Aspect DEVS 검증 틀을 이용한 소프트웨어 정형 검증 방법론

최창범[1]† · 김탁곤[1]

Software Formal Verification Methodology using Aspect DEVS Verification Framework

Chang Beom Choi· Kim, Tag Gon

ABSTRACT

Software is 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 may cause several problems, such as the code tangling and the code scattering problems. This paper proposes the Aspect DEVS Verification Framework which supports the verification and testing process. The Aspect DEVS Verification Framework utilizes Aspect Oriented Programming features to handle the code tangling and the code scattering problems. By applying aspect oriented feature, a user can find and fix the inconsistency between requirement and implementation of a software without suffering the problems. The first step of the verification process is building aspect code to make a software act as a generator. The second step is developing a requirement specification using DEVS diagrams and implementing it using DEVSIM++. The final step is comparing the event traces from the software with the possible execution sequences from DEVS model.

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

요 약

사용자가 요구하는 기능과 특성에 대한 다양한 요구사항은 소프트웨어를 점점 더 복잡하게 만들고 이를 검증하고 검사하는 것은 어려운 문제이다. 게다가 개발된 소프트웨어 코드를 검증하고 검사를 수행하는 과정에서 트레이싱 혹은 로깅 코드를 삽입하는 과정과 검사가 완료되어 삭제하는 과정에서 사용자의 부주의로 버그가 삽일될 수 있는 문제점도 발생한다. 본 논문은 소프트웨어 개발 과정 중에서 검증 및 검사 과정을 지원할 수 있는 Aspect DEVS 검증 틀을 제시한다. Aspect DEVS 검증 틀은 관점 지향 프로그래밍 기법을 사용하여 이미 구현되어 있는 소프트웨어와 사용자는 요구사항을 만족하는 지를 확인하는 동시에 소스 코드에 테스팅 코드를 삽입할 때 발생하는 문제들을 해결한다. Aspect DEVS 검증틀을 사용한 검증의 첫번째 단계는 관찰 대상을 관점 지향 프로그래밍 기법을 사용하여 명세한 하고, 사용자의 요구사항을 DEVS 다이어그램을 명세한 후 이를 DEVSIM++로 구현한다. 마지막으로 프로그램의 수행 과정 중에서 발생하는 이벤트들을 대상으로 구현한 DEVS 모델의 이벤트 입력으로 넣어 소프트웨어가 사용자의 요구사항을 만족하는 지를 검사함으로써 검증 과정 중에 대상 소프트웨어의 수정 없이 검증 및 검사를 수행할 수 있다.

주요어 : DEVS, 관점 지향 프로그램, 소프트웨어 검증, VV/A

---

[1] 한국과학기술원 전기 및 전자 공학과
주 저 자 : 최창범
교신저자 : 최창범
E-mail; cbchoi@smslab.kaist.ac.kr

1. 서론

현대의 정보화 사회에서는 일상생활의 모든 영역으로 소프트웨어가 보급되었으며, 소프트웨어의 도움을 받고 있다. 하지만 사용자들은 소프트웨어의 편리함에 의존하고 있는 현 상황 속에서 소프트웨어가 자체적으로 가지고 있는 문제점들 속에 노출되어 있다. 이러한 문제점들은 소프트웨어가 가지고 있는 결함(버그)과 같이 소프트웨어 자체가 가지고 있는 문제와 소프트웨어가 만족해야하는 요구 사항을 만족시키지 못하는 것으로 정의될 수 있다. 이러한 문제점들은 소프트웨어의 신뢰성을 떨어뜨리며 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 총 개발 비용에서 테스팅 기법을 사용한 디버깅 과정이 소모하는 비중이 늘어날 것으로 예상된다. 게다가 모든 가능한 수행 동작을 검사할 수 없는 테스팅 기법만으로는 높은 수준의 신뢰성을 확보하는 것은 어렵다. 따라서 소프트웨어가 가지는 문제점들에 대한 신뢰성 확보를 위하여 수학적 정확성을 바탕으로 하는 정형 검증(Formal Verification) 기법이 제시되었다[2][3].

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 수행하기 때문에 모델에서 발견한 문제와 실제 소스 코드 상에서의 문제들과는 다를 수 있다.

동적인 정형 검증 기법은 소프트웨어가 실행될 때 모니터링을 수행하여 해당 수행 경로가 사용자가 요구한 요구 사항을 만족시키는 지 검사하는 방법이다[5]. 실제 소프트웨어의 코드를 기반으로 하며 테스팅과 달리 오류가 검출되기 전까지 수행 결과가 소프트웨어의 요구사항을 만족한다는 것을 보장한다. 하지만 사용자의 요구사항을 기술하기 위해서 선형 시간 논리(Linear Temporal Logic)와 같은 난해한 요구사항 언어를 습득해야 하므로 검증 대상 소프트웨어의 행동을 기술하기 난해한 문제점이 있다.

검증하려는 대상을 수학적으로 모델링하고 이를 검증하는 정형 검증 기법과 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 있고 시간 명세를 할 수 있는 수학적 의미론에 기반을 둔 모델과 모델링 틀을 제공한다. 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&S는 검증 대상의 행동을 수학적으로 기술하는데 용이하여 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 모델링하고 이를 모의하여 그 결과를 분석하는 연구는 미진하였다.

본 논문에서는 관점 지향 프로그래밍 기술과 DEVS 형식론을 사용하여 M&S를 기반으로 하여 소프트웨어 정형 검증 방법론을 제안한다. 기존의 테스팅 만으로는 신뢰성 확보에 문제가 있으며 정형 검증 기법은 실제 소프트웨어 개발에 적용하기에는 무리가 있다. 따라서 본 논문에서는 사용자의 요구사항을 쉽게 기술할 수 있는 DEVS 형식론으로 사용자의 요구사항을 기술하고, 실제 구현된 소프트웨어를 대상으로 소프트웨어 정형 검증을 수행할 수 있도록 관점 지향 프로그래밍 기법을 사용한 정형 검증 틀인 Aspect DEVS 검증 틀을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 Aspect DEVS 검증 틀에서 사용자의 요구사항을 수학적으로 기술할 수 있는 DEVS 형식론을 소개한다. 3장에서는 검증 대상 소프트웨어의 동작을 방해하지 않으면서 검증을 가능하게 하는 관점 지향 프로그래밍기법에 대하여 소개한다. 4장에서 제안하는 Aspect DEVS 검증 틀을 소개한다. 5장에서는 보안 취약성을 검증하는 사례 연구를 다루고, 마지막으로 6장에서 결론을 맺는다.

2. 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 원자 모델 표현

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

AM = <X, Y, S, δ_{ext}, δ_{int}, λ, ta>
X: 이산사건 입력 집합
Y: 이산사건 출력 집합
S: 일련의 이산사건 상태의 집합
δ_{ext} : Q × X → S: 외부 상태 천이 함수
Q={(s,e)|s∈S,0≤e≤ta(s)}: total state of M
δ_{int} : Q → Q: 내부 상태 천이 함수
λ : Q → Y: 출력 함수
ta: S → $R_{0,\infty}^{+}$: 시간 진행 함수

2.2 결합 모델

결합 모델(Coupled Model)은 여러 모델을 내부적으로 연결하여 만든 모델이다. 내부 구성요소가 되는 모델은 원자 모델과 결합 모델이 모두 가능한데. 이러한 내부 모델들을 계속 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 : $2^{\{M_i\}}$ - Ø → M_i: 같은 시각에 존재하는 사건을 발생하는 모델들에 대한 선택 함수

2.3. DEVS 추상화 시뮬레이터

DEVS 형식론에서 모델 자체는 수동적인 개체이다. 모델은 미리 정해진 집합과 함수들로 이루어져 있을 뿐, 실제로 정해져 있는 함수를 호출하는 것은 모델에 연결되어 있는 시뮬레이션 프로세스이다. 원자 모델에는 시뮬레이터(*Simulator*)라고 하는 시뮬레이션 프로세스가, 결합 모델에는 코디네이터(*Coordinator*)라는 시뮬레이션 프로세스가 위치한다[6].

표 1. DEVS 시뮬레이션 메시지

메시지	설 명
(*, t)	내부적으로 t 시간에 생성된 메시지로 스케쥴된 시간에 도달했음을 알린다.
(x, t)	t 시간의 외부 입력 이벤트
(y, t)	t 시간의 내부 출력 이벤트
(done, t_N)	t_N 시간에 생성된 시간 동기화 메시지로 다음 스케쥴될 시간이 t_N임을 나타냄

표 1은 DEVS 시뮬레이션 메시지를 나타내고, 그림 2는 시뮬레이터의 알고리즘을 나타낸다. 시뮬레이터 알고리즘은 원자 모델의 동작 알고리즘으로 외부에서 입력을 받았을 경우 다음 스케쥴 시간을 계산하여 반환하고 스케쥴된 시간이 되었다는 이벤트가 발생했을 때 출력 이벤트를 발생시키고 내부 천이를 수행한 후 다음 스케쥴 시간을 계산하여 반환한다. 이를 통하여 원자 모델은 이벤트에 대한 처리와 시간에 따른 이벤트의 처리를 수행한다.

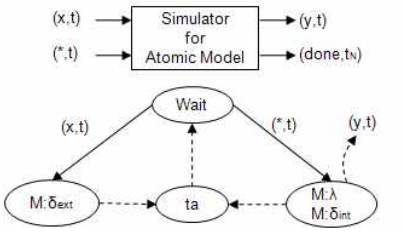

그림 2. 시뮬레이터 알고리즘[7]

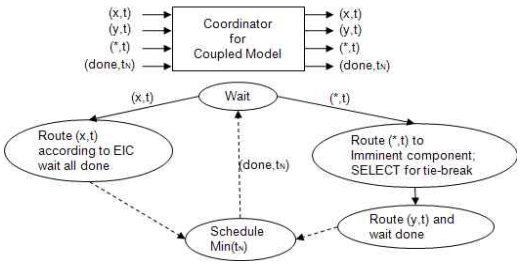

그림 3. 코디네이터 알고리즘[7]

그림 3은 코디네이터 알고리즘을 나타낸다. 코디네이

터는 결합 모델의 알고리즘으로 결합 모델 내의 원자 모델들의 동작을 담당한다. 코디네이터 알고리즘은 결합 모델 내의 원자 모델 간의 우선 순위를 조절하고 스케쥴된 시간의 진행을 담당한다.

2.4. DEVSim++

DEVSim++는 DEVS 형식론을 시스템 모델링 도구로 사용하여 DEVS 형식론의 추상화된 시뮬레이터를 시뮬레이션 엔진으로 사용한 이산사건 시스템 모델링 및 C++를 기반으로 한 DEVS 시뮬레이션 환경이다[8]. 따라서 DEVSim++는 객체 지향 프로그래밍이 가지는 캡슐화, 상속, 재사용과 같은 장점을 가진다. DEVSim++는 시스템의 구현된 원자 모델의 이벤트를 관리하며, 이를 위하여 원자 모델을 위한 클래스 및 API를 제공한다.

3. 관점 지향 프로그래밍

현재 대부분의 소프트웨어 개발 프로젝트에서는 객체 지향 프로그래밍을 사용하여 프로젝트를 진행하고 있다. 객체지향 프로그래밍은 모듈단위로 공통된 동작을 모델링하는 강점이 있으나 횡단관심사라 칭하는 많은 모듈에 걸쳐있는 동작을 모듈화하지 못한다.

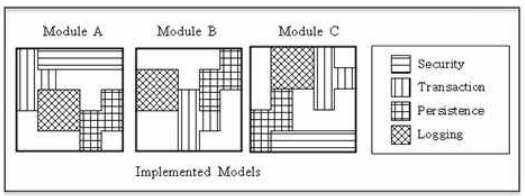

그림 4 . 여러 모듈에서의 횡단관심사의 예

위 그림 4는 어떤 모델을 구현했을 때의 내부 모듈을 설명하는 그림이다. 위 그림에서와 같이 사용자의 요구사항에 따른 핵심 관심사에 따라 모듈 A, B, C로 잘 모듈화 되어 만들어진 프로그램이라도 시스템 전체에 걸쳐 나타나는 부가적인 요구사항들, 예를 들면 사용자인증, 로깅, 자원공유, 메모리관리, 보안, 트랜잭션, 무결성 등은 각 모듈에 산재되어 나타나게 된다. 이와 같이 각 모듈에 산재되어 나타는 부가적인 요구사항들을 만족시키기 위해서 기존 객체 지향 방법에서는 해당 위치에 부가적인 요구사항을 만족시키는 코드를 삽입하여 처리하고, 이후 부가적인 요구사항의 변경으로 인해 삽입한 코드의 삭제가 빈번하게 일어나는 과정 중에서 쉽게 결함이 삽입되는 문제점이 발생하였다. 이와 같은 필요성으로 시스템 전체에 걸쳐 나타나는 부가적인 요구사항들을 만족시키는 코드의 모듈화 및 관리할 수 있는 새로운 방법론으로 관점 지향 프로그래밍(Aspect-oriented Programming :AOP)이 제시되었다[9]. AOP는 횡단관심사를 독립적으로 모듈화하여 나중에 합성(직조)과정을 거쳐 한 개의 시스템을 조립하는 새로운 방법론으로 이를 잘 활용하면 소프트웨어의 복잡성과 시장 출시시간을 줄이는데 크게 기여할 수 있고, 작은 투자와 작은 위험부담으로 큰 혜택을 얻을 수 있다.

3.1. 관점 지향 프로그래밍 용어

AOP는 기존 모듈에 대한 규격화 개념을 지원하는 동시에 다양한 횡단 관심사에 대해 규격화 기법을 지원하기 위해 결합점 모델(Joint Point Model)에 기반한 규격화 기법을 제공한다. 즉, 횡단 관심사를 정의하는 독립된 구조와 *aspect* 모듈을 사용하여 각 *aspect* 모듈은 이미 정의되어 있는 기능성에 대하여 소프트웨어의 모듈 구조에서 어느 호출 위치와 어떤 상황에 *aspect* 모듈을 실행하여 최종적으로 원하는 행위를 갖는 프로그램으로 변형시킬 것인가에 대하여 기술한다.

표 2 AOP의 결합점 모델

결합점 종류	설명
method call, constructor call	함수나 생성자의 호출 전
method call reception, constructor reception	함수나 생성자의 호출 후
method execution, constructor execution	함수나 생성자 수행 중
field get	속성 값이 읽혔을 경우
field set	속성 값이 쓰였을 경우
exception handler	예외처리가 수행되는 경우
class initialization	정적 클래스의 초기화
object initialization	객체의 동적 초기화

표 2는 AOP가 가지는 결합점 모델에 대한 설명이다. *aspect* 모듈은 기본 결합점을 표현할 수 있는 기본 *pointcut* 지정자들로 구성된다. 이러한 *pointcut*은 소프트웨어의 함수들이 호출될 때 어느 위치와 어느 시점에서 *aspect* 모듈이 실행될 것인지를 기술한다. *Pointcut*은 함수가 호출될 때, 다른 함수를 호출할 때 객체의 속성을 읽거나 변경할 때, 예외 상황 발생시, 그리고 객체나 클래스

가 정적 변수 형태로 초기화되는 시점에서 실매개 변수 형태로 전달되는 값에 기반하여 프로그램의 행위를 변경시킨다. 즉, AOP에서는 *pointcut*으로 필요한 결합점의 집합을 정의하고, 각 결합점 집합에서 공통으로 적용될 수 있는 프로그램 변형 코드인 *advice*를 정의한다.

3.1. Aspect C++

AOP는 제록스 연구원들에 의하여 Java 언어에 관점지향 개념을 추가하여 확장한 범용의 언어인 AspectJ[9]에 처음 도입되었다. AspectJ는 현재 이클립스의 서브프로젝트로 발전하고 있다. 그러나 아직 많은 응용 프로그램들, 특히 임베디드 시스템에 적용되는 대부분의 프로그램들은 C/C++로 개발되고 있다. 앞서 살펴본 관점지향 프로그래밍의 여러 장점들을 C/C++ 언어를 사용한 프로그램에서도 적용하기 위하여 동일하게 C++에도 관점지향 개념을 확장한 언어가 등장하게 되었는데 이것이 AspectC++이다[그림 5].

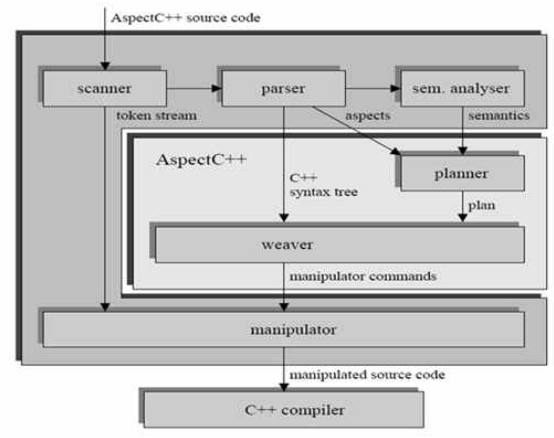

그림 5 AspectC++의 동작

AspectC++는 AspectJ를 기반으로 개발되어 AspectJ의 문법과 AOP 개념을 따르고 있다. 또한 사용자는 그림 5와 같이 AspectC++를 사용하여 횡단관심사를 *aspect*로 모듈화해서 구현하고, AspectC++는 이를 표준 C++로 변환하여 원 소프트웨어 코드에 삽입하여 이를 기존 C++ 컴파일러가 컴파일하여 수행할 수 있게 한다. 이를 통하여 AspectC++로 개발된 *aspect*은 AOP를 적용한 소프트웨어에서 AOP를 삽입하는 경우에 발생할 수 있는 실행 시간 부하를 전역적인 관점에서 최적화함으로써 최소화할 수 있다.

4. Aspect DEVS 검증 방법론

Aspect DEVS 검증 방법론은 일반 소프트웨어를 대상으로 하여 DEVS 형식론으로 사용자의 요구사항을 기술한 후 AOP 기법을 사용하여 검증을 수행하는 방법론이다. Aspect DEVS 검증 방법론은 동적 정형 검증 기법 중 하나로, 실제 소프트웨어가 실행될 때 발생하는 이벤트들을 바탕으로 검증기를 사용하여 검증을 수행한다. DEVS 형식론으로 기술한 사용자의 요구사항은 소프트웨어의 동작을 정형적으로 모델링함으로써 소프트웨어의 수행 시 발생하는 이벤트들을 검증한다. 또한, Aspect DEVS 검증 방법론은 AOP 기법을 사용하여 소프트웨어의 수행을 모니터링하기 위한 모니터링 코드의 삽입/삭제를 자동화함으로써 삽입/삭제 과정 중에서 사용자의 부주의로 발생할 수 있는 문제점을 해결한다. 이러한 Aspect DEVS를 사용한 검증 방법론을 적용한 검증 틀은 그림 6과 같다.

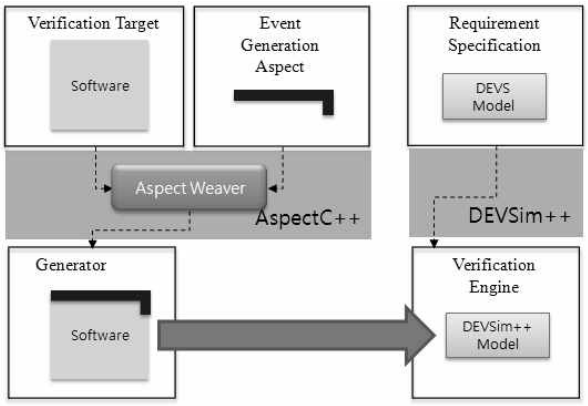

그림 6 Aspect DEVS 검증 틀

Aspect DEVS 검증 틀은 Generator와 Verification Engine으로 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 검증 과정에서 사용자의 부주의로 발생할 수 있는 문제점을 해결하며 검증 대상 소프트웨어가 사용자의 요구사항을 만족함을 보일 수 있다. 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.

Aspect DEVS 검증 틀은 DEVS 형식론으로 사용자 요구 사항을 기술하고, 소프트웨어를 이벤트 생성기로 만드는 과정과 DEVS 형식론으로 구현된 사용자 요구사항 모델과 이벤트 생성기에서 생성된 이벤트를 사용하여 검증을 수행하는 검증 과정으로 구성된다.

4.1. 요구사항 명세

소프트웨어가 만족해야 하는 사용자의 요구사항은 DEVS 형식론의 원자 모델을 사용하여 명세 한다. 사용자의 요구사항은 시스템의 동작으로 명세할 수 있으며 이 요구사항들은 시간 명세와 관련된 요구사항과 시간 명세와 관련 없는 요구사항들로 나눌 수 있다.

4.1.1. 시간을 고려하지 않은 행위 명세

사용자의 요구사항은 소프트웨어에서 교착상태와 같이 특정 이벤트가 일어나면 안 되는 Safety 특성은 대개 현재 진행된 시간에 상관없이 시스템의 동작을 모델링하는데 사용될 수 있다. 이러한 특성은 DEVS 형식론의 원자 모델로 쉽게 표현할 수 있다.

Safety 특성은 그림 7과 같이 소프트웨어에서 발생하는 이벤트들에 대해서 Safety 특성을 위반하는 이벤트 발생 시에 시간 전진이 무한대인 상태로 천이하는 상태로 표현될 수 있다. 즉, Safety 특성은 특정 이벤트가 발생하지 않는다는 것을 요구사항으로 명세하기 때문에 Safety 특성을 만족하고 있는 상태에 머물고 있다가 Safety 특성을 위반하게 만드는 이벤트가 발생하면 상태를 천이하게 하도록 DEVS 원자 모델을 작성할 수 있다.

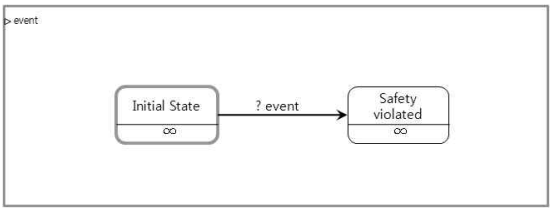

그림 7 Safety Property DEVS 모델

4.1.1. 시간을 고려한 행위 명세

사용자 요구사항 명세에서는 Safety 특성과는 다르게 특정 이벤트가 언젠가는 일어나야 한다는 Liveness 특성이 있다. 선형 시간 논리(Linear Temporal Logic)에서 Liveness 특성은 시스템의 동작에 있어서 무한한 시간 내에 정해진 이벤트가 발생하면 되는 특성을 지닌다. 하지만 실제 응용에서는 정해진 시간 내에 이벤트가 일어나야 한다는 것을 요구사항으로 명세해야 한다.

DEVS 형식론에서는 시간 전진함수가 정의되어 특정 시간을 초과하면 상태가 천이되도록 사용하여 사용자가 요구하는 Liveness를 정확하게 표현할 수 있다. 즉, Liveness 특성은 Liveness 특성을 만족시키는 이벤트가 발생했을 때 천이하는 상태로 표현될 수 있다. 그림 8은 특정 시간(ta) 내에 해당 이벤트가 발생해야 한다는 것을 표현한 원자 모델이다.

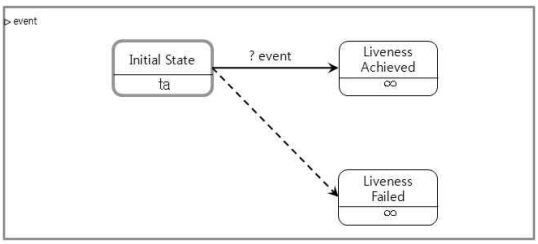

그림 8 Liveness Property DEVS 모델

위와 같은 Safety 특성과 Liveness 특성 DEVS 모델을 조합하여 사용자의 요구사항을 기술할 수 있다.

4.2. Aspect 명세

Aspect DEVS 검증 틀에서 검증을 수행하기 위해서는 Aspect을 명세해야한다. Aspect 명세는 크게 결합점을 명시하는 것과 모니터링해야 하는 이벤트를 기술하는 것으로 나뉜다.

Aspect DEVS 검증 틀에서의 발생 이벤트는 특정 함수의 호출 및 특정 함수가 호출될 때의 인자 값과 반환 값을 기반으로 생성된다. 따라서 Aspect DEVS 검증틀에서는 모니터링해야 하는 함수의 signature를 기반으로 결합점이 명세된다. 또한 모니터링해야 하는 함수 호출 혹은 반환 시에 호출 될 때의 동작을 명세하는 *advice*를 명세해야 한다. 다음 그림은 Aspect DEVS 검증 틀에서의 결합점 명세와 *advice*의 예이다.

```
00: aspect Generator  {
01: private:
02:    CADEVSNetHandler            m_NetHandler;
03:    CProtocolSuite              m_Protocol;
04:    CSelectNetwork m_Network;
05:
06: public:
07:    Generator():m_Network(&m_Protocol,&m_NetHandler){
08:        SMSL::CNetwork::instance( &m_Network );
09:        SMSL::CNetwork::RunThreadMode();
10:
11:        printf("connecting₩n");
12:        SMSL::CIPAddr addr("127.0.0.1", 5454);
13:        m_Network.Connect(addr);
14:    }
15:
16: advice execution("% foo(...)") :
17: before ()
18: {// Send Event Message to Server
19:    m_NetHandler.Send( EVENT0, NULL );
21: }
```

그림 9 Aspect Specification의 예

그림 9의 line02~line04는 검증을 수행하는 대상 소프트웨어에서 Aspect DEVS 검증 틀의 Verification Engine에게 이벤트를 전달하기 위해 필요한 변수를 선언한다. line07~line13에서는 Generator의 초기화 루틴이다. 마지막으로 line16~line21은 검증 대상 소프트웨어에 결합될 결합점과 수행해야 하는 동작이 기술되어 있다. line16에서의 advice execution(...)은 foo라는 함수의 수행시에 동작한다는 것을 나타내며 before 지시자는 foo라는 함수의 수행 전에 동작을 수행하는 것을 나타낸다. line19에서는 Verification Engine으로 이벤트를 보내는 것을 나타낸다.

4.3. 검증 과정

Aspect DEVS 검증 틀의 검증 과정은 다음과 같다. 사용자는 검증 대상 소프트웨어에 대해서 검증을 하고자 하는 요구사항을 작성한다. 즉, 사용자는 검증 대상 소프트웨어의 동작을 기술하고, 모니터링해야 하는 위치를 요구사항 단계에서 기술해야 한다. 이 후 사용자가 기술한 요구사항에 대해서 소프트웨어의 동작을 정상적인 동작과 비정상적인 동작으로 구분한 후 이를 DEVS 형식론을 사용하여 기술한다. 이 후 이를 DEVSim++와 같은 도구를 사용하여 DEVS 모델을 구현한다. 이후 해당 DEVS 모델을 Verification Engine에 추가한다. 이와 마찬가지로 사용자가 기술한 요구사항에서 모니터링해야 하는 위치를 관점 지향 프로그래밍에서의 aspect로 기술하여 Generator를 생성한다. Generator의 생성 후에는 AspectC++ 컴파일러를 사용하여 검증 대상 프로그램에 직조시킨다. 이후 검증 대상 프로그램에 맞는 컴파일러로 컴파일한 후 대상 프로그램을 수행하여 발생한 이벤트를 Verification Engine에서 검증하는 과정으로 검증이 수행된다.

5. 사례 연구

본 논문에서는 사례 연구로 Aspect DEVS 검증 틀을 사용하여 소프트웨어의 보안 취약성을 검증한다. 소프트웨어의 보안 취약성은 시간 명세 검증이 중요한 취약성과 소프트웨어의 동작 명세 검증이 중요한 취약성으로 나뉜다. 본 논문에서는 시간 명세 검증의 사례로 패스워드 공격과 동작 명세 검증 사례로 exec 취약성을 연구한다.

5.1 패스워드 공격

패스워드 공격이란 시스템에 입력하는 합법적인 사용자의 패스워드 키를 입수하거나 복호화하려는 시도를 말한다. 기존의 사용 가능한 패스워드 사전, 크래킹 프로그램을 사용하여 짧은 시간에 사용자의 패스워드 키를 입수하려는 시도를 한다. 이러한 패스워드 공격을 사용한 바이러스도 현재 발견되어 패스워드 공격을 조기에 탐지하여 공격에 대한 조치를 수행하는 필요성이 대두되고 있다 [10].

패스워드 공격은 기존의 사용 가능한 패스워드 사전, 크래킹 프로그램 등을 사용하여 짧은 시간 동안 다수의 로그인 시도를 수행하게 된다. 따라서 임의의 시간 동안 몇 번의 로그인 시도를 하게 되었는 지를 확인하여 패스워드 공격을 탐지할 수 있다. 다음은 패스워드 공격을 탐지하기 위한 DEVS 원자 모델의 예이다.

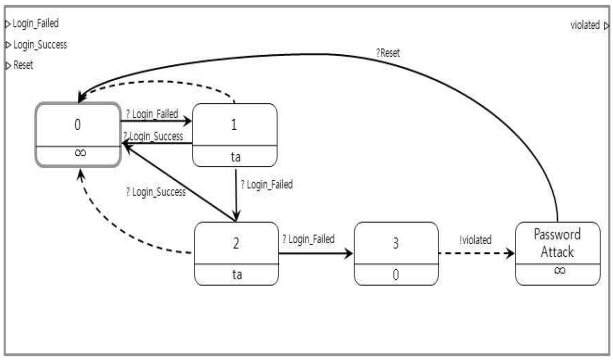

그림 10 패스워드 공격 탐지 DEVS 모델의 예

그림 10에서 상태는 로그인 실패 이벤트의 횟수를 나타낸다. 각 상태에서의 시간 전진함수는 DEVS 형식론의 Continue 개념을 사용하여 새로운 시간 전진함수를 사용하지 않고, 현재까지 진행된 시간을 시간 전진 값으로 사

용한다. 그림 10에서의 DEVS 모델의 동작은 다음과 같다. 로그인의 실패 이벤트가 발생하면 상태 천이를 수행한다. 이 후 정해진 시간 ta 이전에 로그인 성공 이벤트가 발생하거나 로그인 실패 이벤트가 발생하지 않으면 패스워드 공격이 일어나지 않은 상태 즉, 0인 상태로 돌아간다. 이와 반대로 1인 상태에서 ta 시간 이전에 로그인 실패 이벤트가 발생하면 상태 2인 상태로 천이한다. 상태 2인 상태인 상태는 현재까지 진행된 시간을 시간 전진 값으로 사용하고, 이 시간 동안 로그인 성공 이벤트나 다른 이벤트가 발생하지 않으면 0인 상태로 천이한다. 그렇지 않으면 3인 상태로 천이하고 3인 상태에서는 패스워드 공격이라는 이벤트를 출력으로 내보내는 모델이다. 위 모델에서 로그인 실패 횟수는 사용자의 입력에 따라 다르게 줄 수 있다.

5.2 exec() 취약성

Hao Chen 등은 리눅스와 솔라리스, 유닉스의 setuid() 함수에 대한 보안의 취약성에 대하여 연구하였다[11].

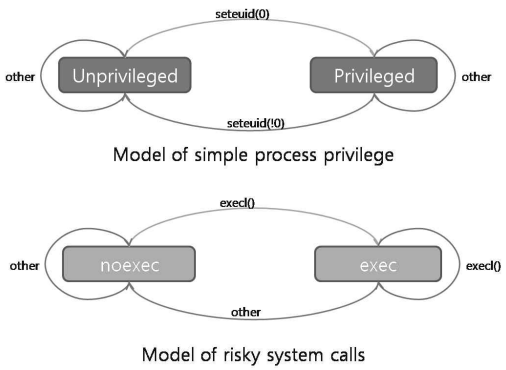

그림 11 프로세스 권한 모델과 시스템 콜 모델

그림 11는 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, 해당 프로세스가 더 높은 권한을 가져도 되는 지 검사해야 하며, 자신이 가지고 있는 권한을 포기해야 한다. Hao Chen 등은 위의 보안 취약성 모델을 오토마타로 생성한 후에 프로그램 소스 코드의 제어 흐름 그래프를 도출한 다음에 해당 제어 흐름 그래프가 보안 취약성을 나타내는 오토마타를 통과하는 지를 확인하였다. 이에 따라 프로그램을 수행하지 않고 보안의 취약성을 분석함으로써 데이터 흐름을 고려하지 않고 검사를 수행하였다.

본 사례 연구에서는 Aspect DEVS 검증 틀을 사용하여 제어 흐름 및 데이터의 흐름도 고려하여 검증을 수행할 수 있도록 작성된 소프트웨어의 소스 코드에 대하여 관점 지향 프로그래밍 기법을 사용하여 앞서 정의한 검증 과정들을 거쳐 검증을 수행하였다. 다음 그림은 프로세스 권한 모델과 execl() 시스템 콜 모델을 사용하여 작성한 사용자 요구사항 DEVS 원자 모델이다.

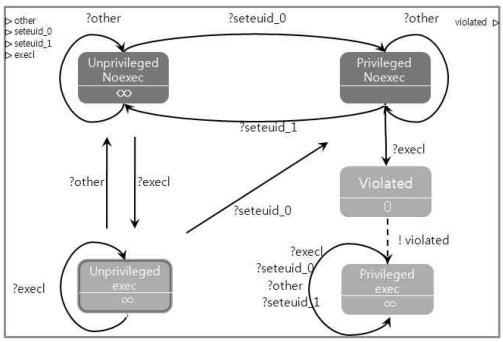

그림 12 보안 취약성 검증 모델

보안 취약성을 검증하기 위한 검증 모델은 other, seteuid_0, seteuid_1, execl 포트를 가지고 있다. 즉, 소프트웨어의 수행 과정 중에서 발생되는 이벤트들에 따라서 해당 이벤트에 따라 권한을 포기하지 않은 상태에서 execl과 같은 시스템 콜을 호출하는 것을 발견하면 violated port로 출력 이벤트를 발생시킨다. 이러한 요구사항을 바탕으로 다음 예제 프로그램을 검증하였다.

```
#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 13 보안 취약성을 내재한 프로그램

그림 13는 보안 취약성을 가진 프로그램으로써 C++로

구현되어 있다. 그림 14는 AOP로 모니터링해야 하는 위치를 기술한 Aspect 명세이다. Aspect 명세에서 seteuid 함수의 호출과 호출 되었을 때의 인자 값에 따라 이벤트를 다르게 발생시켜 검증 모델로 전송한다. 이와 마찬가지로 execl 함수가 호출 되었을 때의 이벤트를 검증 모델로 전송하도록 Aspect 명세를 한 후 이를 Aspect DEVS 검증틀을 사용하여 검증을 수행한 결과는 그림 15와 같다.

```
aspect Generator {
    pointcut method(int i) = args(i) && execution("% seteuid(...)");
    advice method(i) :before(int i)
    {
        if(i == 0)
        {
            // Send seteuid Event Message to Server
            printf ("Sending SetEuid_0 Message₩n");
            m_NetHandler.Send( SETEUID_0, NULL );
        }
        else
        {
            // Send seteuid Event Message to Server
            printf ("Sending SetEuid_1 Message₩n");
            m_NetHandler.Send( SETEUID_1, NULL );
        }
    }

    advice execution("% getpwuid(...)") :
    before () {
        // Send seteuid Event Message to Server
        printf ("Sending Other Message₩n");
        m_NetHandler.Send( OTHER, NULL );
    }

    advice execution("% execl(...)");
    before(){
        printf ("Sending EXECL Message₩n");
        m_NetHandler.Send( EXECL, NULL );
    }
};
```

그림 14 보안 취약성 분석을 위한 Aspect Specification

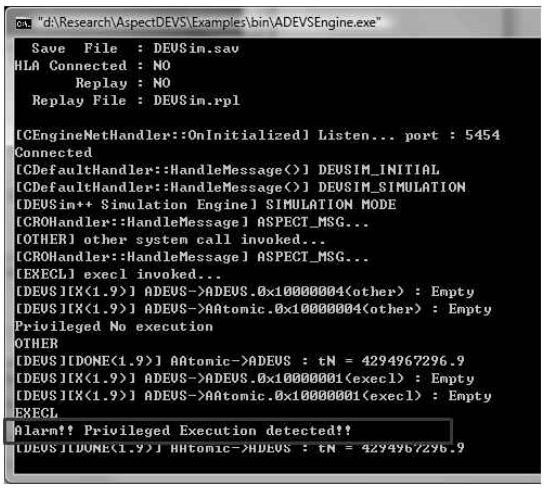

그림 15 검증 수행 결과

위 결과에서 볼 수 있듯이 프로그램의 수행에 따라 이벤트가 생성되어 검증 모델에 전달되어 권한을 가진 상태에서 시스템 콜 함수가 호출되었음을 확인할 수 있다.

7. 결론

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

또한, 시뮬레이션 틀에서 모델을 검증하기 위해서는 모델 실행 시 발생하는 이벤트를 모니터링하기 위하여 모니터링을 가능하게 하는 코드를 삽입해야 한다. 이와 같은 과정에서 실제 시뮬레이션의 수행과 상관없는 모니터링 코드의 삽입에서 발생할 수 있는 문제점을 Aspect DEVS 검증 틀에서는 AOP를 사용하여 개발된 소프트웨어에 검증 관점을 추가함으로 코드 삽입 시에 발생할 수 있는 문제점을 해결한다. 또한 기존에 검증을 위해서 추가한 모니터링 코드와 이를 검증하기 위한 검증기를 시뮬레이션 모델과 분리하여 타 시뮬레이션 틀에서 재사용할 수 있도록 함으로써 기존에 개발된 시뮬레이션 틀의 유지 보수 비용을 줄일 수 있는 장점이 있다. 그리고 사용자의 요구사항에 따라 Safety 또는 Liveness 특성을 모델링할 수 있어 다양한 사용자의 요구사항을 DEVS 형식론으로 모델링하여 구현된 DEVS 모델로 검증을 수행할 수 이TEk. 이를 통해서 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.

참 고 문 헌

[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] Tag G. Kim, Lecture Note: EE612, EE Dept, KAIST, Available: http://smslab.kaist.ac.kr/Course/EE612/
[8] T. G. Kim, DEVSimHLA User's Manual, 2007. Available: http://smslab.kaist.ac.kr
[9] R. Laddad, “AspectJ in Action: Practical Aspect-Oriented Programming,” Manning, 2003.
[10] Win32.HLLW.Shadow.based exploits http://www.antivirusworld.com/news/win32-hllw-shadow-based-exploits-vulnerability-of-windows.html
[11] Hao Chen, Drew Dean, and David Wagner, "Model Checking One Million Lines of C Code," In Network and Distributed System Security (NDSS 2004), 2004.
[12] 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

최창범 (cbchoi@smslab.kaist.ac.kr)

2005 경희대학교 전기전자정보컴퓨터공학부 학사
2007 KAIST 전자전산학과 석사
2007～현재 KAIST 박사과정

관심분야 : DEVS 형식론, 소프트웨어 품질 보증, VV/A

김탁곤 (tkim@ee.kaist.ac.kr)

1975 부산대학교 전자공학과 학사
1980 경북대학교 전자공학과 석사
1988 Univ. of. Arizona, 전기및컴퓨터공학과 박사
1980~1983년 부경대학교, 통신공학과, 전임강사
1987~1989년 (미)아리조나 환경연구소, 연구엔지니어
1989~1991년 Univ. of Kansas, 전기및컴퓨터공학과, 조교수
1991~현재 KAIST 전자전산학과, 교수
- 한국시뮬레이션 학회 회장 역임
- 국제시뮬레이션학회(SCS) 논문지(Simulation) Editor-In-Chief 역임
- SCS Fellow
- 모델링 시뮬레이션 기술사(미국)
- *Who's Who in the World* (Marguis 16th Edition, 1999) 등재
- 연합사, 국방부/합참, 기품원 자문위원 역임
- KIDA Fellow 역임
- ADD 자문위원(현)

● 연구 실적
- 교재
  Theory of Modeling and Simulation, Academic Press, 2000, 등
  영어 교재/Chapter 8 권 저술
- 연구 논문
  국.·내외 학술지/학술대회 M&S 관련 논문 200 여 편 발표
- 과제 수행
  훈련, 분석, 획득, 전투실험 관련 국방 M&S 과제 10 여 건 수행(중)

관심분야 : 모델링/시뮬레이션 이론, 방법론 및 환경개발, 시뮬레이터 연동