187-10 서울시 구로구 구로동 코오롱 사이언스 밸리 II 812호 한국인터넷방송통신·TV학회 / Tel. 02-2025-4777 / Fax. 02-2025-4790

제6권 제1호 2006
ISSN 1738-4281
1

# 한국인터넷방송통신·TV학회논문지

The Journal of The Institute of Webcasting, Internet Television and Telecommunication
VOL. 6 NO. 1

## 논 문



사단법인 한국인터넷방송통신·TV학회
The Institute of Webcasting, Internet Television and Telecommunication
(http://www.iwit.or.kr)

# 차 례

2006년 3월 제6권 제1호

## 논 문

The Journal of The Institute of
Webcasting, Internet Television and Telecommunication
VOL. 6 No. 1, March 2006

# CONTENTS

## Technical Papers

논문 2006-1-9

# MDA 기반의 임베디드 소프트웨어 설계에 관한 연구

# A Study on Design for Embedded S/W based on Model Driven Architecture

김동호*, 김우열*, 김영철*

Dong-Ho Kim*, Woo-Yeol Kim*, Young-Chul Kim*

요 약

이 논문은 기존의 MDA방법을 임베디드 소프웨어에 개발에 채택하고자 제안한다. 기존 임베디드 소프트웨어의 개발에 있어서 하드웨어, 운영체제, 프로그래밍언어, 플랫폼에 종속적인 것이 가장 큰 문제이다. 이 부분을 해결하기 위해 플랫폼에 독립적인 소프트웨어 설계가 가능한 MDA(Model Driven Architecture)방법을 사용하였으며 기존의 임베디드 소프트웨어 개발이 코드생성 에 종속되어 개발의 어려움이 많았다. 이 문제를 해결하기 위해 xUML(Executable UML)을 사용하여 실행 가능한 모델을 생성하여 개발자가 모델 중심의 개발이 가능하도록 하였으며, 임베디드 소프트웨어 개발 단계로서 확장되고 개선된 V-Model을 제안하였다. 임베디드 소프트웨어에 대한 상호 운용성을 향상 시킨다.

Abstract

This paper suggests to adopt MDA(Model Driven Architecture) into embedded s/w development. On an embedded software development there is a big problem which is dependent on H/Ws, operating systems, programming languages, and platforms. So, we cannot reuse any code on the heterogenous embedded systems. To solve this problem, we attempt to adopt a high-level software process development, that is, MDA with target independent and target dependent mechanism for reusability of resources such as design and source code. Also, we use executable UML instead of UML 1.x, embedded UML, and UML2.0. As a result, we can produce an executable software through model-oriented development, that is, conversion from target independent model to target specific model. Also, we suggest a refined V-model for the embedded software development. It will enhance interoperability of embedded software.

Keyword: 임베디드, MDA(Model Driven Architecture), xUML(Executable UML)

## I. 서 론

최근의 IT 기술은 마이크로프로세서의 가격이 낮아지고 소형화 및 고성능화가 진행됨에 따라 제품 경쟁력의 핵심이 하드웨어 생산 기술에서 소프트웨어 최적화 기술로 이동하는 변혁기를 맞이하여 임베디드 소프트웨어가 탑재된 상품의 가치가 하드웨어보다는 소프트웨어에 의해 좌우되

* 홍익대학교 컴퓨터정보통신 소프트웨어공학연구실
접수일자: 2006.1.13, 수정완료일자: 2006.3.15

는 기술집약적 고부가가치 산업으로 발전하고 있다. 초창기 임베디드 소프트웨어는 간단한 제어 프로그램만으로 산업용 기기를 제어하는데 그쳤으나, 최근에는 멀티미디어 처리와 같은 점차 복잡한 기능을 위해 멀티태스크 및 네트워크 기능을 제공하는 임베디드 OS(Operating System)를 이용한다.

이렇게 복잡한 기능을 제공하는 임베디드 소프트웨어를 좀 더 빠르고 효율적이며 안정적인 개발방안에 관해 많은 연구들이 진행되고 있다. 그러나 하드웨어에 종속되는 임베디드 소프트웨어의 특성과 소스 코드 중심의 개발이 진행됨으로 인해 소프트웨어의 재사용의 문제를 갖고 있다. 그래서 하드웨어와 운영체제, 그리고 프로그래밍 언어에 독립적인 설계를 하고자 MDA (Model Driven Architecture)방법을 임베디드 소프트웨어 개발에 적용하고자 한다. MDA는 설계 환경이나 시스템에 종속적이지 않고 소프트웨어를 개발하기 위한 기술이다[1]. 그러나 기존의 MDA는 커다란 시스템을 구축하는데 적합한 아키텍처이다[2]. 그리고 MDA만으로는 임베디드 소프트웨어의 타겟 독립적인 설계가 가능하지 않다. 그래서 본 논문에서는 임베디드 소프트웨어를 기술하는 정형화 된 기술 언어로서 기존의 UML (Unified Modeling Language) 대신 실행 가능한 xUML을 채택하였다.

본 논문에서는 임베디드 소프트웨어의 모델링을 기술하는데 적합한 언어인 xUML(Executable UML)을 사용하고자 한다. 그리고 이를 위해 개발 단계로서 종전의 Multiple V-Model의 단점을 보완한 MDA 기반의 Multiple V-Model 및 개발 단계를 제안한다. 이 개발 단계를 통해 타겟에 독립적인 임베디드 소프트웨어 개발을 가능하게 함으로써 소프트웨어의 재사용을 도와주며 그 결과로 생산 비용의 절감을 가져올 수 있다. 본 논문의 구성은 다음과 같다. 2장에서는 관련연구로써 MDA의 구성요소 및 특징과 MDA 개발 프로세스 대해 알아본다. 그리고 xUML의 특징에 대해 자세히 설명을 한다. 3장에서는 기존 임베디드 소프트웨어 모델링 방법인 Multiple V-Model에 대해 알아본다. 4장에서는 xUML을 사용한 MDA기반의 임베디드 소프트웨어 디자인에 대한 설명을 위해 임베디드 소프트웨어 모델링 방안을 그 사례로 언급한다. 마지막으로 5장에서는 결론 및 향후 연구방향에 대해 기술한다.

## II. 관 련 연 구

### 1. MDA(Model Driven Architecture)

소프트웨어가 점점 더 복잡해지고 대형화가 이루어짐에 따라 이러한 문제를 해결하고자 객체지향 기술이 도입되었다. 그러나 소프트웨어의 재사용성과 유지보수성을 향상시키기 위해 객체보다 더 큰 단위인 컴포넌트를 재사용하기 위한 컴포넌트 개발 방법론이 대두되었다[3]. 그러나 개발 플랫폼의 다양화로 인하여 최종 구현에 있어 많은 어려움이 있으며 시스템 통합 면에서도 많이 어렵다. 그래서 모델 수준에서의 재사용을 하고자 Model Driven Development (MDD)가 대두되었고, OMG에서 발표한 CORBA(Common Object Request Broker Architecture)는 IT 업계에 객체 기술의 도입하여, 이제는 기업 상호 운용성의 표준으로 자리 잡았다. 하지만 기업에서 선택은 CORBA 하나뿐이 아니라 이종의 플랫폼도 가능하다. MDA는 설계 환경이나 시스템에 종속적이지 않고 소프트웨어를 개발하기 위한 기술이다[3]. MDA는 변환법칙에 의해서 툴 혹은 사람의 손을 통해 플랫폼 독립적인 모델(Platform Independent Model)을 플랫폼 종속적인 모델(Platform Specific Model)로 변환하고, 이 플랫폼 종속적인 모델을 구현소스로 변환하는 사상이다. 이로 인해서 MDA는 플랫폼에 독립적인 표준화 기술로 새롭게 대두되고 있다[4].

가. MDA 개발 프로세스

원래 MDA는 임베디드 소프트웨어를 위한 것은 아니다. MDA는 기존에 OMG가 발표한 여러 가지 표준들을 기반에 두고 서는 특정 플랫폼에 귀속되지 않은 플랫폼 독립적인 모델을 중심으로 개발과 시스템 통합 및 연동을 수행하는 프레임워크이다. MDA개발 프로세스에서는 플랫폼에 독립적인 PIM을 생성하고, PIM에서는 플랫폼에 종속적인 여러 개의 PSM으로 변환하며 최종적으로 코드로 변환한다. 모델간의 변환은 기존의 전통적인 개발 프로세스와 다를 것 없지만 MDA에서는 모델이 UML프로파일로 작성되기 때문에, PIM에서 PSM으로 변환이 모두 매핑 규칙에 의해 자동화 되므로 개발자의 노력을 최소화한다[5].

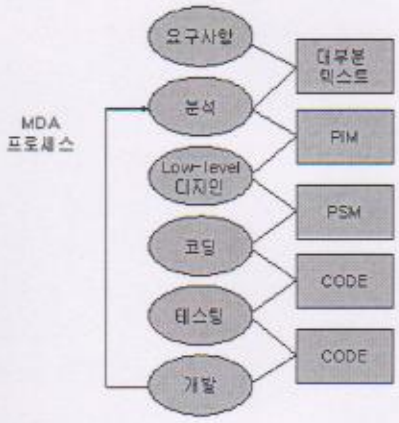


그림 1. MDA 개발 프로세스
Fig. 1. MDA development process

MDA개발 프로세스에서는 플랫폼에 독립적인 정보를 가지고 있는 PIM에 주로 초점을 맞추어 개발한다. 그림 1은 MDA개발 프로세스이다. 먼저 PIM을 위한 UML Profile을 이용해서 functionality와 behavior에 초점을 맞춘 PIM을 모델링한다. 그리고 다음으로 PIM을 목표하고 있는 플랫폼의 PSM으로 변환하는데, 이 때 필요한 것이 PSM 작성을 위한 UML Profile과 PIM과 PSM 사이의 매핑 규칙에 해당되는 모델 변환 규칙이다. 다음으로 Rational Rose와 같은 CASE 툴을 이용해서 PSM에서 코드를 얻어낸 다음 세부 코딩을 하는 과정을 거친다. Process에서 가장 중점을 두어야 할 부분은 PIM을 PSM으로 바꾸는 부분이다[5].

2. xUML(Executable UML)

Executable UML은 UML(Unified Modeling Language) 표기법에 실행에 관련된 개념(executable semantics)들과 시간에 관련된 규칙(timing rules)을 더한 것이다. 그림 2는 xUML의 기본 개념을 나타낸다. Executable UML을 이용하면, 클래스와 상태와 액션 모델로 이루어진, Executable 시스템 스펙을 만들 수 있다. Executable 시스템 스펙은 하나의 완전한 프로그램처럼 실행된다. 기존 스펙과 달리 Executable 스펙은 실행과 테스트, 디버깅이 가능하며, 스펙(모델)에 대한 테스트가 끝나면, 이를 타겟 코드(target code)로 변환(translate)할 수 있다.

Executable UML은 리얼타임 시스템을 위한 코드를 지원한다. 타겟 코드는 단일 프로세서 환경에서 실행될 수 있고, 여러 개의 프로세서들로 이루어진 프로세서 네트워크 환경에서 실행될 수도 있다. 기존의 UML과 달리, Executable 모델은 시간에 관련된 문제와 동기화에 관련된 문제, 그리고 자원의 획득과 이용에 관련된 문제들을 자세히 서술한다[6].


그림 2. xUML의 기본 원리
Fig. 2. Basis of xUML

가. 기존 UML과 xUML과의 비교

표 1을 통해 기존의 UML과 xUML을 비교하였다. xUML을 기존의 UML과 비교하여 보았을 때

두드러지게 나타나는 부분은 실행과 시간관련 그리고 테스트 및 분명성, 배치방식 부분이다. 실행 부분은 모델을 바로 실행 가능함으로써 빠른 개발 및 검증이 가능하고 시간 관련 부분은 임베디드 시스템 특성상 실시간 시스템의 형태를 지니고 있기 때문에 시간 관련 부분을 고려하였으며, 모델이 실제 프로그램처럼 실행 가능하기 때문에 객관적인 테스트가 가능하다 또한 모델을 다양한 방식으로 배치(deployment)할 수 있으므로 모델이 실행되는 환경에 독립적이다.

표 1. 기존 UML과 Executable UML 비교
Table1. Comparison with the existing UML and xUML

| 구분 | UML 1.X | UML 2.0 | Embedded UML | Executable UML |
|---|---|---|---|---|
| 실행 가능 여부 | 실행 안됨 | 실행안됨 | 실행안됨 | 실행 |
| 시간 관련 | 고려안함 | 고려함 | 고려함 | 고려함 |
| 분명성 | 모호 | 분명 | 모호 | 분명 |
| 테스트 가능 여부 | 불가능 | 불가능 | 불가능 | 가능 |
| 배치방식과의 독립 여부 | 고려안함 | 독립 | 고려안함 | 독립 |
| N-ary association | 가능 | 가능 | 가능 | 단지 3-way 허가 |
| 포함관계 (Aggregation & Composition) | 사용 | 사용 | 사용 | 사용안함 |
| OCL(Spec. Language) | 사용안함 | 사용 | 사용 | 사용 |
| 참여개수 | 어떤 수라도 사용 | 0,1,* | 어떤 수라도 사용 | 0, 1, * |

## 3. Embedded 모델링 방법

### 가. Multiple V-Model

Multiple V-Model은 기존 임베디드 테스팅 모델이다. Multiple V-model은 잘 알려진 V-mode을 기반으로 한다. 원칙적으로 각 산출형태(모델, 프로토타입, 최종 산출물)는 디자인과, 빌드 그리고 테스트활동을 포함하는 완전한 V개발 주기를 따른다[7].

Multiple V-모델은 임베디드 시스템 개발에 적절한 모형으로 기존의 V 모델을 모델, 프로토타입, 최종 제품에 관한 세 가지로 나누어 연결한 모델이다. Multiple V-모델의 처음 단계인 모델 단계는 PC를 통해 요구된 시스템의 행위를 모의 실험하는 단계이다. 그리고 이러한 모델 단계가 적합하다고 판정되면 프로토 타입 단계에서는 모델로부터 소스코드를 생성하고 실험용 하드웨어에 소스코드를 삽입하여 점차적으로 실제 하드웨어로 전환해 가면서 최종 제품으로 개발해 간다. 그리고 각 단계는 디자인, 구현, 테스트의 순차적인 개발 과정이 포함되어 있다. 그것은 그림 3과 같다.

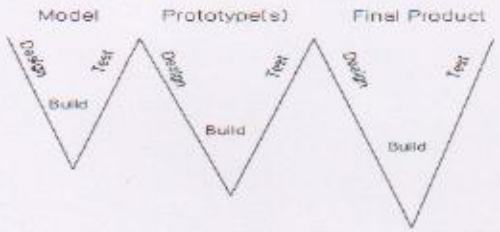


그림 3. Multiple V-Model 개발 주기
Fig.3. Multiple V-Model life cycle

Multiple V-모델은 임베디드 시스템 개발하기에 적합하다. 왜냐하면 앞서 설명한 바와 같이 Multiple V-모델은 임베디드 시스템의 개발 단계인 모델, 프로토타입, 최종 제품과 동일하게 세 단계로 구분하여 각 단계의 특성에 따라 테스트 디자인 기법이나 테스트 단계 등을 적용할 수 있기 때문이다. 또한 요구 사항에 변화나 오류로 인한 변경이 생길 경우 최종 산출물 보다는 프로토타입 단계, 그리고 프로토타입 단계보다는 모델 단계에서 변경함으로써 시간과 비용을 모두 감소시키는 효과도 기대할 수 있다[7].

# III. 임베디드 소프트웨어 모델링 방안

## 1. 개선된 V-Model

기존의 Multiple V-Model은 하나의 특정 타겟 도메인을 목적으로 모델링한다. 만약 같은 모델을 다른 타겟에 적용하려 했을 때 적용이 힘들다는 단점이 있다. 그래서 본 논문에서는 이런 문제점을 해결하고자 MDA의 타겟 독립적인 요소를 가져와서 적용하는 것을 제안한다. 이 모델은

TIM(Target Independent Model)을 통해 일반적인 모델링을 한 후 TSM(Target Specific Model)을 통해 특정 타겟에 명세적인 모델을 만든다. 이렇게 만들어진 TSM을 통해 각 타겟에 맞는 코드를 만들어 낸다. 기존의 Multiple V-Model과 비교하면 기존의 모델은 하나의 특정 도메인 타겟에서 모델을 재사용 할 수 있었으나 확장되고 개선된 V-Model은 여러 다양한 도메인의 타겟에서 모델을 재사용 할 수 있다는 장점이 있다. 다음 그림 4는 개선된 V-Model을 나타낸다. 그러나 그 변환 과정에 있어서 자동화 된 툴이 없다는 단점이 있다.

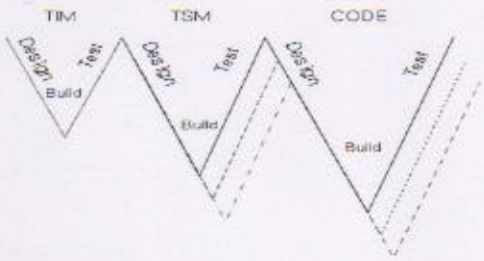


그림 4. 개선된 V-Model

Fig.4. Extended Revised V-Model

## 2. MDA 기반 임베디드 소프트웨어 설계 단계

본 논문에서는 또한 MDA 기반 임베디드 소프트웨어의 설계 단계를 제안한다. 전체 세 단계로 이루어졌는데, 첫 번째 단계에서는 타겟의 일반적인 모델링을 구축함으로서 타겟 독립적인 모델을 만들고 그 하위에 3개의 단계로 다시 나뉜다. 1-1 단계에서는 클래스를 명세화 하기 위한 클래스 다이어그램을 작성한다. 1-2단계에서는 객체의 행위를 표현하기 위한 상태 다이어그램을 작성한다. 1-3단계에서는 여러 객체들이 시간 경과에 따라 객체 상호간 교류 과정을 표현하고자 작성한다. 이 때 xUML을 사용하여 정형화되고 실행 가능한 모델 구축을 한다. 그 두 번째 단계로 특정 타겟에 명세적인 모델링을 구축을 한다. 이 때 모델의 명세적인 표현을 하기 위해 세 개의 하위 단계로 나뉜다. 앞의 첫 번째 단계과 마찬가지로 모델을 표현하기 위해 세 가지 다이어그램을 사용한다. 마지막 세 번째 단계로 코드 생성 부분이 있는데 본 논문에서는 xUML을 사용한 MDA기반의 임베디드 소프트웨어 디자인에 관한 부분을 제안하기 때문에 코드생성에 대해 자세한 설명은 하지 않고 사례 연구에서 간단한 스켈레톤(skeleton) 코드만을 제시한다. 임베디드 소프트웨어 설계 단계는 다음과 같다.

1-Step: TIM(Target Independent Model) 모델링 구축

1.1 xUML 기반의 클래스 다이어그램 작성

1.2 xUML 기반의 스테이트 다이어그램 작성

1.3 xUML 기반의 메시지 시퀀스 다이어그램 작성

2-Step: TSM(Target Specific Model) 모델링 구축

2.1 xUML 기반의 클래스 다이어그램 작성

2.2 xUML 기반의 스테이트 다이어그램 작성

2.3 xUML 기반의 메시지 시퀀스 다이어그램 작성

3-Step: Code 변환

## IV. 사 례 연 구

사례로서 전자렌지에 대해 설명한다. 이 사례를 통해 스켈레톤 코드를 발생시키는 부분에 대해 언급할 것이다. 우선 전자렌지는 그림 5와 같이 7개의 객체들 - Interface Panel, Door, Cooking Element, Timer, Thermal Sensor, Light, Rotating Base - 로 구성된다. Interface Panel은 버튼 등 인터페이스 부분을 객체화 하였다. Door는 전자레인지의 문을 객체화 한 것이고 Cooking Element는 Microwave를 발생하여 음식을 가열하는 부분을 객체화 하였고, Timer는 시간의 변화 부분에 대한 객체화, 그리고 Thermal Sensor는 온도의 변화를 감지하는 온도 센서를 객체화 하였다. Light

는 전자렌지 내부를 비추는 불을 객체화하였고 Rotating Base는 전자렌지 내부의 음식이 고루 조리 되도록 회전하는 회전판을 객체화 한 것이다.

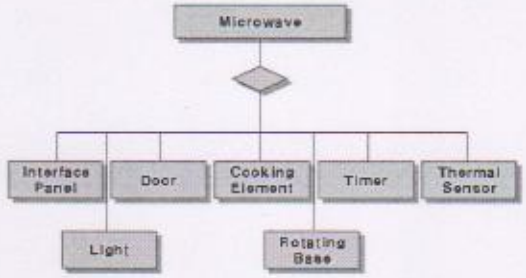


그림 5. 전자렌지의 객체들의 구성
Fig. 5. The structure of the object of microwave

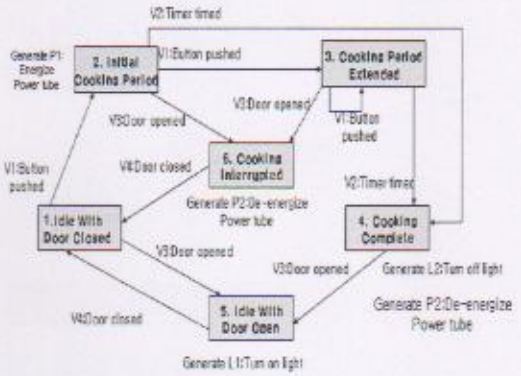


그림 6. 전자렌지의 스테이트 다이어그램
Fig.6. State diagram of microwave

그 다음 스텝으로 전자렌지 객체의 상태 변화를 분석하기 위해 스테이트 다이어그램을 그린다. 각 스테이트는 문이 닫히고 아이들 상태, 요리 주기의 초기화 상태, 요리 주기의 확장 상태, 요리완료 상태, 문이 열리고 아이들인 상태, 요리 도중 인터럽트 상태로 변화한다. 이것은 그림 6과 같다.

이 상태 다이어그램을 통해 코드를 발생 시킬 수 있으나 무시되거나 발생되지 않는 더미 스테이트 또한 코드화 된다. 이로 인해 소프트웨어의 품질이 낮아지는 문제가 있다. 그래서 이런 부분을 표 2와 같이 스테이트 테이블을 통해 더미 스테이트를 없애고 그 없앤 나머지 부분을 코드와 하여 코드 최적화를 한다.

표 2. 전자렌지 State Table에서의 공통부분 영역화
Table2. A commonness part area of a microwave state table

| | V1: Button Pused | V2: Timer Timed out | V3: Door Opened | V4: Door Closed |
|---|---|---|---|---|
| 1. Idle With Door Closed | 2 | Can't Happen[1] | 5 | Can't Happen[3] |
| 2. Initial Cooking Period | 3 | 4 | 6 | Can't Happen[3] |
| 3. Cooking Period extended | 3 | 4 | 6 | Can't Happen[3] |
| 4. Cooking complete | Event ignored | Can't Happen[1] | 5 | Can't Happen[3] |
| 5. Idle with door open | Event ignored | Can't Happen[1] | Can't Happen[2] | 1 |
| 6. Cooking interrupt | Event ignored | Can't Happen[1] | Can't Happen[2] | 1 |

[1] Timer is not running , [2] Door is already open
[3] Door is already closed

그림 7은 스테이트 테이블에서 더미 스테이트를 없앤 후 나온 CASE문이다. 각 스테이트별 발생하는 이벤트를 케이스문 형태의 코드로 만든 후 이것을 통해 각 타겟에 맞게 코드를 발생시킬 수 있다.

```
CASE
      •State 1: Event      Action/Code
      •State 2: Event      Action/Code
                :
      •State 6: Event      V3(State2 or State3)
                           Action: De-energize power tube
                                   Clear the timer
      •Can't happen: Error Code
      •Event / Ignore: No-Op
END CASE
```

그림 7. 전자렌지의 골격 코드
Fig.7. Skeleton code of a microwave

## V. 결 론 및 향 후 연 구

현재 임베디드 시스템은 하드웨어 기술이 발전

함에 따라 점차 변화되고 있다. 기존의 간단하고 제한된 기능만을 수행하던 시스템들이 사용자의 요구 수준이 높아져 감에 따라 고도의 기능이 요구되고 있기 때문이다. 사용자의 요구에 따라 빠른 임베디드 소프트웨어 개발을 하기 위해서는 개발 프로세스가 필요하며 그 프로세스에 맞는 자동화 된 틀이 필요하다.

이 논문에서는 임베디드 소프트웨어의 개발을 MDA기반으로 상호운용성을 향상시키고자 한다. 기존의 임베디드 소프트웨어 개발 시 하드웨어에 종속적인 면을 해결하기 위해 타겟 독립적인 개발 방법인 임베디드 소프트웨어 지향의 MDA (Model Driven Architecture)를 사용하여 해결하였으며 MDA에서 TIM(Target Independent Model)을 사용하여 타겟 독립적인 모델을 만들고 TIM으로부터 각 타겟에 맞게 TSM(Target Specific Model)을 만들어 타겟 명세적인 모델을 통한 코드 생성을 하는 방안을 제안하였다.

그리고 기존의 UML이 임베디드 소프트웨어를 기술하기에 적합하지 않기에 실행가능하며 정형화된 기술언어로 xUML을 사용하는 것을 제안하였다. 개발 프로세스는 기존의 Multiple V-Model이 갖는 특정 한 도메인에 종속적인 부분을 해결하기 위해 MDA방법을 Multiple V-Model에 적용하여 해결방안을 제안하였다. 따라서 빠르고 안정적인 임베디드 소프트웨어의 개발을 위해 xUML을 사용한 MDA기반의 설계를 함으로써 타겟에 독립적이고 실행 가능한 모델을 만들 수 있을 것이다. 이기법의 문제점은 각각의 모델에서 자동변환 및 코드 생성기의 구현이 필수적이라는 점이다.

향후 연구 과제로 각 모델들 사이에서의 변환을 자동화 된 틀을 통해 변환을 하여 정형화되고 빠른 개발이 가능하도록 하며 개발프로세스에 테스트 프로세스를 적용시켜 소프트웨어의 품질을 높이는 연구 및 코드의 자동생성에 관한 연구 및 모델의 재사용 및 저장을 위한 임베디드 소프트웨어 레파지토리에 관한 연구를 들 수 있다. 현재 본 논문에서 제안한 방법을 모델링할 수 있는 자동화 된 틀의 개발에 대한 연구가 진행 중이다.

## 참 고 문 헌


[1] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY, 1990.

[2] J. Poole, Model-Driven Architecture: Vision, Standards And Emerging Techchnologies, Position Paper Submitted to ECOOP 2001, 2001.

[3] Jon Siegel and OMG Staff Strategy Group, "Developing in OMG'S Model-Driven Architecture", Object Management Group White Paper, Nov. 2001.

[4] Michael Guttman, A Response to Steve Cook, MDA Journal, February 2004.

[5] A. Kleppe, J.Warmer, W.Bast, "MDA Explained: The Model Driven Architecture: Practice and Promise", Addison-Wiseley, 2003.

[6] Mellor, Stephen J., Marc J.Balcer, "Executable UML: A Foundation for Model-Driven Architecture". Boston: Addison-Wesley, 2002.

[7] Bart Broekman, Edwin Notenboom, "Testing Embedded Software" Addison-Wesley, 2003.


* 본 연구 논문은 정보통신연구진흥원 지원과제(B1220-0501-0321)로 수행되었습니다.

## 저 자 소 개


김동호(준회원)

- 2003년: 홍익대학교 컴퓨터정보통신 학사 졸업.
- 2005년: 홍익대학교 소프트웨어공학전공 석사 졸업.
- 2005년: 홍익대학교 소프트웨어공학 연구실 연구원

<관심분야 : 사용자 행위 분석 방법론, 임베디드 소프트웨어 개발 방법론, 모델 기반 아키텍처(MDA)>

김우열(준회원)

- 2004년: 홍익대학교 컴퓨터정보통신 학사 졸업.
- 2006년: 홍익대학교 소프트웨어공학전공 석사 졸업.
- 2006년: 현재 홍익대학교 소프트웨어공학 연구실 박사과정

<관심분야 : 사용자 행위 분석 방법론, 임베디드 소프트웨어 개발 방법론, 모델 기반 아키텍처(MDA)>

김영철(정회원)

- 2000년: 일리노이공대(박사)
- 2000년-2001년: LG 산전 중앙연구소 Embedded System 부장
- 2001년-현재: 홍익대학교 컴퓨터정보통신공학과 교수

<관심분야 : 사용자 행위 분석 방법론, Use Case 방법론 및 도구 개발, 정보 구조, 임베디드 소프트웨어 개발 방법론, 테스트/상호운용성 성숙도 모델(TMM/IMM)>