관점지향 개발 방법론을 지원하기 위한 SysML의 확장*

이재욱, 김두환, 홍장의
충북대학교 컴퓨터과학과
{jwlee, dhkim}@selab.cbnu.ac.kr, jehong@chungbuk.ac.kr

A SysML extension to support Aspect Oriented Software Development

Jae-Wuk Lee, Doo-Hwan Kim, Jang-Eui Hong
Dept of Computer Science, Chungbuk National University

요 약

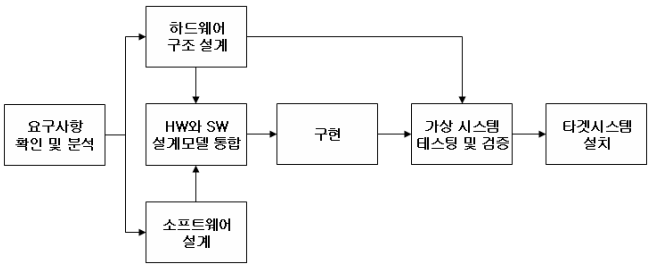
관점지향 개발 방법론은 시스템의 중복된 요소를 최소화 하여 높은 수준의 모듈화를 달성하기 위한 기법으로 그 적용 범위를 넓히고 있다. 특히 최근에는 임베디드 시스템 개발에 관점지향 개발 방법론을 적용한 사례가 증가하고 있다. 많은 임베디드 시스템들이 SysML과 같은 설계 언어를 이용하여 하드웨어와 소프트웨어를 동시에 설계하고 통합하는 방식으로 개발되고 있다. 그러나 이러한 하드웨어와 소프트웨어를 고려한 설계 언어들은 구조적 개발 방법론이나 객체지향 개발 방법론에 초점을 두고 있기 때문에 관점지향 개발 방법론을 지원하기에 많은 한계가 있다. 따라서 본 연구에서는 임베디드 시스템 개발에 관점 지향 개발 방법론을 지원하기 위해 확장된 SysML을 제안한다.

1. 서론

객체지향 개발 방법론은 기존의 개발 방법론에 비해 높은 수준의 모듈화를 구현할 수 있다는 장점이 있다. 하지만, 객체지향 소프트웨어 개발 방법론에서도 비슷한 기능이 여러 객체에서 요구 될 때는 비슷한 코드가 여러 모듈에 구현되어야만 한다. 이러한 단점을 극복하여 중복된 코드를 최소화 하고 더 높은 수준의 모듈화를 구현하기 위해 관점지향 개발 방법론[1]이 등장하였다.

기존의 관점지향개발 방법론에 대한 연구들은 대부분 구현 기법을 중심으로 진행되었다[2][3][4]. 그러나 소프트웨어 생명주기에 비추어 볼 때, 이전 단계에서의 체계적인 고려 없이 구현 단계에서만 관점지향 개발 방법론이 적용된다면 설계 모델과의 일관성이 다소 낮아지는 부작용을 수반 할 수 있다. 따라서 최근에는 분석 및 설계 단계에서 관점지향 개발 방법론을 적용하는 연구들[5][6][7][8]이 이루어지고 있다. 이러한 연구들을 살펴보면 AspectJ[2]와 같은 관점지향 프로그래밍 언어들을 기반으로 UML[9]을 이용한 소프트웨어 개발에 관점지향의 개념을 표현할 수 있도록 하였다.

최근의 임베디드 시스템 개발은 (그림 1)과 같이 하드웨어와 소프트웨어를 동시에 설계하여 통합하는 방법으로 개발하는 사례가 빈번히 등장하고 있다. 따라서 UML만을 이용한 소프트웨어 개발은 이러한 시스템 개발 지원에서 어려움을 겪는 경우가 발생하고 있다. 때문에 최근 OMG(Object Management Group)에서도 SysML[10]을 발표하여 이와 같은 시스템 개발을 지원하고 있다. 그러나 기존의 하드웨어와 소프트웨어를 고려한 설계 언어들은 대부분 구조적 개발 방법론이나 객체지향 개발 방법론에 초점을 두고 있기 때문에 관점 지향 개발 방법론을 지원하기에는 많은 제한점들이 있다. 또한 앞서 언급한 UML 기반의 관점지향 개발 방법론에 대한 연구들은 임베디드 시스템 개발에서 소프트웨어 개발만을 지원하므로, 하드웨어에 해당하는 부분에 관점지향 개발 방법론을 적용하기에 적합하지 않다. 따라서 본 연구에서는 소프트웨어 뿐 아니라 하드웨어를 동시에 설계하여 개발하는 임베디드 시스템 개발에서 관점지향 개발 방법론을 적용할 수 있도록 확장된 SysML을 제안한다.



(그림 1) 임베디드 시스템의 개발 과정

본 논문은 다음과 같이 구성된다. 2장에서는 기존의 관점지향 소프트웨어개발을 지원하기 위한 관련 연구들을 살펴보고, 3장에서는 본 연구에서 제시하는 관점지향 시스템 개발을 지원하기 위한 SysML의 확장에 대해 설명한다. 4장에서는 확장된 SysML을 예제시스템에 적용하는 과정을 보이고, 5장에서 결론 및 향후 연구에 대해 기술한다.

* 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음" (NIPA-2010-(C1090-1031-0001))

2. 관련연구

기존의 연구들 중 모델링 언어를 이용해 관점지향 개발 방법론을 지원하기 위한 연구로는 D. Stein[5]의 연구와 O. Aldawud[6]의 연구, R. Pawlak[7]의 연구가 대표적이다.

Stein의 연구는 AspectJ를 지원하는 것에 중점을 두어 UML notation을 이용하여 관점지향 설계 기법을 제시하였다. 해당 연구는 시퀀스 Diagram을 이용해서 Join point를 표현하였고, <<aspect>> 스테레오타입을 정의하여 이를 클래스에 이용함으로써 Aspect를 나타냈다. 이때 Aspect의 내부에는 해당 Aspect의 Joint point 및 이때 이용되는 Advice에 대한 정보가 표현된다. 핵심관심사와 횡단관심사가 결합되는 직조(Weaving)에 대한 표현은 Use Case Diagram을 이용해서 표현하였다.

Aldawud의 연구에서는 UML Profile을 확장하여 관점지향 개념을 표현하였다. Stein의 연구에서와 같이 Aspect를 표현하기 위해 스테레오타입 <<aspect>>를 정의하였다. 또한 스테레오타입 <<crosscut>>을 정의하여 이를 assosication에 이용함으로써 Aspect와 Class간의 결합관계를 표현하는데 사용하였다. 마지막으로 직조(Weaving)에 대해서는 State machine 상에서 트랜지션이 발생할 때 이에 관여하는 Aspect와의 mapping을 통해 표현하였다.

Pawlak의 연구는 UML notation을 확장하여 groups, pointcut relations, aspects의 세 가지 개념을 나타내는 것에 주안점을 두었다. “groups“은 핵심관심사들을 종류별로 그룹지어 횡단관심사와의 결합 관계를 표현한다. "pointcut relations"를 표현하기 위해서는 association에 저자가 정의한 <<pointcut>> 스테레오 타입을 이용하여 Aspect와 결합되는 클래스들을 나타내기 위해 사용된다. Aspect를 나타내기 위한 notation은 UML의 클래스를 확장하여 정의하였다.

이러한 연구들은 소프트웨어 개발 과정에서 구현단계뿐 아니라, 설계단계에서도 관점지향 개발 방법론을 적용할 수 있게 하기 위한 목적으로 수행되어 왔다. 하지만 이러한 연구들이 대부분 소프트웨어에 한정지어 UML을 사용하였으므로, 하드웨어에 대한 관점지향 개발 방법론을 지원하기에는 어려움이 있다.

3. 관점지향 개발 방법론을 지원하기 위한 SysML 확장

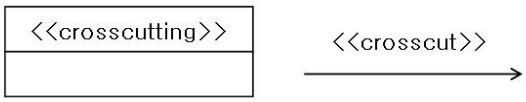
관점지향 개발 방법론은 다음과 같은 개념들로 이루어진다.

- 결합점(join point) : 직조에 의해 횡단관심사 코드가 결합되는 지점
- 교차점(Pointcut) : 결합점들을 지정하고, 그에 대한 환경정보를 가지고 있는 구조물
- 어드바이스(Advice) : 결합점에서 실행되는 코드. 결합점의 실행 전(before), 실행 후(after), 결합점을 대체(around)하여 실행
- 에스펙트(Aspect) : 횡단관심사를 모듈화 하는 단위. 결합점, 어드바이스등을 정의
- 위버(Weaver) : 횡단관심사와 핵심관심사를 조합하는 기능을 하는 소프트웨어

본 장에서는 위와 같은 개념들을 반영하여 임베디드 시스템 개발에 있어서 관점지향 개발 방법론의 적용이 용이하도록 SysML을 확장한다.

3.1 Requirement Diagram 확장

핵심관심사와 횡단관심사는 ‘요구사항 확인 및 분석’단계에서 식별될 수 있다. 때문에 본 연구에서는 이러한 특징을 고려하여 SysML의 Requirement Diagram의 확장을 수행하였다. (그림 2)는 Requirement Diagram에서 횡단관심사를 나타내기 위해 추가되는 notation을 나타낸다.

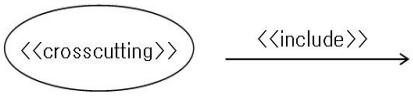


(그림 2) Requirement Diagram 확장

횡단관심사로 분류되는 각각의 Requirement에는 <<crosscutting>> 스테레오타입을 기술함으로써 핵심관심사와 구분한다. <<crosscut>>은 핵심관심사와 횡단관심사의 관계를 표현하기 위해 사용한다.

3.2 Use Case Diagram 확장

핵심관심사나 횡단관심사는 모두 하나 또는 그 이상의 모듈에서 제공하는 기능에 해당한다. 때문에 기능적 모델링을 수행하기 위해 사용되는 Use Case Diagram이 이들의 관계를 나타내기에 적합하다. Use Case Diagram에서 횡단관심사를 나타내기 위해 본 연구에서는 (그림 3)과 같이 <<crosscutting>> 스테레오타입을 해당 Use Case에 사용하였다. 이때, 횡단관심사에 해당하는 Use Case는 <<include>> 관계를 통해서만 사용이 가능하다.



(그림 3) Use case Diagram 확장

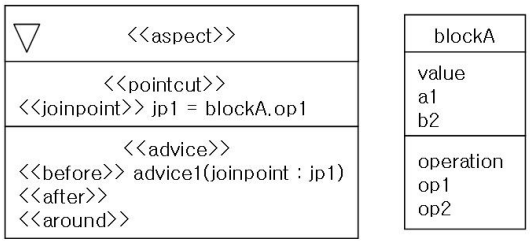
3.3 Block Diagram 확장

식별된 Aspect의 구조 및 다른 block들과의 관계를 표현하기 위해 이를 확장하였다. Aspect에 해당하는 block을 표현하기 위해 본 연구에서는 <<aspect>> 스테레오 타입을 정의하였다. (그림 4)에서 나타내는 바와 같이 Aspect에 해당하는 block은 일반적인 block에 <<aspect>> 스테레오타입을 기술함으로써 표현된다. Aspect에 대한 설계를 수행할 때는 advice와 pointcut에 대한 정보가 요구되는데, 이를 표현하기 위해 block의 compartment에 각각 <<advice>>와 <<pointcut>>을 이용하여 나타낸다.

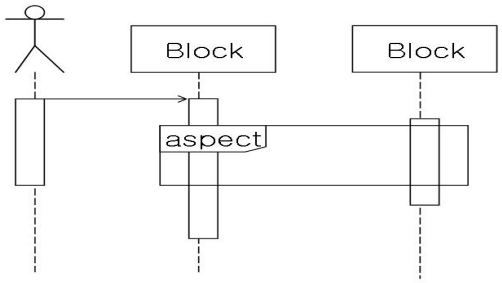
3.4 Sequence Diagram 확장

관점지향 시스템 개발을 지원하는 시스템의 동적 구조 모델링을 위해서는 Sequence Diagram을 확장하였다. 먼저 (그림 5)와 같이 핵심관심사 모듈과 Aspect가 직조되는 부분에 "aspect" Combined Fragment를 이용하여 표현한다. 이 결합지점에서 Aspect의 동작에 대한 상세정보가

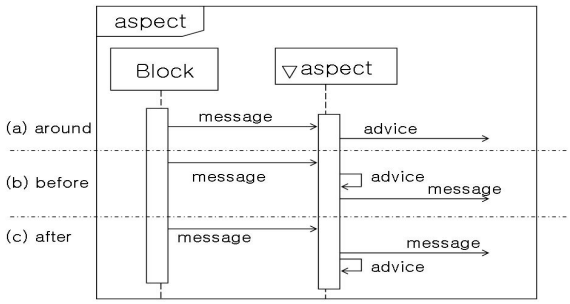
요구될 때는 (그림 6)과 같은 Aspect Sequence Diagram을 추가한다. 여기서 (a), (b), (c)는 around 속성에 따라 각각 "around", "before", "after"의 모델링 형태의 예시를 나타낸다.



(그림 4) Block Diagram 확장



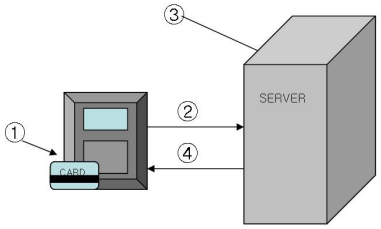
(그림 5) 확장된 Sequence Diagram



(그림 6) Aspect Sequence Diagram

4. 예제 시스템의 적용

본 연구에서 제안하는 관점지향 개발 방법론을 지원하기 위해 확장된 SysML을 이용한 시스템 모델링의 유용성을 확인하기 위해 예제 시스템에 적용해 보았다.



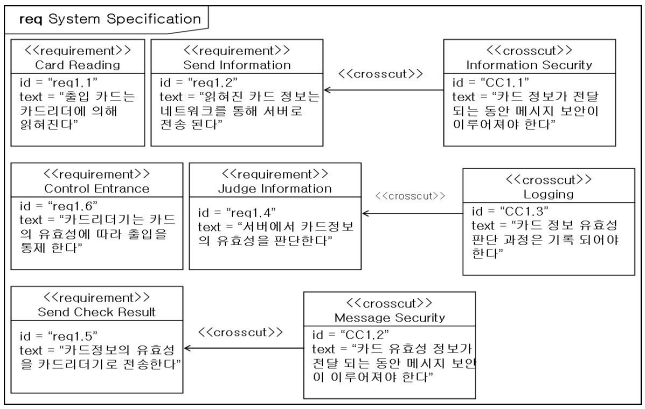
(그림 7) 출입 관리 시스템

4.1 예제 시스템 정의

정의된 예제 시스템은 출입 통제 관리 시스템으로 (그림 7)에서 나타내는 바와 같다. 해당 예제 시스템은 출입을 원하는 사용자가 카드리더기에 자신이 소지한 출입 카드를 인식시키고(①), 카드리더기는 해당 출입 카드의 정보를 읽어 서버로 전송한다(②). 서버는 카드리더로부터 전송된 정보를 통해 인가된 출입 카드인지 여부를 식별한다(③). 식별된 결과는 다시 카드리더기를 전송되어(④) 출입 통제를 수행하도록 한다.

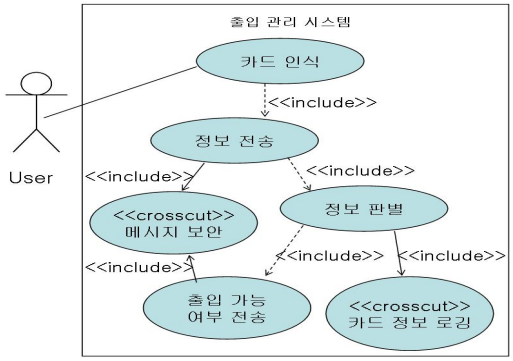
4.2 확장된 Diagram을 이용한 예제 시스템

예제 시스템의 모델링을 위해 먼저 Requirement Diagram을 작성하였다. (그림 8)은 예제 시스템에서 식별된 요구사항들을 기반으로 작성된 Requirement Diagram이다. 이 중 <<crosscut>>으로 표기된 "Information Security", "Message Security", "Logging"이 횡단 관심사로 추출된 요구사항을 나타낸다.



(그림 8) 예제 시스템의 Requirement Diagram

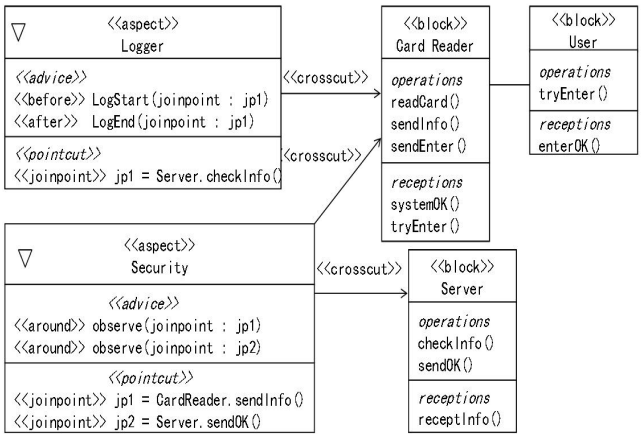
작성된 Requirement Diagram을 기반으로 작성된 Use Case Diagram은 (그림 9)와 같다. Use Case Diagram을 이용한 요구사항 분석 과정에서는 앞서 작성된 Requirement Diagram에서의 "Information Security"와 "Message Security"에 해당하는 요구사항 들이 횡단관심사로써 추출되어 하나의 Use Case로 식별되었다. 그 외에도 "Logging" 또한 횡단관심사로 식별되었음을 알 수 있다.



(그림 9) 예제 시스템의 Use Case Diagram

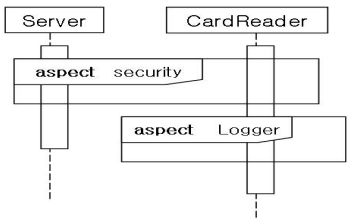
다음으로 Block Diagram을 이용한 시스템 정적 구조 Modeling을 수행한다. 이 과정에서 작성된 Block

Diagram은 (그림 10)에서 나타내는바와 같다. 앞서 작성된 Use Case Diagram에서 Aspect로 식별된 "Logging"과 "Security"가 다른 <<aspect>> stereo type을 통해 Aspect로 모델링되었음을 확인할 수 있다.

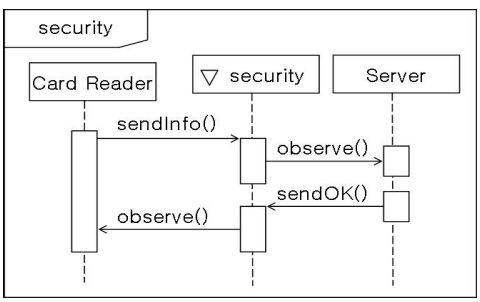


(그림 10) 예제 시스템의 Block Diagram

마지막으로 예제 시스템의 동적 구조를 나타내기 위해 Sequence Diagram을 작성한다. (그림 11)에 나타난 Sequence Diagram은 "Server"가 인식된 카드의 출입 가능 여부를 판단한 후에 "CardReader"로 결과를 송신하는 과정에 대한 과정을 나타낸다. "Server"에서 "CardReader"로 전송하는 메시지는 Aspect로 식별된 "security"가 관여하므로 해당 Aspect를 삽입하였다. 또한 "security" Aspect의 세부적인 동작을 표현하기 위해 (그림 12)와 같은 Aspect Sequence Diagram을 작성하였다. 현재 "security" Aspect는 advice에서 "around"를 나타내므로 해당 Aspect가 Server의 메시지를 받아 이를 대체하여 처리하는 모습을 보이고 있다.



(그림 11) 예제 시스템의 Sequence Diagram



(그림 12) Aspect의 동작을 상세화한 Sequence Diagram

5. 결론 및 향후 연구

관점지향 개발 방법론은 횡단관심사의 모듈화를 통해 여러 모듈에서 요구하는 요구사항을 모듈화 하여 객체지향 개발 방법론의 한계를 보완할 수 있다. 때문에 다수의 연구에서 관점지향 개발 방법론을 적용한 소프트웨어 개발을 달성하기 위한 노력을 해왔다. 하지만 최근 소프트웨어뿐 아니라 하드웨어까지 동시에 개발하는 사례가 점차 증가하고 있는 임베디드 시스템 개발에서는 이러한 연구들을 적용하여 관점지향 개발 방법론의 이점을 충분히 살리기엔 어려움이 있다. 본 연구에서는 이러한 임베디드 시스템 개발에서의 관점지향 방법론 적용을 용이하게 하기 위해 SysML 확장을 수행하였다. 확장된 SysML은 예제 시스템 모델링을 통해 "요구사항 확인 및 분석 단계"에서부터 "설계"단계에 이르기까지 시스템 개발의 다양한 모델링 활동에서 관점지향 개발 방법론의 지원이 가능함을 확인 할 수 있었다. 향후에는 현재 확장에서 다루지 않은 Diagram들을 확장하고, 이를 이용한 관점지향 시스템 설계 모델 기반의 분석 등에 대한 연구를 수행하고자 한다.

참고문헌

[1] G. Kiczles, et al., "Aspect-Oriented Programming", ECCOOP, pp220-242, 1997.

[2] G. Kiczales et al., "An Overview of AspectJ", LNCS, Vol. 2072, Springer, pp.327-353, 2001.

[3] D. suvee, et al., "JAsCO:an Aspect-Oriented approach tailored for Component Based Software Development", AOSD Conference, pp.21-29, 2003.

[4] R. Pawlak, et al., "A Flexible soution for Aspect-Oriented Programming in Java", in Proc. of Reflection, LNCS, VOL. 2192, pp.1-21, 2001.

[5] D. Stein, et al., "An UML-based Aspect-Oriented Design Notation For AspectJ", in Proc. of the 1st International Conference on AOSD Enschede, 2002.

[6] O. Aldawud et al., "UML Profile For Aspect-Oriented Software Development", in Proc. of Workshop of Aspect-Oriented Modeling with UML of AOSD, 2003.

[7] R. Pawlak et al., "A UML Notation for Aspect-Oriented Software Design", in The 1st Int. Workshop on Aspect-Oriented Modeling with UML, in conjunction with UML2004, Lisbon, 2004.

[8] F. Losavio et al., "UML Extensions for Aspect Oriented Software Development", in Journal of Object Technology, vol. 8, no. 5, 2009.

[9] OMG, "Unified Modeing Language: Superstructure", version 2.1.1(formal/2007-02-03), 2007.

[10] OMG, "Systems Modeling Language", Version 1.2 (formal/2010-06-02), 2010.