관점 지향 프로그래밍이 내장된 DEVS 형식론 기반 소프트웨어 정형 검증 방법론

최창범*, 김탁곤*

Software Formal Verification Methodology Based on DEVS Formalism Instrumented with Aspect Oriented Programming

Choi, ChangBeom, Kim, Tag Gon

Abstract

Softwares are getting more complex due to a variety of requirements that include desired functions and properties. Therefore, verifying and testing the software is a complicated problem. Moreover, if the software is already implemented, inserting and deleting tracing/logging code into the source code of a software may cause the code tangling and the code scattering problems. This paper proposes the Aspect DEVS Verification Framework. The Aspect DEVS Verification Framework utilize Aspect Oriented Programming features to handle the code tangling and the code scattering problems. By applying aspect oriented feature, user can find and fix the inconsistency between requirement and implementation of software without suffering the code tangling and the code scattering problems. The first step of the verification process is generating aspect to make a software act as a generator. The second step is developing a requirement specification using DEVS diagrams and implement it using DEVSIM++. The final step is checking the event traces from the software compared with the possible execution sequences from DEVS model.

Key Words : DEVS, Aspect Oriented Programming, Software Verification, VV/A

* 한국과학기술원 전자전산학과 시스템 모델링 시뮬레이션 연구실

1. 서론

유비쿼터스 컴퓨팅 사회에서는 일상생활의 모든 영역으로 소프트웨어가 보급되었으며, 소프트웨어의 도움을 받고 있다. 하지만 나날이 소프트웨어의 편리함에 의존하고 있는 현 상황 속에서 사용자들은 소프트웨어가 자체적으로 가지고 있는 문제점들 속에 노출되어 있다. 이러한 문제점은 좁은 의미로 소프트웨어가 가지고 있는 버그와 같이 소프트웨어 자체가 가지고 있는 문제로 정의될 수 있고, 넓은 의미로는 소프트웨어가 만족해야하는 요구 사항을 만족시키지 못하는 것으로 정의될 수 있다. 소프트웨어가 가지는 문제점들은 소프트웨어의 신뢰성을 떨어뜨리며 이런 문제점을 해결하기 이전에 문제점을 발견하는 것 자체가 어렵다. 소프트웨어의 문제점을 탐지하기 위하여 실제 소프트웨어 개발에서는 테스팅 기법을 사용하고 있으며 문제점을 발견하고 해결하는 과정인 디버깅 과정에 소프트웨어의 총 개발 비용 중 50% 이상이 소요되고 있다[1]. 따라서 날로 대형화되고 그 구조가 복잡해지고 있는 소프트웨어 발전 경향으로 미뤄볼 때 점차 총 개발 비용에서 테스팅 기법을 사용한 디버깅 과정이 소모하는 비중이 늘어날 것으로 예상된다.

하지만 모든 가능한 수행 동작을 검사할 수 없는 테스팅 기법만으로는 높은 수준의 신뢰성을 확보하는 것은 어렵다. 이러한 테스팅 방법을 보완하는 기법으로 수학적 정확성을 바탕으로 하는 정형 검증(Formal Verification) 기법이 제시되었고 소프트웨어의 신뢰성 확보에 사용되었다[2][3]. 이와 유사한 개념으로 어떤 대상을 모델링하여 검증하는 모델링/시뮬레이션(M&S) 공학이 있다. M&S 공학은 어떠한 요구사항에 따라 M&S의 목적을 정립하고 이로부터 모델링 요구 사항을 도출한다. 모델 검증 기법과 같이 M&S에서도 수학적 형식론으로 대상 시스템의 행위를 표현할 수 있는 수학적 의미론에 기반을 둔 모델과 모델링 틀을 제공 한다. 대표적인 모델링 틀로서는 이산사건 시스템의 모델링 틀인 DEVS(Discrete Event Systems Specification) 형식론이 있으며 이를 구현하여 시뮬레이션에 적용할 수 있는 개발 도구들이 개발되어 있다. 그 동안 M&S에서는 환경오염 시뮬레이션, 워 게임(War game) 시뮬레이션, 유체 역학 시뮬레이션 등 다양한 공학 분야에서 각각 다른 대상들에 대하여 모델링을 수행하고 시뮬레이션을 통하여 해당 모델들의 동작을 분석하는 연구가 진행되었지만, 소프트웨어를 대상으로 모델링 및 시뮬레이션에 대한 연구가 미진하였다.

본 논문에서는 관점 지향 프로그래밍 기술과 DEVS 형식론을 사용하여 소프트웨어 정형 검증 방법론을 제안한다. 기존에 테스팅만으로는 신뢰성 확보에 문제가 있으며 정형 검증 기법은 실제 소프트웨어 개발에 적용하기에는 무리가 있다. 따라서 본 연구는 실제 구현된 소프트웨어를 대상으로 한 소프트웨어 정형 검증 틀 개발을 병행 하여 연구용 이외에 소프트웨어 개발자들이 사용할 수 있는 소프트웨어 정형 검증 틀에 대한 연구를 수행한다. 본 논문의 구성은 다음과 같다. 2장에서는 관련 연구들과 Aspect DEVS 검증 틀을 구성하는 DEVS 형식론 및 관점 지향 프로그래밍에 대하여 소개한 다음 3장에서 제안하는 Aspect DEVS 검증 틀을 소개한다. 4장에서는 보안 취약성을 검증하는 사례 연구를 다루고 5장에서 결론을 맺는다.

2. 관련 연구

소프트웨어의 신뢰성을 높이기 위한 정형 검증 기법은 크게 정적인 방법과 동적인 방법으로 나뉜다.

정적 정형 검증 기법 중에서 대표적인 검증 기법인 모델 검증 기법은 모든 적용 범위에 대하여 문제점을 검사하고, 결과에 대해서 보장한다[4]. 정적 정형 검증 기법을 소프트웨어에 적용할 경우 모델 검증 기법은 실제 프로그램 소스 코드에 대해서 실행 될 때의 상태에 대해서 검증을 수행할 수 없으며 소프트웨어 소스 코드에서 모델을 추출하고, 그 모델에 대해서 검증하려고 하는 요구사항들에 대해서 검증을 수행한다. 따라서 모델에서 발견한 문제와 실제 소스 코드 상에서의 문제점들과는 다를 수 있다.

동적인 정형 검증 기법은 소프트웨어가 실행될 때 모니터링을 수행하여 해당 수행 경로가 사용자가 요구한 요구 사항을 만족시키는지 검사하는 방법이다[5]. 이 방법은 소프트웨어를 다운 받고 실행시키는 소프트웨어가 재구성되는 실행 환경에서 소프트웨어를 실시간으로 검증하기 위하여 사용된 방법으로 실제 소프트웨어의 코드를 기반으로 하며 테스팅과 달리 오류가 검출되기 전까지 수행 결과가 소프트웨어의 요구사항을 만족한다는 것을 보장한다. 하지만 현재 개발된 실시간 검증을 위한 도구는 JAVA의 바이트코드를 쉽게 수정할 수 있는 JAVA 언어의 기술적인 이점을 사용하기 때문에 실행 플랫폼에 제한되어 있으며 현재 C++ 기반의 실시간 검증 도구는 알려진 바가 없다. 또한 사용자의 요구사항을 기술하기 위해서 선형 시간 논리(Linear Temporal Logic)과 같은 난해한 요구사항 언어를 습득해야 하는 문제점이 있다. 따라서 본 논문은 요구사항을 기술하는 도메인 전문가가 쉽게 사용할 수 있는 DEVS 형식론과 소스 코드 수준에서 자동으로 코드를 직조할 수 있는 관점 지향 프로그래밍 기법을 사용하여 소프트웨어 검증 틀을 제시한다.

2.1. DEVS 형식론

DEVS 형식론은 이산 사건 시스템을 객체 단위로 모듈화 하여 계층적으로 결합하여 표현하는 집합론에 근거한 수학적인 틀이다. DEVS 형식론에는 시스템 기본적인 구성 요소를 나타내는 원자 모델과 여러 모델을 합쳐서 새로운 모델을 구성할 수 있는 결합 모델이 있다[5].

2.1.1 원자 모델

원자 모델(Atomic Model)은 DEVS 형식론을 구성하는 가장 기본적인 모듈로서 시스템의 행동을 기술하는 모델이다. 원자 모델 M의 수학적인 표현은 다음과 같다.

$M = \langle X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta \rangle$
X: 이산사건 입력 집합
Y: 이산사건 출력 집합
S: 일련의 이산사건 상태의 집합
$\delta_{ext} : Q \times X \rightarrow S$: 외부 상태 천이 함수
$Q=\{(s,e)|s\in S, 0\leq e\leq ta(s)\}$: total state of M
$\delta_{int} : Q \rightarrow Q$: 내부 상태 천이 함수
$\lambda : Q \rightarrow Y$: 외부 상태 천이 함수
$ta: S \rightarrow R_{0,\infty}^{+}$: 시간 진행 함수

2.1.2 결합모델

결합 모델(Coupled Model)은 여러 모델을 내부적으로 연결하여 만든 모델이다. 내부 구성요소가 되는 모델은 원자 모델과 결합 모델이 모두 가능한데. 이러한 내부 모델들을 계

속 합쳐서 더욱 큰 시스템을 표현할 수 있다. 다음은 결합 모델의 수학적 명세이다.

CM = <X, Y, {M_i}, EIC, EOC, IC, SELECT>
X : 이산사건 입력 집합
Y : 이산사건 출력 집합
{M_i} : 모든 이산사건 콤퍼넌트 모델들의 집합
EIC : 외부 입력 연결 관계
EOC : 외부 출력 연결 관계
IC : 내부 연결 관계
SELECT : $2^{(Mi)} - \emptyset \rightarrow M_i$: 같은 시각에 존재하는 사건을 발생하는 모델들에 대한 선택 함수

2.2. 관점 지향 프로그래밍

관점 지향 프로그래밍(Aspect Oriented Programming)이란 시스템 전반에 퍼져 있는 횡단 관심사를 독립적으로 모듈화 하여 나중에 합성(직조) 과정을 거쳐 한 개의 시스템으로 조립하는 방법론이다[7]. 관점 지향 프로그래밍이 적용 가능한 대표적인 분야는 감시 기법으로 로깅, 트레이싱과 프로파일 등이 있다. 관점 지향 프로그래밍 방법을 사용하여 소프트웨어의 복잡성과 시장 출시 시간을 줄일 수 있어 이와 같은 장점을 이용하여 관점 지향 프로그래밍을 적용한 여러 사례에서 프로그램의 디버깅 또는 성능 분석 등에 유용하게 사용할 수 있다는 사례 보고가 있다. 또한 관점 지향 프로그래밍을 지원하기 위하여 Java 개발환경에서는 AspectJ와 C++ 환경에서는 AspectC++가 제공되고 있다.

3. Aspect DEVS 검증 방법론

Aspect DEVS 검증 방법론은 일반 소프트웨어를 대상으로 하여 DEVS 형식론으로 사용자의 요구사항을 기술한 후 관점 지향 프로그래밍 기법을 사용하여 검증을 수행한다. Aspect DEVS 검증 방법론은 동적 정형 검증 기법 중 하나로, 실제 소프트웨어가 실행될 때 발생하는 이벤트들을 바탕으로 검증기를 사용하여 검증을 수행한다. 이러한 Aspect DEVS를 사용한 검증 방법론을 적용한 정형 검증 틀은 그림 1과 같다.

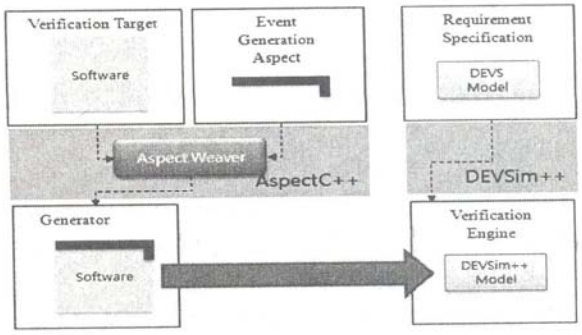

그림 1 Aspect DEVS 정형 검증 틀

Aspect DEVS 검증 틀은 DEVS 형식론으로 사용자 요구 사항을 기술하고, 소프트웨어를 이벤트 생성기를 만드는 과정과 DEVS 형식론으로 구현된 사용자 요구사항 모델과 이벤트 생성기에서 생성된 이벤트를 사용하여 검증을 수행하는 검증 과정으로 구성된다. 이러한 Aspect DEVS 검증 틀은 Generator와 Verification Enginedm로 구성된다.
Generator는 소프트웨어에서 이벤트를 생성하도록 소스 프로그램에 이벤트 생성기를 삽입한다. 이러한 과정은 관점 지향 프로그래밍 기법의 Aspect Weaver를 사용하여 자동으로 수행되며, 추후 검증된 소프트웨어에 대하여 삽입한 이벤트 생성기를 제거함으로써 사용자의 요구사항을 만족함을 보일 수 있다. 이벤트를 생성할 수 있도록 한다.
Verification Engine은 입력되는 이벤트들을

DEVS 모델로 전달하여 실제 검증을 수행한다. Verification Engine은 사용자가 DEVSim++와 같은 도구로 구현한 DEVS 모델으로 이루어져 있다. 이 구현된 DEVS 모델에 Generator로부터 입력받은 이벤트들을 입력하고 도출된 출력을 사용하여 해당 소프트웨어가 사용자의 요구사항을 만족하는 지 검증한다.

4. Case Study: exec() 취약점 검증

Hao Chen 등은 리눅스와 솔라리스, 유닉스의 setuid()함수에 대한 보안의 취약성에 대하여 연구하였다[8].

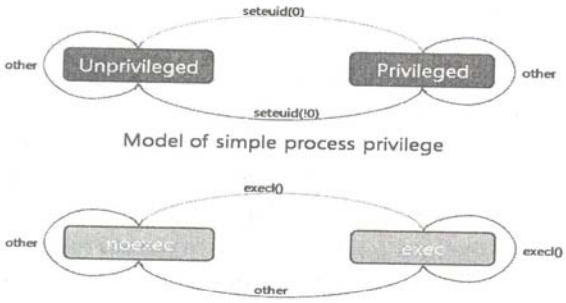

그림 2 프로세스 권한 모델과 시스템 콜 모델

그림 2는 Hao Chen 이 발견한 프로세스의 권한 모델과 위험한 시스템 콜의 모델이다. 리눅스 시스템에서는 관리자 권한을 획득하는 방법은 seteuid()함수를 호출하여 권한을 획득한다. 따라서 어떤 사용자의 권한을 가진 프로세스가 자신의 권한보다 더 높은 권한의 작업을 수행할 때, 더 높은 권한을 가져도 되는지 검사해야 하며, 자신이 가지고 있는 권한을 포기해야 한다. 다음 그림은 프로세스 권한 모델과 execl() 시스템 콜 모델을 사용하여 작성한 사용자 요구사항 DEVS 원자 모델이다.

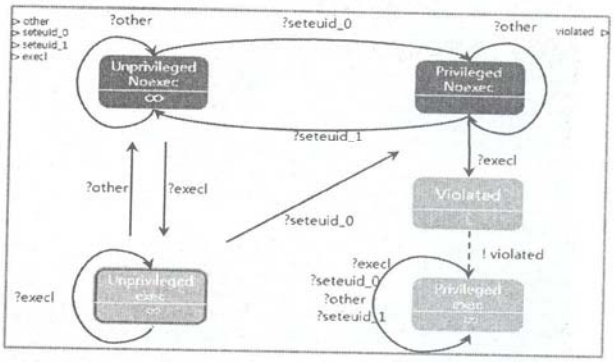

그림 3 보안 취약성 검증 모델

보안 취약성을 검증하기 위한 검증 모델은 other, seteuid_0, seteuid_1, execl 포트를 가지고 발견되면 violated port로 출력 이벤트를 발생시킨다.

```
#include "stdafx.h"
#include <stdio.h>
#include "SystemCall.h"

void drop_privilege()
{
        passwd *password;
        if((password = getpwuid(getuid())) == NULL)
                return; // but forget to drop privilege!
        printf("drop priv for %s\n", password->pw_name);
        seteuid(getuid());
}

int _tmain(int argc, _TCHAR* argv[])
{
        //start with root privilege
        do_something_with_privilege();
        drop_privilege();
        execl("/bin/sh", "bin/sh", NULL); // risky system call
        return 0;
}
```

그림 4 보안 취약성을 가진 프로그램

그림 4는 보안 취약성을 가진 프로그램으로써 C++로 구현되어 있으며 그림 5는 관점 지향 프로그래밍으로 직조된 그림 4의 프로그램이 생성한 이벤트에 대해서 DEVSim++[9]로 구현된 검증기와 검증 모델을 사용하여 검증한 결과이다.

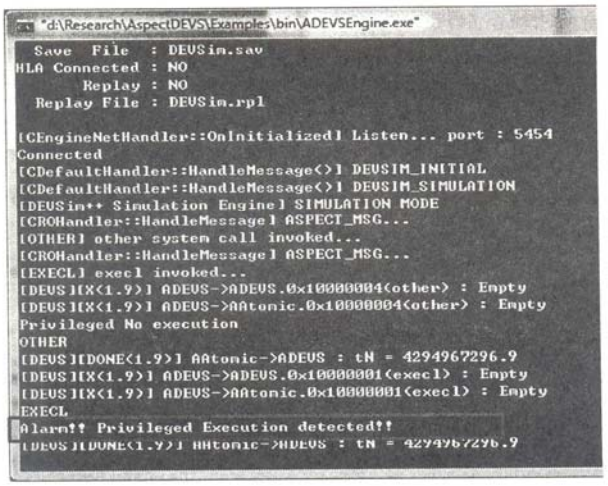

그림 5 검증 결과

5. 결론

본 논문은 M&S와 관점 지향 프로그램 기법을 사용하여 소프트웨어를 검증하기 위한 정형 검증 방법론과 검증을 위한 정형 검증 틀을 제시하였다. 본 논문에서 제안하는 검증 방법론은 기존에 소프트웨어 공학에서 소프트웨어를 모델링하고 이를 검증하였던 연구를 M&S 분야에서 연구함으로써 M&S 분야의 적용 가능 분야를 확장시킬 수 있다.

또한, 시뮬레이션 틀에서 모니터링하기 위해 틀자체에 모니터링 코드를 삽입해야 했지만, 본 연구를 통하여 이를 자동화하고 모니터링 서버를 둠으로써 기존 시뮬레이션 틀의 유지 보수 비용을 줄일 수 있고, 국방 M&S와 같이 VV/A (Verification, Validation/Analysis)가 매우 중요한 분야에서 본 연구 결과를 적용하여 생성되는 결과를 VV/A를 위한 근거로 활용할 수 있을 것으로 기대된다.

참고문헌

[1] Software Engineering: A Practitioner's Approach 6th ed. McGraw.

[2] Virtual Library of Formal Methods. http://vl.fmnet.info.

[3] E. Clarke and J. Wing, et al, "Formal Methods: State of the Art and Future Directions," In ACM Computing Surveys, Volume 28, Number 4, 1999.

[4] A. Cimatti, E. Clarke, F. Giunchiglia and M. Roveri. "NuSMV: a new symbolic model verifier," In Computer-Aided Verification, 1999

[5] M. Kim, S. Kannan, I. Lee, O. Sokolsky, and M. Viswanathan, "Java-MaC: a Rigorous Run-time Assurance Tool for Java Programs," Formal Methods in System Design, 2004 (vol 24 no 2)

[6] Bernard P. Zeigler, Herbert Praehofer and Tag Gon Kim, "*Theory of Modeling and Simulation,*" ACADEMIC PRESS, 2001.

[7] R. Laddad, "*AspectJ in Action: Practical Aspect-Oriented Programming,*" Manning, 2003.

[8] Hao Chen, Drew Dean, and David Wagner, "Model Checking One Million Lines of C Code," In Network and Distributed System Security (NDSS 2004), 2004.

[9] T. G. Kim, DEVSimHLA User's Manual, 2007. Available: http://smslab.kaist.ac.kr

[10] J.H. Byun, C.B. Choi, and T.G. Kim, "Verification of the DEVS Model Implementation using Aspect Embedded DEVS," SCS Spring Simulation Multiconference, 2009