컴포넌트 기반 개발을 위한 점진적 시뮬레이터 설계

신영술, 박인수, 이정선, 이우진
경북대학교 전자전기컴퓨터학부
e-mail : youngsulshin@google.com

A Design of Incremental Simulator Supporting Component-Based Approach

Youngsul Shin, Ho Dong Ryu, In Su Park, Jung Sun Lee, Cao Thi Ly
Jin Wook Kwon, Mi Heui Seok, Woo Jin Lee
School of Electrical Engineering and Computer Science, Kyungpook National University

요 약

컴포넌트 기반 개발방법론의 이점은 공통의 인터페이스를 가지는 모델의 재사용성과 확장성이다. 시스템은 컴포넌트의 조립을 통하여 점진적으로 개발이 가능하다. 컴포넌트의 행위를 검증하기 위한 기존의 시뮬레이션은 모델을 이용한 검증만 가능했다. 하지만 모든 컴포넌트의 모델링 및 구현이 동시에 완료되지 않기 때문에 점진적인 컴포넌트 기반의 개발을 지원하기 위해서는 모델과 코드를 연계한 시뮬레이션이 필요하다. 본 논문에서는 모델과 코드를 연계한 점진적인 시뮬레이션 기법을 제안한다.

1. 서론

시스템을 모델링 하는 언어 중에서 UML 은 객체지향적인 소프트웨어 개발에서 주로 사용된다[1]. UML 은 시스템의 구조적인 측면과 행위적인 측면을 표현할 수 있는 다이어그램들을 지원하고, 직관적인 표기법을 제공하므로 시스템을 쉽게 기술할 수 있다. 하지만 UML 은 정형화(Formality)가 부족하여 시스템을 자동적으로 분석하기 위하여 다른 정형적 방법을 이용하여 변형하여야 한다[2].

모델 검증 방법 중에서 시뮬레이션과 모델 체킹(Model Checking)이 있다. 시뮬레이션은 사용자가 모델과의 상호작용을 통하여 시스템을 원하는 상태로 조작함으로써 모델을 검증하는 기능을 제공한다[3]. 시뮬레이션 과정을 기록하여 시스템의 행위를 정량적으로 분석할 수 있으며, 이전에 이루어진 시뮬레이션 과정을 동일하게 재생할 수 있다. 모델 체킹은 시스템을 정형 언어로 기술하고, 기술된 모델이 제약사항을 만족하는지 검사할 수 있다[4]. 모델 체킹에서 발견한 시스템의 특정 행위는 시뮬레이션을 이용하여 가시적으로 확인할 수 있다.

컴포넌트 기반의 개발 방법론은 시스템을 단위 기능을 가진 컴포넌트로 나누고, 각 컴포넌트 단위로 개발한다. 컴포넌트는 공통의 인터페이스를 갖기 때문에 다른 시스템에서 조립되어 재사용될 수 있다. 시스템을 컴포넌트로 나누지 않는 기존의 개발 방법론은 시스템의 복잡도가 높아지면 개발에 드는 비용이 상승하고, Time-To-Market 의 요구사항을 만족시키기 어렵다. 컴포넌트 기반 개발방법은 기존에 검증된 컴포넌트를 재사용함으로써 복잡한 시스템을 개발하는데 필요한 비용과 시간을 줄일 수 있다[5].

본 논문에서는 컴포넌트 기반 개발방법론의 장점을 효율적으로 지원하는 점진적인 시뮬레이션을 제시한다. 점진적인 시뮬레이션은 모델과 코드를 연계하여 시뮬레이션 함으로써 설계 단계와 구현 단계를 유연하게 연결한다. 기존의 시뮬레이션은 모델만 검증할 수 있지만, 점진적인 시뮬레이션은 모델과 코드를 연계하여 시뮬레이션 하기 때문에 설계와 구현 단계에서 모든 컴포넌트가 모델링 되거나 구현되지 않아도 개발 단계를 연속하여 컴포넌트 검증이 이루어진다.

이후 논문의 구성은 제 2 장에서 점진적인 시뮬레이션의 개념을 기술하고, 점진적 시뮬레이션에 필요한 소프트웨어 구조를 제시한다. 제 3 장에서 결론 및 향후 연구를 기술한다.

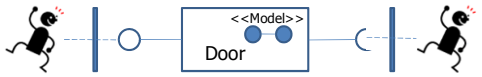
2. 컴포넌트 기반의 점진적 시뮬레이션

시스템 검증 방법 중의 하나인 시뮬레이션이 컴포넌트 기반의 개발 방법론을 효과적으로 지원하기 위해서는 점진적인 시뮬레이션 기법을 제공해야 한다. 이 장에서는 컴포넌트 기반의 점진적 시뮬레이션 기법과 점진적 시뮬레이션을 지원하는 시뮬레이터의 소프트웨어 구조를 제시한다.

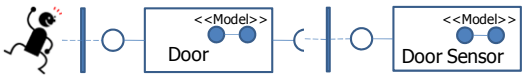
2.1 모델-코드 연계 시뮬레이션

시스템을 구성하는 각각의 컴포넌트는 독립적으로 구현이 된다. 구현 단계에서 일부 컴포넌트는 구현이 완료되지만, 동시에 아직 모델로 남아있는 컴포넌트도 존재한다. 컴포넌트는 단위 테스팅은 가능하지만 연계된 컴포넌트 간의 통합 테스팅은 모든 컴포넌트가 구현되어야 가능하다. 컴포넌트 기반의 점진적인

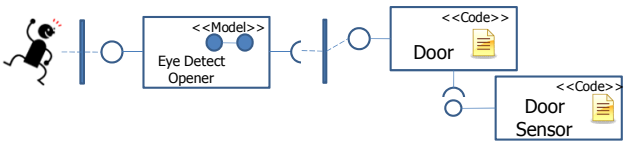
개발을 효과적으로 지원하기 위해서는 컴포넌트를 검증하기 위한 점진적인 시뮬레이션이 필요하다.



(가) 단일 컴포넌트 모델 시뮬레이션



(나)컴포넌트 모델 통합 시뮬레이션



(다)모델과 소스 코드 연계 시뮬레이션

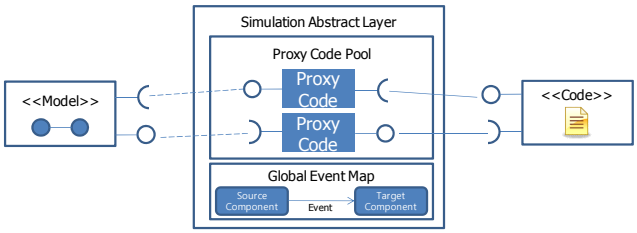
(그림 1) 점진적 시뮬레이션 개념

점진적인 시뮬레이션은 모델과 코드를 연계한 시뮬레이션 기법이다. 그림 1 은 점진적인 시뮬레이션의 개념을 보인다. 그림 1 (가)는 시스템을 구성하는 일부 컴포넌트가 모델링 되었다. 컴포넌트를 검증하기 위해 시뮬레이션을 이용하고 컴포넌트가 필요로 하는 외부 컴포넌트의 행위는 개발자가 시뮬레이터와의 상호작용을 통하여 대신한다. 그림 1 (나)에서 이전 단계에서 정의되지 않은 컴포넌트 중에서 일부 컴포넌트가 정의되었으며, 이미 정의된 컴포넌트는 상세히 모델링 되었다. 구체적으로 모델링 된 컴포넌트 중에서 점진적 시뮬레이션을 통한 검증이 완료된 컴포넌트는 구현 단계로 넘어간다. 구현이 완료된 컴포넌트는 시스템에 통합되어 검증되어야 한다. 그러나 시스템을 구성하는 일부 컴포넌트가 모델링 단계에 남아있거나, 정의조차 되지 않았다면 통합 검증이 어렵다. 이 경우에는 그림 1 (다)와 같이 구현된 컴포넌트는 점진적 시뮬레이션을 이용하여 모델 단계의 다른 컴포넌트와 통합되어 검증될 수 있다.

2.2 프록시 코드를 이용한 점진적 시뮬레이션

시스템의 구조를 기술하는 UML 의 컴포넌트 다이어그램은 컴포넌트 간의 상호작용 관계를 볼 수 있다. 상호작용의 관계는 Required Interface 와 Provided Interface 를 통하여 표현된다. 컴포넌트 다이어그램에 표현된 인터페이스 관계는 모델과 코드 간에도 유지된다. 점진적 시뮬레이션을 지원하는 시뮬레이터는 이러한 인터페이스 관계의 정보를 컴포넌트 다이어그램 모델로부터 추출하여 Global Event Map(GEM)을 작성한다. GEM 에는 인터페이스 관계와 해당 컴포넌트의 구현 여부를 나타내는 타입 정보가 포함된다.

모델과 코드를 연계한 시뮬레이션 과정에서 외부 컴포넌트로 전달될 글로벌 이벤트가 모델이나 코드에서 발생하면 시뮬레이터는 GEM 의 정보를 참조하여 이벤트를 해당 컴포넌트로 전달한다. 모델과 모델 간의 글로벌 이벤트는 메시지 형태로 해당 모델에게 곧바로 전달된다. 모델과 코드 간의 글로벌 이벤트는 프록시 코드를 통하여 전달된다. 시뮬레이터의 Simulation Abstract Layer(SAL)은 GEM 을 참조하여 글로벌 이벤트에 해당하는 프록시 코드를 찾는다. SAL 은 모델과 코드 사이에 위치하여 서로의 형태를 직접 볼 수 없도록 한다. 모델은 자신과 관계있는 코드를 모델로 보게 되고, 코드는 자신과 연계되는 모델을 코드로 보게 된다. 컴포넌트 다이어그램의 인터페이스에는 컴포넌트 간에 전달되는 이벤트가 정의되어 있기 때문에 프록시 코드는 컴포넌트 다이어그램을 이용하여 자동 생성할 수 있다.



(그림 2) 프록시를 이용한 모델과 코드의 연계

모델에서 코드로 전달될 글로벌 이벤트가 발생하면, SAL 은 GEM 을 이용하여 해당되는 프록시 코드를 찾는다. 이 프록시 코드는 소스 코드로 구현된 해당 컴포넌트에 이벤트를 전달하고 결과를 시뮬레이터로 반환한다. 코드에서 모델로 글로벌 이벤트를 전달할 경우, 코드는 프록시 코드를 직접 호출한다. 호출된 프록시 코드는 SAL 로 이벤트를 전달하고 SAL 은 GEM 을 참조하여 이벤트를 처리할 모델을 찾는다. 그림 2 는 프록시 코드를 통한 모델과 코드의 연계 방법을 보인다.

3. 결론

컴포넌트 기반 개발을 돕기 위하여 모델과 코드를 연계한 시뮬레이션 기법과 이를 위한 시뮬레이터의 소프트웨어 구조를 제시하였다. 프록시 코드는 모델과 코드가 상대 컴포넌트를 자신이 원하는 형태로 볼 수 있는 메커니즘을 제공한다. 프록시 코드는 모델로부터 자동 생성된다. 향후 연구 내용으로서 본 논문에서 제시한 점진적 시뮬레이션 기법을 구현할 예정이다. 또한 모델과 코드 간의 연계 시뮬레이션에 발생하는 동기화 문제를 해결할 예정이다.

참고문헌

[1] OMG, “Unified Modeling Language: Superstructure,” Version 2.2, 2009.

[2] Bernardi S, Donatelli S, Merseguer J, “From UML sequence diagrams and statecharts to analyzable Petri net models,” In Proc of WOSP 2002, 2002.

[3] B. Broekman, E. Notenboom, Testing Embedded Software, Addison Wesley, 2003.

[4] Ribeiro, O.R., Fernandes, J.M., Pinto, L.F., “Model checking embedded systems with PROMELA,” In Proc of ECBS 2005, 2005.

[5] George T. Heineman, William T. Council, Component-Based Software Engineering: Putting the Pieces Together, Addison Wesley Professional, 2001.