중첩된 구조의 유즈케이스 모델을 이용한 임베디드 시스템의 요구사항 분석 방법 제안

진용호[0] 배두환
한국과학기술원 전산학과
(yhchin, bae)@salmosa.kaist.ac.kr

A Requirements Analysis Method for Embedded System Using Nested Use Case Model

Yong-Ho Jin, Doo-Whan Bae
Dept. of CS, KAIST

요 약

현재 임베디드 시스템을 위한 객체지향 개발 방법론이 많이 제시되고 있지만, 이들 방법론에서 요구사항 분석 모델은 하드웨어와 소프트웨어를 포함하는 임베디드 시스템의 특징을 반영하지 못하고 있다. 즉, 개발하고자 하는 시스템의 경계를 명확히 하지 못하고 있다. 본 논문에서 제시하고 있는 중첩된 구조의 유즈케이스 모델에서는 하드웨어를 포함한 시스템 경계와 소프트웨어 시스템 경계를 나누고 이와 관련된 모델링 요소들-액터와 유즈케이스-을 명확히 구분한다. 명확한 시스템 경계는 시스템의 환경을 포함한 시스템을 이해하는 데 도움을 주며, 하드웨어 측면의 개발이 소프트웨어 측면의 개발에 선행하는 일반적인 임베디드 시스템 개발 과정에 적합하고, 이후 분석 과정과도 자연스럽게 연결된다. 또한 제시된 유즈케이스 모델을 이용한 모델링 절차를 제시하며, 모델의 구성요소를 효율적으로 추출할 수 있는 방법을 제시한다.

1. 서론

일반적인 비즈니스 어플리케이션(Business Application) 도메인에서 언급되는 시스템은 대부분 소프트웨어 시스템을 지칭한다. 이에 반해, 임베디드 시스템 (Embedded System)의 개발은 소프트웨어 뿐만 아니라 하드웨어 개발, ASICs 디자인, 기계장치 등과 관련된 여러 가지 활동을 포함한다. 임베디드 시스템의 여러 가지 요소들은 소프트웨어와 직/간접적인 관계를 갖는다. 이 중 시스템의 입/출력 장치(Input/Output Device)는 소프트웨어 시스템에 직접 영향을 미치는 중요한 요소로, 일반적인 비즈니스 어플리케이션 도메인에서는 찾아보기 힘든 것이다. 기존 방법론의 시스템 요구사항 분석 과정에서는, 내부 혹은 환경 요소로 하드웨어를 포함하는 임베디드 시스템의 특성을 이끌어 내지 못하고 있다.

본 논문의 구성은 다음과 같다. 2장에서는 유즈케이스 모델링에서 고려되어야 하는 비즈니스 어플리케이션 도메인과 임베디드 시스템 도메인의 특징을 비교하여 살펴보고, 3장에서는 기존의 임베디드 시스템을 위한 개발 방법론을 살펴본다. 4장에서는 본 논문에서 제안하는 유즈케이스 모델에 대해서 설명을 하고, 5장에서 결론을 맺는다.

2. 비즈니스 어플리케이션 도메인과 임베디드 시스템 도메인에서의 유즈케이스 모델

본 장에서는 비즈니스 어플리케이션 도메인에서 사용되는 유즈케이스 모델링의 특성을 살펴보고, 이를 임베디드 시스템 도메인에 적용하였을 때 생기는 문제점에 대하여 살펴본다.

2.1 비즈니스 어플리케이션 도메인

광범위한 의미로 유즈케이스 모델(Use Case Model)은 관심있는 업무(Business)를 이끌어 내는데 사용되며, 대상 업무와 이와 관련된 주변 요소(Environment)를 기술한다[6]. 소프트웨어를 개발하고자 하는 의도에서 바라보면, 대상 업무는 개발하고자 하는 시스템으로 모델링 되어 지고, 시스템은 제공해야 할 기능인 유즈케이스(Use Case)로 구성된다. 주변 요소로는 개발 시스템을 사용하는 사용자 혹은 기계적 요소가 포함되며, 액터(Actor)로 표현된다. 액터는 시스템을 사용하려고 하는 역할(Role)을 나타내며, 유즈케이스를 정의하기 전에 식별되어 지는 것이 일반적이다.

비즈니스 어플리케이션 도메인에서의 시스템은 대부분 소프트웨어 시스템으로 액터는 사람 혹은 개발 시스템과 연결된 외부 시스템으로 한정될 수 있다. 외부 시스템은 도메인 별로 그 역할이 크게 상이하지 않으므로 이 논문에서 자세히 논의하지 않는다. 사람 액터의 경우 시스템과의 상호작용은 대부분 표준 입/출력 장치를 통해서 이루어 진다. 표준 입/출력 장치에 대한 처리는 운영체제에서 맡기 때문에, 시스템을 이루는 구성요소로써 고려하지 않는다.

유즈케이스는 액터에게 가치를 제공하기 위한 시스템 내의 일련의 일들로 정의된다. 유즈케이스는 일반적으로 액터의 역할을 기준으로 정의됨으로, 액터의 성격에 따라 유즈케이스의 성격도 크게 달라질 수 있다. 비즈니스 어플리케이션 도메인에서의 액터는 대부분 사람이기 때문에, 액터들의 성격이 유사하다. 액터의 성격이 유사하다는 것은 액터가 발생시키는 유즈케이스의 성격(Characteristics) 혹은 수준(Granularity)이 유사하다는 것을 의미한다.

2.2 임베디드 시스템 도메인

임베디드 시스템은 앞서 언급했듯이 소프트웨어뿐만 아니라 감지장치(Sensor), 작동기장치(Actuator), 버튼과 같은 입력장치(Input Device), 화면과 같은 출력장치(Output Device) 등과 같은 많은 주위 디바이스(Device)를 포함하고 있다. 디바이스는 크게 사람과 상호 작용을 하는 것과 그렇지 않은 것으로 구분된다. 사람과 상호 작용하는 디바이스는 표준 입/출력 장치 외에도 시스템마다 특정한 입/출력 장치를 포함한다. 표준 입/출력 장치가 운영체제에 의해 제어되는 반면 특정 입/출력 장치는 시스템 자체적인 처리를 필요로 한다. 사람과 상호 작용을 필요로 하지 않는 디바이스는 자율적인 디바이스(Autonomous Device), 타이머(Timer)를 들 수 있다. 이 디바이스들은 비즈니스 어플리케이션 도메인에서 언급되었던 액터들과는 상관없이 시스템 내부의 요소로서 시스템이 어떤 일을 수행하도록 동작시킬 수 있는 요소들이다.

임베디드 시스템의 경계는 디바이스를 포함한 경우와 소프트웨어만을 포함한 경우로 설정될 수 있으며, 기존의 개발 방법론에서는 이 두 가지 경계를 혼용하여 사용하고 있다. 이는 시스템의 기능을 수행하도록 동작시키는 역할을 사람이나 외부 시스템 뿐만 아니라 시스템의 디바이스가 수행하는 데서 발생한다. 이로 인한 문제점은 다음과 같다. 첫째, 기존의 사람 혹은 시스템 액터가 어떤 역할로 특징지어질 수 있는 반면, 디바이스 액터는 어떤 역할 중심적이기 보다는 시스템의 특정 기능을 유발시키는 요소로서의 의미가 강하다. 이는 작성된 모델을 이해하기 힘들게 만드는 원인이 된다. 둘째, 액터의 성격 차이는 이와 관련된 유즈케이스의 수준 차이로 나타난다. 사람 혹은 시스템 액터(System Actor)가 관련된 유즈케이스는 시스

템의 일련의 동들로 정의되어지는 반면, 디바이스 액터(Device Actor)가 관련된 유즈케이스는 시스템의 단편적인 활동으로 정의되어지는 경우가 많다. 수준이 상이한 유즈케이스가 혼재하는 유즈케이스 모델을 통해서 시스템의 형태를 가늠하기는 쉽지 않다. 또 유즈케이스의 수준 차이에 따른 구현과정의 상이함으로 인해 개발의 일관성을 잃을 수도 있다. 이는 개발 프로젝트를 관리할 수 없게 만드는 원인이 된다. 마지막으로 액터의 추출 기준 및 액터들 간의 관계가 명확하지 않은 점을 들 수 있다.

3. 관련연구

본 장에서는 임베디드 시스템을 위한 개발 방법론인 COMET과 OCTOPUS에 대해서 살펴 본다. 본 연구가 요구사항 분석 과정에 초점을 맞추고 있기 때문에, 여기에서는 개발 방법론의 요구사항 모델링 과정 및 분석 모델링 과정을 살펴본다.

3.1 COMET

COMET (Concurrent Object Modeling and architectural design mEThod)[1]은 유즈케이스 개념을 기반으로 하는 객체지향 개발 방법론이다. 정의에서 볼 수 있듯이, 대부분의 개발 과정은 유즈케이스 중심으로 구성된다.

요구사항 모델링 과정에서는 시스템의 기능적 요구사항이 유즈케이스 모델을 통해서 기술된다. 이 개발 방법론에서 유즈케이스 모델의 개발은 액터의 추출에서부터 시작된다. 액터는 주로 입/출력 디바이스를 통해서 시스템과 상호작용 하는 사람과 개발 시스템과 연관된 외부 시스템 외에 외부 입/출력 디바이스 및 타이머를 포함한다. 외부 입/출력 디바이스는 사람과 상호 작용 없이 자율적으로 시스템의 기능을 활성화시키는 경우에 액터로 추출되어질 수 있다. 시스템으로부터 정기적인 정보가 산출되어야 할 필요가 있을 경우에는, 타이머가 액터로 추출되어 질 수 있다. 이 모델에서는 액터를 통해서 시스템의 경계를 결정하고, 유즈케이스를 추출하고 있다. 그러나 이 방법론에서 액터의 수준은 일정하지 않다. 디바이스를 통해서 시스템과 상호 작용을 하는 경우, 디바이스가 아닌 디바이스를 이용하는 개체가 액터가 되는 반면, 다른 경우엔 디바이스 자체가 액터가 된다. 결과적으로 유즈케이스의 수준 또한 일정치 못하다. 이를 통해서는 시스템의 경계도 정해질 수 없다. 하드웨어를 포함한 시스템 경계와 소프트웨어만을 시스템으로 보는 두 견해 모두 사용되고 있기 때문이다.

분석 모델링 과정에서 유즈케이스는 관련된 객체(Object)와 이들 간의 관계로 표현되어 진다. 이런 과정을 유즈케이스의 구현(Use Case Realization)이라고 부르며, 최종적으로 동적 모델(Dynamic Model)을 통해서 기술된다. 유즈케이스의 구현을 위해서는 유즈케이스에 관련되는 객체의 추출이 우선되어야 한다. 유즈케이스에 관련되는 객체는 시스템 내부적인 것은 물론 외부적인 것도 포함하며, COMET에서는 각각의 객체를 추출하기위한 모델링 과정을 두고 있다. 우선 정적 모델링(Static Modeling) 과정은 문제 영역(Problem Domain)의 주요한 클래스(Class)를 추출하는 것을 목적으로 한다. 이 과정에서는 시스템 내부의 정보를 중심으로 추출되는 엔터티 클래스(Entity Class)와 물리적으로 존재하는 개체를 모델링한 피지컬 클래스(Physical Class)에 초점을 맞추고 있다. 피지컬 클래스는 물리적인 디바이스, 사람, 외부 시스템, 타이머를 포함한다. 클래스들이 추출된 후 시스템의 외부적인 측면은 시스템 컨텍스트 모델(System Context Model)을 통해 좀더 구체적으로 기술된다. 시스템 컨텍스트 모델은 시스템과 상호 작용하는 외부 클래스(External Class)의 구조를 보여준다. 외부 클래스는 피지컬 클래스와 유즈케이스 모델의 액터로부터 정의된다. 외부 요소들이 정적 모델링과 시스템 컨텍스트 모델링 과정에서 추출되는 반면 시스템 내부의 요소들은 객체의 구조화(Object Structuring) 과정을 통해서 추출되어진다.

유즈케이스 모델링 과정에서의 시스템 경계 설정과는 달리 분석 모델링 과정에서의 시스템 경계는 소프트웨어만으로 한정되어 진다. 분석 모델링 과정에서는 시스템 경계 설정의 변화로 유즈케이스 모델에서는 정의되지 못한 시스템 외부 객체를 찾는 작업이 수행되며, 이후 유즈케이이스 모델의 액터와 시스템 컨텍스트 모델링 과정과 동적 모델링 과정에서 추가된 외부 객체의 관계를 설정하는 작업을 수행하게 된다. 이는 시스템의 외부적인 측면을 다루는 유즈케이스 모델에서 고려될 수 있는 내용임에도 불구하고, 현재 유즈케이스 모델이 그 내용을 충분히 반영하지 못하고 있기 때문에 중복적이고 복잡한 개발 과정을 통해 이를 보충하고 있다. 또한 시스템의 외부 요소와의 관계를 설정하기 위한 일련의 작업이 여러 개발 과정으로 분산되어 처리되기 때문에 작업의 일관성이 훼손될 수 있으며, 관계 설정에 필요한 구성요소를 빠뜨리는 실수를 범할 가능성이 높다.

3.2 OCTOPUS

OCTOPUS[2]는 임베디드 시스템을 위한 객체지향 개발 방법론으로, OMT[8]와 Fusion method[7]를 기반으로 한다. 방법론의 요구사항 명세 및 서브시스템 분석 과정에 대해 자세히 살펴본다.

요구사항 명세(Requirements Specification)는 유즈케이스 모델과 시스템 컨텍스트 모델로 이루어 진다. 유즈케이스 모델은 Jacobson의 모델[5]을 기본으로, 액터가 명시적으로 언급되지 않는 시스템의 자율적인 활동(Autonomous Activities)을 지원할 수 있게 확장된 형태이다. 시스템의 자율적인 활동은 시스템의 디바이스로부터 시작되어 지는 활동을 말하며, 이 활동의 명확한 액터가 없다는 의미는 그 시스템 디바이스를 시스템의 외부 개체로 보지 않고, 시스템 경계 내에 포함시킨다는 것이다. 이 모델에서 액터는 Jacobson의 유즈케이스 모델에서처럼 사람과 외부의 시스템으로 구분되어 진다. 유즈케이스 모델에서 이 외의 액터는 명시되지 않는다. OCTOPUS에서 유즈케이스 모델은 클래스 다이어그램(Class Diagram)으로 표현되며, 유즈케이스 간의 상관 관계를 나타낼 수 있다. 유즈케이스 모델을 작성한 후, 이 모델의 정보를 바탕으로 시스템 컨텍스트 모델이 작성된다. 시스템 컨텍스트 모델에서의 시스템은 소프트웨어만을 포함하는 것으로 사용된다. 이 모델에서 시스템의 환경 요소로는 유즈케이스 모델의 액터와 시스템의 입/출력 디바이스를 위주로 고려되어 진다.

앞서 언급했듯이 임베디드 시스템 도메인에서는 시스템이 하드웨어와 소프트웨어로 구성되어 있으며, 일반적으로 시스템 사용자는 하드웨어를 통해 소프트웨어와 상호 작용하므로 이들 간의 관계가 명확하게 기술 되어야 시스템의 이해를 높일 수 있고, 더 나아가 유즈케이스의 구현에 체계적이고 쉽게 접근할 수 있다. OCTOPUS에서는 시스템 외부 요소를 표현하기 위해 두 가지 모델을 사용하고 있지만, 이들 외부 요소와 시스템의 관계를 명확히 표현하지 못하고 있다. 유즈케이스 모델에서는 하드웨어를 포함한 시스템과 외부 요소와의 관계를 표현하고 있으며, 시스템은 유즈케이스들로 표현된다. 시스템 컨텍스트 모델에서는 시스템의 경계가 소프트웨어만으로 한정되며, 시스템의 디바이스를 포함한 외부 요소만을 보여준다. 실제 기술 되어야 할 액터와 시스템의 디바이스의 관계, 디바이스와 유즈케이스와의 관계는 나타나 있지 않다.

4. 중첩된 구조의 유즈케이스 모델

본 장에서는 지금까지 살펴본 임베디드 시스템의 특징을 잘 반영할 수 있는 중첩된 구조의 유즈케이스 모델을 제시한다. 또한 제시된 유즈케이스 모델을 이용한 모델링 절차를 제시하고, 모델의 구성요소를 추출하는 효율적인 방법을 제시한다.

4.1 Nested use case model

본 논문에서 제안하는 중첩된 유즈케이스 모델([그림 1])은 다음과 같은 특징을 갖는다.

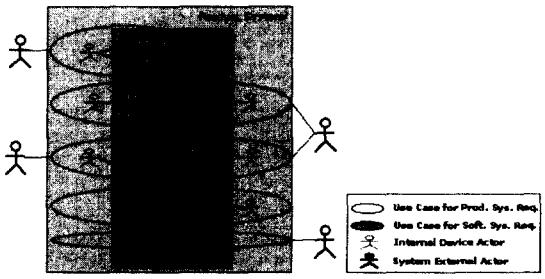
첫째, 시스템의 경계는 제품 시스템 경계(Product System Boundary)와 소프트웨어 시스템 경계(Software System Boundary)로 구분되어 진다. 소프트웨어 시스템 경계가 소프트웨어만을 시스템에 포함하는 경계를 설정하고 있는 반면, 제품 시스템 경계는 하드웨어와 소프트웨어 모두를 시스템에 포함시킨다. 시스템의 경계에 따라 시스템의 요구사항을 제품 시스템의 요구 사항과 소프트웨어 시스템의 요구사항으로 나누어 접근할 수 있다.

둘째, 액터는 시스템의 경계 구분에 따라 두 가지로 나뉘어 진다. 즉, 제품 시스템 경계 밖에서 시스템과 상호 작용을 하는 사람과 외부 시스템을 포함하는 시스템 외부 액터 (System External Actor), 제품의 시스템 내에 존재하는 하드웨어 요소-입/출력 디바이스(Input/Output Device)와 타이머(Timer)-에 해당하는 내부 디바이스 액터(Internal Device Actor)로 구분되어 진다. 두 종류의 액터들간 상호 관계는 이들 액터들과 관련 있는 유즈케이스간 계층 구조를 통해 나타난다.

셋째, 유즈케이스 또한 관련되는 액터에 따라 두 가지로 구분되어 진다. 우선, 제품 시스템을 구성하는 시스템 외부 액터와 관련된 유즈케이스가 존재하며, 내부 디바이스 액터와 관련되어 소프트웨어 시스템의 기능을 나타내는 유즈케이스가 존재하게 된다. 제품수준의 유즈케이스는 보통 이 유즈케이스의 사용자가 상호 작용하는 디바이스(디바이스 액터)와 이 디바이스가 관련된 소프트웨어 시스템 수준의 유즈케이스로 구성된다.

이 모델이 기존의 모델과 비교해 얻을 수 있는 장점은 다음과 같

다. 첫째, 명확한 시스템 경계는 시스템의 환경을 포함한 시스템을 이해하는 데 도움을 주며, 관련된 모델링 요소들은 이러한 외부 환경을 반영하는 모델의 작성을 가능하게 한다. 둘째, 제품의 하드웨어 측면의 개발이 선행된 후 소프트웨어 개발을 하게 되는 현재의 임베디드 시스템 개발 과정에도 잘 부합될 수 있다. 셋째, 분석 모델링 과정의 유즈케이스 구현이 용이해 진다. 이유는 이 모델에서는 외부의 요소로부터 소프트웨어 시스템 내부의 기능적 요소까지 단계적으로 명확한 관계를 표현하고 있고, 시스템의 내부 디바이스 액터는 소프트웨어 시스템의 인터페이스 클래스와 일대일 대응 관계를 갖게 되며, 소프트웨어 시스템 수준의 유즈케이스는 제품 수준의 유즈케이스를 처리하기 쉬운(Manageable) 단위로 나누어 주고 있기 때문이다.



[그림 1] nested use case model for embedded system

제시된 모델은 기존의 개발 방법론에서 제시하는 유즈케이스 모델에 비해 비교적 하위 수준의 내용을 다루고 있다. 이로 인해 유즈케이스의 구현에 체계적으로 접근할 수 있는 이점을 얻을 수 있는 반면, 유즈케이스 모델의 복잡성이 증가할 수 있다. 다음 장에서는 제시된 유즈케이스 모델을 이용한 모델링 절차를 살펴보고, 디바이스 액터와 소프트웨어 시스템 수준의 유즈케이스를 효율적으로 처리할 수 있는 방법을 제시한다.

4.2. 유즈케이스 모델링의 절차 및 구성요소 추출 방법

임베디드 시스템을 위한 중첩된 구조의 유즈케이스 모델을 작성하는 절차는 다음과 같다.

- **단계1** : 제품 시스템(Product System)의 경계를 설정한다
- **단계2** : 제품 시스템의 사용자 측면을 고려하여 시스템 외부 액터(System External Actor)를 추출하고, 이를 중심으로 제품 수준의 유즈케이스(Product-level Use Case)를 찾아낸다
- **단계3** : 소프트웨어 시스템(Software System)의 경계를 설정.
- **단계4** : 추출된 제품 수준의 유즈케이스를 기준으로, 제품 시스템 경계 내에 위치하고, 소프트웨어 시스템 외부에 위치하는 입/출력 디바이스(I/O Device)를 나열한다. 필요에 따라 타이머(Timer)도 추가될 수 있다.
- **단계5** : 소프트웨어 시스템 수준의 유즈케이스를 활성화시킬 수 있는 액티브 디바이스 액터(Active Device Actor)를 찾고, 이를 중심으로 소프트웨어 시스템 수준의 유즈케이스를 추출한다.

위의 절차는 Ivar Jacobson 의 유즈케이스 모델링 과정[5]을 따르고 있으며, 단계4는 제품의 시스템 명세서를 통해 어렵지 않게 수행될 수 있다. 단계5를 위해서는, 디바이스 액터들을 특징별로 분류하고, 그 특징에 따라 액티브 디바이스 액터를 추출하는 방법을 제시한다. 또 각각의 액티브 디바이스 액터로부터 활성화될 수 있는 소프트웨어 시스템 수준의 유즈케이스의 특징을 보여줌으로써 유즈케이스의 추출이 보다 쉽게 이루어질 수 있도록 돕는다.

기존의 임베디드 시스템 개발에서 소프트웨어 개발 전 대략의 하드웨어 정보는 이미 존재하는 것이 일반적이다. 그런 하드웨어 정보를 바탕으로, 하드웨어 디바이스는 자료가 입/출력되는 방식에 따라, interrupt-driven I/O device, poll-based I/O device, on demand I/O device로 구분되어질 수 있다. Interrupt-driven I/O device는 외부로부터의 자료 입력이 도착하자 마자 혹은 출력이 종료하자 마자 시스템의 어떤 행위를 유발시키는 특성을 가지며, poll-based I/O device는 시스템이 주기적으로 디바이스를 탐지하여 입/출력에 변화가 있을 경우 시스템의 행위를 유발시키는 특성을 갖는다. 두 종류의 디바이스가 능동적으로 시스템의 행위를 유발시키는 반면, on demand I/O device는 시스템 내의 요구에 의하여 동작하는 수동적인 디바이스로 분류된다. 결국, interrupt-driven I/O device와 poll-based I/O device 만이 액티브 액터로 분류되어질 수 있다. 이는 액티브 디바이스 액터가 되기 위한 기본적인 특성이라 할 수 있으며, 액티브 디바이스 액터는 디바이스가 시스템에서 사용되는 특성을 기준으로 사람과 상호 작용하는 입력 디바이스, 시스템의 자율적인 활동을 유발시키는 입력 센서(Sensor), 자료의 출력이 종료된 후 시스템의 다른 행위가 유발 되어야 할 때 출력을 담당했던 출력 디바이스, 그리고 타이머(Timer)로 구분된다.

액티브 디바이스 액터 추출 후의 작업은 이를 중심으로 한 소프트웨어 시스템 수준의 유즈케이스를 찾아내는 것이다. 위에서 언급한 액티브 디바이스 액터의 후보들을 기준으로 추출 가능한 유즈케이스에 대한 특징은 다음과 같다. 첫째, 사람과 상호 작용하는 입력 디바이스의 경우 사람이 입력하는 자료를 필요로 하는 시스템의 활동을 유즈케이스로 고려할 수 있다. 둘째, 센서의 경우 센서를 통해서 감지하려고 하는 대상고 관련된, 혹은 감지 결과 얻는 자료를 필요로 하는 활동을 유즈케이스로 고려할 수 있다. 셋째, 출력 후 시스템의 다른 활동이 유발될 때의 출력 디바이스의 경우 수행되어야 하는 그 활동이 유즈케이스로 고려될 수 있다. 넷째, 타이머의 경우 주기적인 시스템의 활동, 일정 시간 소모 후 수행되어야 하는 활동, 특정 시간에 발생되어야 하는 활동, 시스템 주위 환경을 감시하는 활동을 유즈케이스로 고려할 수 있다.

5. 결론 및 향후 연구

본 논문에서는 임베디드 시스템의 특성을 반영한 중첩된 구조의 유즈케이스 모델을 제시하고 있다. 시스템의 경계를 제품 시스템 경계와 소프트웨어 시스템 경계로 명확히 구분하였으며, 각각의 시스템 경계에 맞게 액터는 시스템 외부 액터와 내부 디바이스 액터로, 유즈케이스는 제품 시스템 수준의 유즈케이스와 소프트웨어 시스템 수준의 유즈케이스를 정의하고, 이들간의 상관 관계를 중첩된 형태로 표현하였다. 또 본 논문에서는 제시된 유즈케이스 모델을 작성하기 위한 절차를 제시하고 있으며, 효율적인 모델링을 위해 내부 디바이스 액터 및 소프트웨어 시스템 수준의 유즈케이스를 추출하기 위한 방법을 제시하고 있다.

현 단계에서의 지원 방법은 가이드 라인 정도이며, 앞으로 이에 대한 충분한 연구를 필요로 하고 있다. 또한 아직까지 제시된 유즈케이스 모델의 평가가 이루어지지 못하고 있다. 기존의 모델들과 비교/평가하기 위한 기준을 제시하고, 기준에 따라 실제 비교/평가가 이루어져야 한다.

6. 참고문헌

[1] Hassan Gomaa, *Designing Concurrent, Distributed, and Real-Time Applications with UML*, Addison Wesley, 2000

[2] Maher Awad, Juha Kuusela, and Jurgen Ziegler, *Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT and Fusion*, Prentice Hall, 1996

[3] Geri Schneider, and Jason P. Winters, *Applying Use Cases : A Practical Guide*, Addison Wesley, 1998

[4] Martin Fowler, *UML Distilled Second Edition : A Brief Guide to The Standard Object Modeling Language*, Addison Wesley, 2000

[5] Ivar Jacobson, *Object-oriented Software Engineering*, Addison Wesley, 1992

[6] Ivar Jacobson, Maria Ericsson, and Agneta Jacobson, *The Object Advantage : Business Process Reengineering with Object technology*, Addison Wesley, 1995

[7] Derek Coleman, et al., *Object-Oriented Development - The Fusion Method*, Prentice Hall, 1993

[8]James Rumbaugh, et al., *Object-Oriented Modeling and Design*, Prentice Hall, 1991

[9] K.G. van den Berg and A.J.H Simons, "Control-Flow Semantics of Use Cases in UML", Information and Software Technology 41 (1999) 651-659

[10] Friedrich Steimann, "On The Representation of Roles in Object-Oriented and Conceptual Modeling", Data & Knowledge Engineering 35 (2000) 83-106

[11] Alistair Cockburn and Martin Fowler, "Question Time! about Use Cases", OOPSLA '98

[12] Kevin L. Mills, "Automated Generation of Concurrent Designs for Real-Time Software", PhD Dissertation, George Mason University, Fall, 1995

[13] Ivar Jacobson, Grady Booch and James Rumbaugh, *The Unified Software Development Process*, Addison Wesley, 1998