 특화 CIB 모델을 구할 수 있다. 예를 들어, 기능 A, B, C가 있을 경우, A의 특화 행위는 Diff(Diff(A, B), C)에 해당된다.

4. 순차적 행위 패턴 선정 및 분석기 구현

마지막 단계에서는 기능별 특화 CIB 모델 내에 포함되어 있는 순차적 행위 패턴을 추출 후 모니터링 및 검사 기준 패턴들을 선정한다. 선정된 순차적 행위 패턴들을 기준으로 분석기를 구현함으로써 오작동 탐지 기능의 구현은 완료된다. 본 연구에서 정의하는 순차적 행위 패턴 유형은 아래 표 3과 같다.

실행 중에 순차적 행위 패턴에 위배되는 경우는 다양하다. 싱글 쓰레드 내 순차적 불변 속성의 경우, java 예외 이벤트가 발생하는 사례가 대표적인 예이다. 특히, 다중 쓰레드 상에서의 문제는 진단하기 어렵고 복잡하다. 쓰레드 간 록(lock)을 얻는 방식으로 구현할 경우 문제가 적게 발생할 수 있으나, 성능 문제 때문에 부분적으로는 동기화 방식을 적용하지 않는 구현 방식을 채용하는 경우도 있기 때문에 순차적 행위 패턴을 위배하는 문제가 발생할 수도 있다. 개발자 관점에서, 다중 쓰레드 기반의 구현은 난이도가 있기 때문에 설계 결함이 다수 포함될 수 있다. 성능도 보장하면서, 안정적으로 구현하기가 쉽지 않기 때문에 설계 결함으로 인해 순차적 불변 속성을 위배하는 비정상 행위가 발생할 수 있다.

표 3. 순차적 행위 패턴 유형
Table 3. Types of Sequential Behavioral Patterns

유형	설명
다중 쓰레드 간 순차적 행위 패턴	다중 쓰레드 간 행위의 시작/종료가 동기화된 패턴 예)생산자, 소비자 쓰레드
싱글 쓰레드 내 순차적 행위 패턴	특정 기능을 구현하기 위한 일련의 메서드 실행 순서

순차적 행위 패턴을 CIB 모델 기반으로 정의하면 아래와 같다.

- 순차적 행위 패턴(SBP: Sequential Behavior Pattern): CIB 모델 M 내 임의 노드 이후에 연속적으로 상태 전이(transition)가 가능한, 서로 다른 m개의 메서드 실행 이벤트(노드)로 구성된 순서, < N1, …, Nm >
  a) SBP를 구성하는 노드 집합 N, N ⊆ M
  b) Nm을 제외한 < N1, …, Nm-1 >이 문맥 의존적 노드를 포함하지 않을 경우, 불변 순차적 행위 패턴이라고 부른다.
  c) m의 최소값은 1이며, 순차적 행위 패턴 단위(Sequential Behavior Pattern Unit, SBP-Unit)라 한다.

다중 쓰레드 간 순차적 행위 패턴은 CIB 모델로는 추출하기 어렵다. 쓰레드 별 행위를 캡쳐한 모델이기 때문이다. 따라서 다중 쓰레드 상에서 발생하는 메서드 실행 이벤트 순서를 추출하기 위해 확장 모델이 필요하다. 다중 쓰레드들로부터 추출한 실행 로그로부터 다음과 같은 MCIB(Multi-threaded Context Independent Behavior) 모델 을 추출할 수 있다.

- MCIB(s, x, N, E): 노드 집합 N, 엣지 집합 E ⊆ N x N

a) 노드 m ∈ N, 쓰레드 t내 메서드 m의 시작/완료 상태 (시작 상태 값: <t, m>, 완료 상태 값: <t,-m>)

b) 엣지 e ∈ E, 방향성을 가지는 순차적 의존 관계 (속성 값: 순차적 의존 관계가 관찰되는 횟수).

MCIB의 엣지는 CIB와 달리, 전이 관계가 아닌, 순차적 관계를 나타낸다. 즉, 발생 이벤트 순서를 추출한 모델이다. 순차적 행위 패턴 단위의 집합은 CIB 모델의 엣지 집합과 동일하므로 별도의 알고리즘은 필요하지 않으며, 불변 순차적 행위 패턴 단위 집합을 추출하기 위한 알고리즘은 다음과 같다.

표 4. 순차적 불변 속성 집합 추출 함수 Pseudo Code
Table 4. A pseudo code of a function for extracting a sequential invariant attributes

function SBPUnitSet *getInvSBPUnitSet*(M)

```
Input: CIB 모델 M
1:      create empty SBPUnitSet P
2:      create empty SBPUnit u
3:    for each edge eiinM'sEdgeSetdo
4:        if isCSNode(M, ei'sfromNode)isfalsethen
5:            u ß < ei'sfromNode,ei'stoNode>
6:            add u to P
7:        end if
8:    end for
9:    return P
```

P1의 쓰레드 간 순차적 행위 패턴

```
AWT-EventQueue-1,client/MessagingClient$MyKeyListener,-keyReleased ==(3)==|
AWT-EventQueue-1,client/MessagingClient$MyKeyListener,-keyTyped ==(3)==|> T
AWT-EventQueue-1,client/CommandHistory,-add ==(1)==|> Thread-4,client/Serve
AWT-EventQueue-1,client/ServerConnection,-writeObject ==(1)==|> Thread-4,cl
```

P1의 쓰레드 내 순차적 행위 패턴

```
Thread-4,client/ServerConnection,-_writeObject ==(47)==|> Thread-4,client/S
AWT-EventQueue-1,client/MessagingClient$MyKeyListener,-keyReleased ==(1)==|
AWT-EventQueue-1,client/MessagingClient,-sendChat ==(48)==|> AWT-EventQueue
AWT-EventQueue-1,client/CommandHistory,add ==(48)==|> AWT-EventQueue-1,clie
AWT-EventQueue-1,client/MessagingClient$MyKeyListener,keyPressed ==(17)==|>
```

그림 7. 순차적 행위 패턴 파일 예시
Fig. 7. An example file of a sequential behavioral pattern

순차적 행위 패턴 중 모니터링 기준을 선정한 후에는 관련 메서드들 내부에 모니터링 프로브를 삽입해야 한다. 이를 위해 관련 메서드들을 추출해야 한다. 순차적 행위 패턴에 해당되는 메서드들에게만 프로브를 삽입할 경우, 문제가 발생 시 관련 이벤트를 관찰할 수 없을 수 있다. 따라서 순차적 행위 패턴에 해당하는 메서드 외에 비정상 행위에 참여하는 이벤트가 발생할 수 있는 메서드들에도 프로브를 삽입해야 한다.

이를 위해서는 비정상 행위에 어떤 메서드들에서 이벤트 발생 가능한지를 파악해야 한다. 즉, 오작동이 어떻게 발생하는지에 대한 정보도 알아야 한다. 소프트웨어 디버깅을 통해서 해당 문제를 해결한다고 해도 이후 동일한 패턴의 오작동이 발생할 수 있다. 이후에도 특정 사례에 대한 미처리로 인해서 오작동이 발생할 경우 동일한 패턴으로 오작동이 발생할 가능성이 높다. 따라서 버그가 있던 이전 버전에서의 비정상 행위에 대한 로그를 기반으로 행위 모델을 추출해야 한다.

비정상 행위 모델과 정상 행위 모델 간 Diff 분석을 통해서 비정상 행위의 특화 행위 모델을 추출한다. 비정상 행위 특화 행위 모델 내에서 순차적 행위 패턴과 공통으로 가지는 노드 이후에 발생 가능한 노드에 해당하는 메서드에 프로브를 삽입하면 된다. 이를 구현하는 알고리즘은 아래 표 5와 같다.

표 5. 비정상 이벤트가 발생 가능한 메서드 목록 추출 함수 Pseudo Code
Table 5. A pseudo code of a function for extracting methods that is likely to happen abnormal events

function NodeSet *getNodeSetForMonitoringAbnormalEvent*(M_{normal}, $M_{abnormal}$)

```
Input: 임의 기능 p의 정상 행위 모델 Mnormal,비정상 행위 모델 Mabnormal
1:      create empty NodeSet N
2:      create empty CIB D
3:      create empty SBPUnitSet S
4:      D ß Diff(Mabnormal,Mnormal)
5:      S ß getInvSBPUnitSet(Mnormal)
6:    for each edge eiinD'sEdgeSetdo
7:        if ei'sfromNodeisinS'sNodeSetthen
8:            add ei'sfromNodetoN
9:            add ei'stoNodetoN
10:       end if
11:   end for
12:   if N is empty then
13:       for each edge ejinD'sEdgeSetdo
14:                                          if
ej'sfromNodeisinMnormal'sNodeSetthen
15:               add ej'sfromNodetoN
16:               add ej'stoNodetoN
17:           end if
18:       end for
19:   end if
20:   return N
```

Ⅳ. 사례 연구

본 논문에서는 연구 결과에 대한 검증을 위하여 Windows의 Paint Application을 대상으로 사례연구를 수행하였다. 사례 연구에서는, 기존의 연구와 탐지 결과를 비교 분석하였으며, 제안 기법이 행위 모델의 상태 복잡도를 어느 수준까지 해결할 수 있는지 평가하였다.

1. 기존 연구 대비 오작동 탐지 결과 비교 분석

문맥 의존적 행위 모델 기반 상태 추적 방식 기법 중의 하나인 K-behavior 기법[14]과 본 연구에서 제안하는 CIBFD 기법을 이용하여 어플리케이션을 위한 오작동 탐지 기능을 구축한다. 어플리케이션의 알려진 오작동(표 6 참조)을 재현하여 오작동 탐지 여부를 확인 후 탐지 결과를 아래 기준에 따라 비교 분석한다.

- 탐지 정확도: 동일 기능에 대해 오작동 발생 시나리오와 정상 작동 시나리오를 실행하여 오작동을 정확하게 탐지해내는지 확인한다.

표 6. 어플리케이션의 알려진 오작동
Table 6. Known Faults of the application

도메인	App	Service	오작동	오작동 유형
Window GUI App	Paint App	파일 저장	디스크 저장 공간이 부족할 경우, 오작동 발생	특정 케이스의 미처리로 인한 오작동

우선, 실행 로그를 추출하여 K-behavior 기법, CIBFD 기법에 의해 행위 모델을 추출한다. K-behavior의 경우, 컴포넌트 인터페이스를 대상으로 행위를 추출할 것을 권장하고 있으나, 기존 어플리케이션들이 대부분 컴포넌트 기반으로 구현되어 있지 않기 때문에 주요한 클래스를 대상으로 실행 로그를 추출하였다. 추출한 행위 모델은 그림 7과 같다.

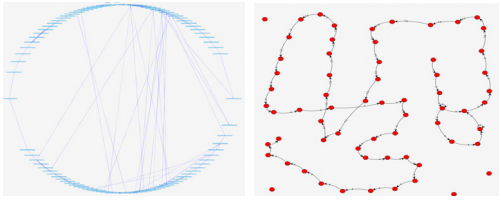

그림 7. 행위 모델(좌: CIBFD, 우: K-behavior)
Fig. 7. A behavioral Model (Left: CIBFD, Right:K-Behavior)

행위 모델 추출 후 알려진 오작동을 재현하여 오작동 탐지를 시도한 결과는 표 7과 같다.

표 7. 행위 모델 세부 정보 및 오작동 탐지 결과
Table 7. A detailed information of a behavioral model and a result of fault detection

행위 모델		오작동 탐지 결과		오작동 발생 이벤트	
K-beh	CIBFD	K-beh	CIBFD	K-beh	CIBFD
노드 65 엣지 63	노드 464 엣지 501	거짓 양성 41/100	거짓 양성 0/100	converter: writeBitmapInfoHeader()	converter:-writeBitmapFileHeader()

K-behavior가 상대적으로 거짓 양성의 발생 비율이 크다, 즉 오작동이 아님에도 오작동이라 판별하는 경우가 많다는 것을 확인할 수 있다. 추가 실험을 통해서 모니터링 대상을 세부 메서드 수준으로 늘릴 경우, 거짓 양성이 100% 가깝게 발생함을 확인할 수 있었다. 역으로, 추상화 수준이 높은 메서드들을 대상으로 실행 로그를 추출하여 행위 모델을 추출한 경우는 거짓 양성이 발생되지 않았다. 본 연구에서 제안하는 기법은 거짓 양성이 발생하지 않았다.

2. 행위 모델 상태 복잡도의 해결 가능 수준 분석

행위 모델의 복잡도는 노드와 엣지 개수를 기준으로 평가하고자 한다. Diff 분석에 의해 추출된 기능 특화 행위 모델의 복잡도가 본래의 행위 모델의 복잡도 대비 어느 정도 감소되는지, 그리고 그로 인해 불변 순차적 행위 패턴 단위의 개수가 얼마나 감소되는지 분석한다. 즉, 행위 모델의 복잡도 개선으로 인해 오작동 탐지 능력이 어느 정도 감소될 수 있는지 평가하고자 한다.

사례 연구들에 대해서 CIBFD를 적용한 결과, 행위 모델의 복잡도를 낮추는 비율에 비해서 오작동 탐지 능력의 감소 비율은 상대적으로 덜 낮아졌음을 확인할 수 있다. 표8과 표 9를 보면, 전반적으로 행위 모델의 복잡도의 감소 비율에 비해 오작동 탐지를 위한 순차적 행위 패턴의 감소 비율이 낮음을 알 수 있다.

표 8. 행위 모델 내 노드, 엣지 정보
Table 8. Information for nodes and edges in a behavioral model

탐지 단위		행위 모델			기능 특화 행위 모델		
Index	Desc	Node	Edge	Pattern	Node	Edge	Pattern
p0	start	378	385	377	375	379	371
p1	draw	42	58	37	38	44	29
p2	undo	16	19	14	10	8	8
p3	save	56	70	48	37	36	29
p4	print	20	24	19	16	14	11
p5	open	70	86	64	52	52	45

표 9. 기능 특화 행위 추출에 의한 행위 모델 복잡도 및 순차적 패턴 감소율
Table 9. A complexity of a behavioral model by extracting a specific functions and a decreasing rate of a sequential pattern

탐지 단위		행위 모델의 복잡도 감소율		순차적 패턴 감소율
Index	Desc	Node	Edge	Pattern
p0	start	-0.79%	-1.56%	-1.59%
p1	draw	-9.52%	-24.14%	-21.62%
p2	undo	-37.50%	-57.89%	-42.86%
p3	save	-33.93%	-48.57%	-39.58%
p4	print	-20.00%	-41.67%	-42.11%
p5	open	-25.71%	-39.53%	-29.69%

V. 결론 및 향후 연구

본 논문에서는 레거시 소프트웨어에 대해 메서드 수준에서의 오작동 탐지 기능을 구축하기 위한 기법으로 문맥 독립적 행위 모델 기반 오작동 탐지 (Context-Insensitive Behavior Model-based Failure Detection) 기법을 제안하였다. 제안하는 기법은 기존의 문맥 의존적 행위 모델 기반의 상태 추적 방식과 달리 메서드 수준에서의 오작동 탐지를 지원하기 위해 문맥 독립적 행위 모델 기반의 순서 검사 방식을 제안하였다. 결과적으로, 기존 탐지 방식에 비해서 거짓 양성 문제가 적게 발생하는 경향을 보이며, 모니터링 오버헤드도 낮출 수 있었다.

그러나 기존 연구 대비 거짓 음성의 발생 비율이 클 수 있다는 단점을 가진다. 거짓 양성, 거짓 음성을 모두 해결하기는 어려운 문제이다. 본 연구 기법은 거짓 음성이 발생 가능하더라도 레거시 소프트웨어를 대상으로 메서드 수준의 오작동 탐지를 구현하면서도 런타임 시의 거짓 양성을 최소화하는 데에 중점을 두고서 기법을 제안하였다.

추가적으로 본 기법에서 Diff 분석의 실효성을 높이기 위해서는 적용 대상 소프트웨어가 아래의 조건을 만족해야 한다. 이를 위해 리팩토링이 선행되는 것이 좋을 수 있다.

- 여러 기능을 단일 컨트롤 플로우(control flow) 형태로 구현한 소프트웨어는 적용하기 어렵다.
- 단일 메서드 안에 단일 기능이 구현되어야 한다. 여러 입력 유형에 대한 처리가 하나의 메서드 내에 포함되어 있을 경우, 메서드 복잡도가 증가할 뿐만 아니라, 기능 특화 행위를 정확하게 추출하기 어렵다.

향후 행위 모델의 상태 복잡도를 낮추면서도 거짓 음성의 증가 비율의 폭을 더 낮출 수 있는 연구가 필요하다. 기능 별 행위 간 Diff 분석을 통해 추출한 기능 특화 행위를 중심으로 오작동을 탐지하기 때문에 공통 행위에 대해서는 정확한 탐지가 어렵다. 공통 행위에서의 에러로 인해서 결국 차이 행위에서의 에러까지 발생시킬 수 있기 때문에 탐지는 가능하나, 모델 기반의 원인 진단을 어렵다. 이를 해결하기 위해 공통 행위에 대해서는 추상화 수준이 높은 상태들로 구성된, 문맥 의존적 행위 모델을 채용하는 것을 검토할 필요가 있다. 즉, 문맥 의존적 모델과 문맥 독립적 모델이 섞여 있는 모델을 중심으로 연구 진행이 필요하다.

보다 세밀한 모니터링 오버헤드 조절이 가능한 기법의 추가 연구가 필요하다. CIBFD에서 모니터링 대상을 선정하여 모니터링 오버헤드를 기능 간의 조절이 가능하다. 보다 세밀한 조절이 가능 하려면 각 기능 별로 실행 시간을 파악하고 그에 따라서 동적으로 모니터링 대상을 조절할 수 있도록 하는 기법이 필요하다. 실행 시간의 분석 정보를 기반으로 프로브를 삽입하는 것에 대한 연구는 기존 연구에서도 다룬 바 있다[13][15].

References

[1] Kindberg, T. and Fox, A., “System Software for Ubiquitous Computing,” IEEE Pervasive Computing, Vol. 1, No. 1, pp. 70-81, 2002.
[2] Kim, J. S., Park, S. Y. and Sugumaran, V.,

"Contextual Problem Detection and Management during Software Execution in Complex Environments," Industrial Management & Data Systems, Vol. 106, No. 4, pp. 540-561, 2006.

[3] DARPA, "Self Adaptive Software," BAA 98-12, Proposer Information Pamphlet, www.darpa.mil/ito/Solicitations/PIP_9812.html, 1997.

[4] IBM, "An Architectural Blueprint for Autonomic Computing, Third edition," Tech. Report, 2005.

[5] Horn, P., "Autonomic Computing: IBM's Perspective on the State of Information Technology," IBM Corporation, 2001.

[6] Wong, W. E. and Debroy, V., "Software Fault Localization," IEEE Reliability Society 2009 Annual Technology Report, January 2010.

[7] Kephart, J. O. and Chess, D. M., "The Vision of Autonomic Computing," IEEE Computer, Vol. 36, No. 11, pp. 41-52, 2003.

[8] Kim, M., Kannan, S., Lee, I., Sokolsky, O. and Viswanathan, M., "Java-MaC: a Run-time Assurance Tool for Java Programs," Electronic Notes in Theoretical Computer Science, Vol. 55, Issue 2, pp. 218-235, 2001.

[9] Hlady, M., Kovacevic, R., Li, J. J., Pekilis, B. R., Prairie, D., Savor, T., Seviora, R. E., Simser, D. and Vorobiev, A., "An Approach to Automatic Detection of Software Failures," Proceedings of 6th International Symposium on Software Reliability Engineering, pp. 24-27, 1995.

[10] Mariani, L., Pastore, F. and Pezze, M., "Dynamic Analysis for Diagnosing Integration Faults," IEEE Trans. On Software Engineering, Vol. 37, Issue 4, pp. 486-508, 2011.

[11] Chang, H., Mariani, L. and Pezze, M., "In-Field Healing of Integration Problems with COTS Components," Proceedings of ICSE 2009, pp. 166-176, 2009.

[12] Cook, J. E. and Wolf, A. L., "Discovering Models of Software Processes from Event-Based Data," ACM Trans. On Software Engineering and Methodology, Vol. 7, Issue 3, pp. 215-249, 1998.

[13] Fischmeister, S. and Lam, P., "Time-Aware Instrumentation of Embedded Software," IEEE Trans. on Industrial Informatics, Vol. 6, No. 4, pp. 652-663, 2010.

[14] Kanstren, T., "Towards A Taxonomy of Dynamic Invariants in Software Behavior," Proceeding of 2nd International Conference on Pervasive Patterns and Applications, pp. 20-27, 2010.

[15] Y. Yang, B. Chang, "A Probability Embedded Expert System to Detect and Resolve Network Faults Intelligently," International Journal of Internet, Broadcasting and Communication, Vol 11, No 2, pp135-143, 2011

저자 소개

김 순 태(정회원)

- 2014년 ~ 현재 : 전북대학교 소프트웨어공학과 조교수
- 2007년 2월 ~ 2010년 8월 : 서강대학교 컴퓨터공학과 (석사 및 박사)
- 2003년 2월 : 중앙대학교 컴퓨터공학과 (학사)
- 2002년 11월 ~ 2004년 8월 : 소프트웨어 크래프트 컨설팅 선임컨설턴트

※ 이 논문은 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대 정보 · 컴퓨팅개발사업의 지원을 받아 수행된 연구임(No.NRF-2014M3C4A7030503).