확장된 xUML을 사용한 MDA 기반 이종 임베디드 소프트웨어 컴포넌트 모델링에 관한 연구

김 우 열[†] · 김 영 철[††]

요 약

본 논문에서는 MDA(Model Driven Architecture) 기반의 임베디드 소프트웨어 컴포넌트 개발 방법을 소개한다. 이 방법은 이종의 임베디드 시스템에서 소프트웨어의 재사용성에 관한 문제점을 해결하고자 MDA기법을 임베디드 소프트웨어 개발에 적용한 것이다. 제안한 방법을 통해 하나의 메타 모델(Target Independent Model)을 각각의 다른 도메인에 맞는 타겟 종속적 모델(Target Specific Model)들을 만들고, 그에 따른 소스 코드(Target Dependent Code)를 개발하는 것이다. 이때 기 개발된 메타모델은 이종의 임베디드 시스템 개발에 재사용하려는 것이 목적이다. 우리는 이 방법에 따른 도구에 기존 xUML의 동적 모델링에서 표현되지 못하는 부분(병렬성, 실시간 등)을 보완하기 위해 확장하여 채택하였다. 확장된 xUML 노테이션을 기반으로 구현한 모델링 도구를 소개한다. 이는 임베디드 또는 병렬/실시간 소프트웨어의 모델링이 가능하다. 제안한 방법의 적용사례로서 이종 임베디드 시스템의 모델링을 통한 코드 개발을 보여준다.

키워드 : 통합 모델링 언어, 실행 가능 UML, 실시간, 병렬성, 모델 기반 아키텍처, 임베디드 시스템

A Study on Modeling Heterogeneous Embedded S/W Components based on Model Driven Architecture with Extended xUML

Woo Yeol Kim[†] · R. Young Chul Kim[††]

ABSTRACT

In this paper, we introduce MDA based Development Method for Embedded Software Component. This method applies MDA approach to solve problems about reusability of the heterogeneous embedded software system. With our proposed method, we produce 'Target Independent Meta Model' (TIM) which is transformed into 'Target Specific Model' (TSM) and generate 'Target Dependent Code' (TDC) via TSM. We would like to reuse a meta-model to develop heterogeneous embedded software systems. To achieve this mechanism, we extend xUML to solve unrepresented elements (such as real things about concurrency, and real time, etc) on dynamic modeling of the particular system. We introduce 'MDA based Embedded S/W Modeling Tool' with extended xUML. With this tool, we would like to do more easily modeling embedded or concurrent/real time s/w systems. It contains two examples of heterogeneous embedded systems which illustrate the proposed approach.

Key Words : Unified Modeling Language, xUML: Executable UML, Real-time, Concurrency, Model Driven Architecture, Embedded System

1. 서 론

최근 복잡한 기능을 제공하는 임베디드 소프트웨어를 좀 더 빠르고 효율적으로 개발하기 위해 소프트웨어를 재사용하는 방안에 대한 연구들이 진행되고 있다. 그러나 임베디드 소프트웨어는 시스템에 종속적인 특성을 지니고 소스 코드를 중심으로 개발이 진행되기 때문에 소프트웨어의 재사용이 어렵다[1].

MDA(Model Driven Architecture)[2, 3]는 플랫폼에 독립적인 메타모델을 설계한 후 필요한 기술 모델을 변경하여 그 모델을 통해 코드를 생성하려는 메커니즘이다. 이와 같은 메타 모델의 재사용으로 관련 코드의 생산성을 높일 수 있다. 그러나 반드시 자동화 도구가 필수적이라는 문제점이 있다.

우리는 임베디드 소프트웨어 개발에 MDA 메커니즘을 접목하여 메타모델의 재사용을 통해 개발시간을 단축시키고자 한다.

본 논문은 소프트웨어 개발 프로세스인 MDA를 임베디드 소프트웨어 개발에 적용하려는데 목적을 두고 있으며 확장된 xUML(Executable UML)[4-7, 12]이 적용된 자동화 도구를 제안한다. 그리고 이를 이용하여 임베디드 소프트웨어를 모델링하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구로서 MDA 기반의 임베디드 소프트웨어 개발 방법론과 확장된 xUML 노테이션 대해 알아본다. 3장에서는 제안한 방법론을

※ 본 연구는 정보통신연구진흥원 IT정책개발지원사업(B1220-0601-0031) 지원으로 수행되었음.
† 준 회 원 : 홍익대학교 일반대학원 박사과정
†† 정 회 원 : 홍익대학교 컴퓨터정보통신 교수
논문접수 : 2006년 11월 1일, 심사완료 : 2007년 1월 2일

지원해 주는 모델링 자동화 도구에 대해 살펴본다. 4장에서는 적용사례로서 이종 임베디드 시스템의 모델링을 보여준다. 마지막으로 5장에서는 결론 및 향후 연구를 언급한다.

2. 관련 연구

2.1 MDA 기반 임베디드 소프트웨어 개발 방법론

우리는 MDA(Model Driven Architecture) 기반의 임베디드 소프트웨어 개발 방법론[3, 12]을 제안하였다. 이를 HiMEM(Hongik MDA Based Embedded Software Com- po-nent Development Methodology) v0.1으로 명명하였다. HiMEM은 이해하기 쉬운 단계로 구분되어 있고 방법론에 따라 개발된 컴포넌트의 메타모델을 재사용함으로써 이종의 임베디드 소프트웨어 컴포넌트 시스템 개발을 쉽게 할 수 있다.

대표적인 임베디드 소프트웨어 개발방법론으로는 ILogix의 Harmony 프로세스[8]와 한국전자통신연구원의 EMMA (Embedded MARMI)[9]가 있다. <표 1>에 Harmony와 EMMA, HiMEM의 특징 및 장단점을 비교하였다.

우선 세 방법론을 비교했을 때, 가장 눈에 띄는 차이점은 개발의 시기이다. Harmony는 나선형 모델을 기반으로 정제된 프로토타입을 재사용하는 방법이다. EMMA는 한 번의 개발 라이프 사이클이 수행된 후에 생성된 핵심 자산[10]을 재사용하여 새로운 시스템 개발이 가능하다. 이에 반해 HiMEM은 한 번의 개발 라이프 사이클 중에 하나의 메타모델을 만든 후, 메타모델을 재사용하여 여러 개의 새로운 시스템 개발이 가능하다. 그리고 세 가지 모두 고품질의 임베디드 시스템을 적시에 경제적으로 개발할 수 있다는 점은 동일하지만 Harmony와 EMMA는 단일 제품군을 개발하기에 용이한 반면, HiMEM은 MDA를 기반으로 이종의 제품군 개발에도 용이하다. 하지만 HiMEM이 EMMA에 비해 장점만 있는 것은 아니다. MDA는 수작업이 아닌 모델의 자동 변환을 통해 여러 플랫폼을 쉽게 지원하고 코드 또한 쉽게 유지보수가 되도록 해야 한다. 이렇듯 자동화 도구, 특히 코드 자동 생성기가 필수 요건이라는 단점을 안고 있다. Harmony 역시 코드 자동 생성기를 필수적으로 요구한다.

〈표 1〉 임베디드 소프트웨어 개발방법론의 비교

Harmony	EMMA	HiMEM v0.1
1) 한번의 개발 라이프사이클 중에, 정제한 프로토타입 재사용하는 기법 2) 단종의 임베디드 시스템을 적시에 경제적으로 개발 (Single Product based Development) 3) 시스템/소프트웨어 책임 구분 4) UML 2.0 + SysML 적용 5) *코드 자동 생성기 필수*	1) 한번의 개발 라이프사이클 후에, 핵심 자산 재사용하는 기법 2) 단종의 임베디드 시스템을 적시에 경제적으로 개발 (Product-Line based Development) 3) Feature Driven 4) 커스터마이징 용이 5) UML 2.0 적용	1) 한번의 개발 라이프사이클 중에, 하나의 메타 모델을 재사용하는 기법 2) 이종의 임베디드 시스템을 적시에 경제적으로 개발 (Heterogeneous Product based Development) 3) xUML + UML 2.0 적용 4) Tailoring 가능 5) *코드 자동 생성기 필수*

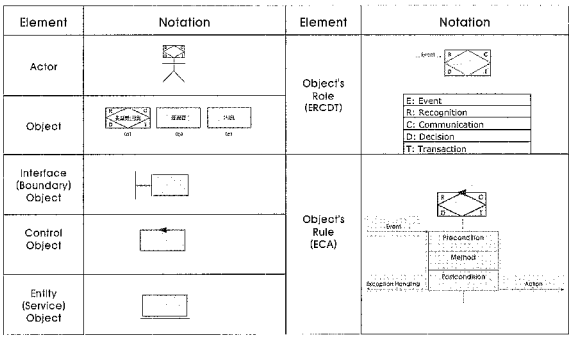
2.2 확장된 xUML

Executable UML[4-7]은 기호로 스펙을 설명하는 언어이다. 기존의 UML(Unified Modeling Language) 1.x[11] 표기법에 "실행에 관련된 개념(executable semantics)들"과 "시간에 관련된 규칙(timing rules)들"을 더한 것이다. 기존의 모호한 모델과 달리, Executable 모델은 시간에 관련된 문제와 동기화에 관련된 문제, 그리고 자원의 획득과 이용에 관련된 문제들을 자세히 서술한다. 결론적으로, Executable UML은 복잡한 실시간(realtime) 분산(distributed) 임베디드(embedded) 시스템의 스펙을 서술하기에 적합하며 많이 이용된다.

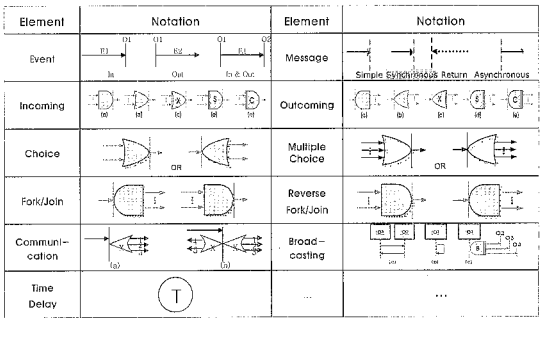
하지만 이러한 xUML 조차도 표현하지 못하는 부분이 있기에 xUML을 확장[12]하였다. <표 2>와 같이 확장된 xUML에서 클래스 다이어그램은 역할(Role)과 규칙(Rule)을 포함하며 시스템의 정적 구조를 묘사한다. 객체는 ERCDT(Event/Recognition /Communication/Decision/ Transaction)의 구조를 가지고 있다. 이는 객체가 각자 다른 역할을 수행하는 것을 의미한다. 인터페이스 객체는 이벤트(Event)가 들어오면 인지(Recognition)해서 통신(Communication)하는 ERC의 구조를 가진다. 그리고 제어 객체는 인터페이스의 기능뿐만 아니라 결정(Decision)을 내리고 수행(Transaction)하는 ERCDT의 모든 구조를 가질 수 있다. 또한 서비스 객체는 이벤트(Event)가 들어오면 수행(Transaction)을 하게 되는 ET의 구조를 가지게 된다. 이벤트가 들어오면 객체 내부에서 조건에 맞는지 검사를 한 후, 조건에 맞으면 액션을 수행하게 되고 맞지 않으면 예외처리를 하게 된다.

병렬 메시지 다이어그램(CMD: Concurrent Message Diagram)은 기존의 시퀀스 다이어그램에 실세계에서 발생될 수 있는 병렬 메커니즘(fork-join, reverse fork-join 등)을 포함하도록 확장한 것이다.

〈표 2〉 오브젝트 관련 노테이션



〈표 3〉 병렬 메시지 다이어그램 관련 노테이션



<표 3>은 확장된 xUML의 노테이션을 묘사한 것이다. 우선 확장된 xUML은 동기 메시지를 기본으로 한다. 그리고 클래스 다이어그램에서 정의한 것과 마찬가지로 각각의 객체에 역할(Role)이 있다. 이는 어떠한 이벤트가 들어오면 객체가 인지를 하고 통신을 하여 제어객체가 결정을 내리면 수행하게 되는 것이다. 이 모든 역할을 객체 내에 표현해 줌으로써 각 객체들의 역할을 한눈에 파악할 수 있다. 또 다른 객체의 특징으로는 객체 내부의 규칙을 가지고 있다는 것이다. ECA(Event/ Condition/ Action) 법칙을 따르기 때문에 이벤트가 들어오면 객체 내부에서 조건에 맞는지 검사를 한 후, 조건에 맞으면 액션을 수행하게 되고 맞지 않으면 예외처리를 하게 된다. 그리고 메시지가 지정한 시간 안에 수신이 되어야 하는 Time delay도 추가하여 사용자 행위를 좀 더 자세히 모델링 할 수 있다.

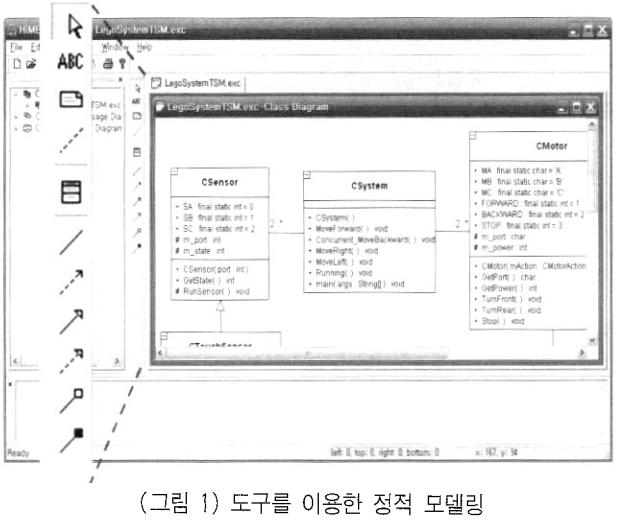
병렬 상태 다이어그램(CSD: Concurrent State Diagram)은 기본적으로 OCL(Object Constraint Language)을 포함하여 서술된다. 그리고 이 다이어그램은 Deterministic/ Stochastic 메커니즘을 포함한다. 향후 Non-deterministic 상태를 Deterministic 상태로 자동 변환할 수 있도록 연구를 진행 중이다.

3. 자동화 도구

본 장에서는 제안한 방법의 자동화를 위해 구현한 도구에 관해 설명한다. 도구는 확장된 xUML 노테이션이 적용되어 실시간의 병렬성까지 표현이 가능하다. 현재 모델링 단계까지는 완성이 되어있는 상태이며 자동 코드 생성에 관해 연구가 진행 중이다.

3.1 정적 모델링

클래스는 툴바의 클래스 버튼( )을 누르고 다이어그램 윈도우에 클릭을 하면 클래스가 생성된다. 이때 오브젝트 뷰에는 생성된 클래스 이름과 속성과 연산자가 트리형태로 화면에 표시된다. (그림 1)은 도구를 이용하여 정적 모델링을 해 본 것이다.

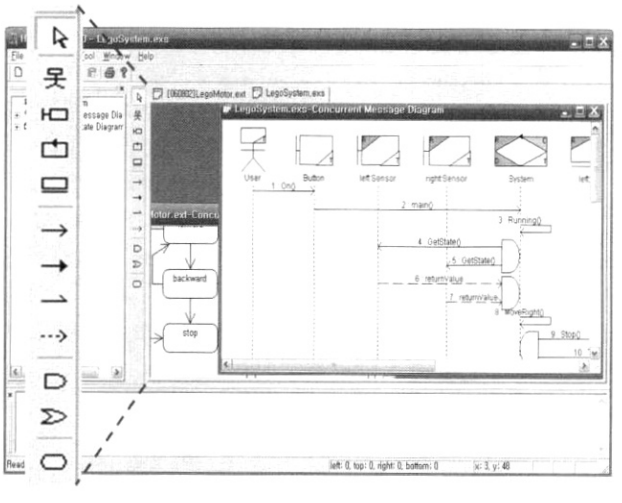


(그림 1) 도구를 이용한 정적 모델링

연관 관계는 툴바에서 연관(, Association), 의존(, Dependency), 일반화(, Generalization), 실현(, Realize) 4가지 타입을 설정할 수 있다. 4가지 타입 중 원하는 관계를 설정하고 싶은 클래스 위에서 마우스를 클릭하여 누르고 있는 상태에서 연결될 클래스로 마우스 드래그 하여 마우스 버튼에서 손을 때면 연결이 된다. 클래스 위에서 클릭하지 않으면 그려지지 않으므로 유의해야 된다. 또한 연결된 선을 더블클릭하여 관계 속성 창이 나오면 관계의 이름을 설정할 수 있고 접근자(public, protected, private)를 지정할 수 있다. 접근자는 기본적으로 Public으로 설정 되어 있다.

3.2 동적 모델링

툴바에서 액터(, Actor), 인터페이스(, Interface), 컨트롤(, Control), 서비스(, Service)의 4가지 중 하나를 선택한 다음 뷰어창에 마우스 왼쪽 클릭으로 오브젝트를 생성한다.



(그림 2) 도구를 이용한 동적 모델링

(그림 2)는 도구를 이용한 동적 모델링을 표현한 것이다. 툴바에서 플랫(, Flat), 동기(, Synchronous), 리턴(, Return), 비동기(, Asynchronous) 4가지 종류의 메시지를 선택할 수 있다. 먼저 4가지 중 원하는 메시지를 클릭하고 뷰어창에 마우스 왼쪽버튼을 이용하여 연결된 객체와 객체사이를 마우스의 드래그를 이용하여 두 객체를 연결하게 되면 뷰어 창에 객체와 객체사이에 메시지가 설정되고 그에 맞는 메시지가 그려지게 된다. 또한 메시지 속성 창에서 메시지의 이름을 설정하여 줄 수가 있다. 툴바에서 이벤트는 AND( )와 OR( ) 을 이용하여 나타낼 수 있는데 앞서 그렸던 방법과 같이 툴바에서 그리고자하는 이벤트를 클릭한 다음 뷰어 창에 마우스 왼쪽 버튼으로 클릭하여 그릴 수가 있다. 기본적으로 이벤트는 AND로 설정되어 있는데 더블클릭하여 속성 창을 띄우고 다른 이벤트(Concurrent, Sequential, Broadcast)로 변환할 수 있다. 이때 Incoming과 Outgoing의 두 방향을 설정할 수 있다. AND와 마찬가지로 OR 역시 같은 방법으로 뷰어창에 나타내고 더블클릭

하여 속성 창에서 다른 종류(XOR)의 이벤트로 변환하고 방향을 설정할 수 있다.

4. 적용 사례

본 장에서는 적용사례로서 이종의 임베디드 시스템을 모델링 해 보았다. 첫 번째 절에서는 이종의 시스템이 보유하고 있는 하드웨어 및 소프트웨어 성능을 비교 설명하였다. 두 번째 절에서는 개발한 모델링 자동화 도구를 사용하여 타겟에 독립적인 메타모델을 생성하였다. 그리고 세 번째 절에서는 기 개발된 메타모델을 이용하여 타겟에 종속적인 모델을 생성하였다. 본 논문에서는 자동화 도구를 이용한 모델링을 설명하는 관계로 타겟 독립 모델에서 타겟 종속적인 모델로 변환 시의 세부적인 연관관계는 생략하였다. 마지막 절에서는 타겟 종속 모델을 이용하여 타겟 종속적인 코드를 생성하였다. 도구는 현재 모델링 단계까지만 자동화 되므로 수작업을 통해 코드화하였다.

4.1 이종 임베디드 시스템의 비교

4.1.1 HexAvoider.

HexAvoider의 CPU는 Atmel사의 AT89C51이다. AT89C51은 Intel사의 8051 호환 One-chip CPU이며 4K byte의 Flash Memory를 내장하고 있다. 그리고 롬 라이터(ROM Writer)를 사용하여 손쉽게 프로그램을 지우고 써 넣을 수 있다. HexAvoider는 서보 모터들을 제어하는 것 이외에도 많은 일들을 해야 하기 때문에 처음에는 Thread 개념의 멀티태스킹을 고려했지만 CPU와 리소스의 한계로 폴링 기법을 사용해야 했다. HexAvoider는 C언어로 동작한다.

4.1.2 Javelin

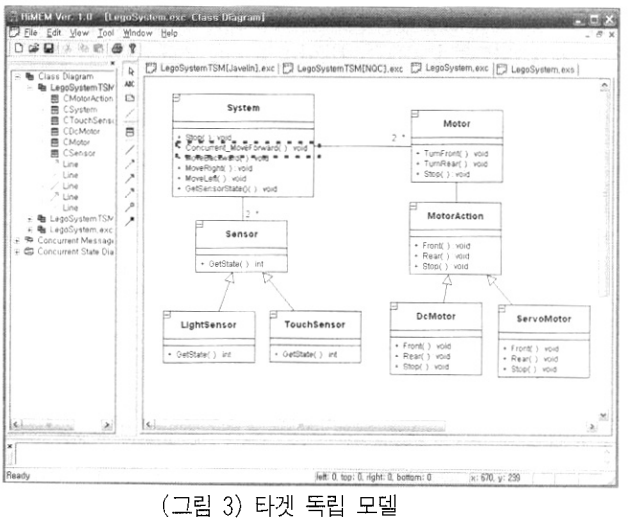
Javelin의 CPU는 Ubicom사의 SX48AC이다. 이 CPU는 32Kbyte의 RAM 용량과 32 kbyte의 EEPROM을 가지고 있다. 그리고 시리얼 포트를 사용하여 손쉽게 프로그램을 지우고 써 넣을 수 있다. Javelin는 서보 모터들을 제어하는 것 이외에도 많은 일들을 해야 하기 때문에 처음에는 Thread 개념의 멀티태스킹을 고려했지만 Javelin이 쓰레드를 허용하지 않고 어셈블리 언어를 제공하지 않기 때문에 폴링 기법을 사용해야 했다. Javelin은 Java언어로 동작한다.

이처럼 HexAvoider와 Javelin은 CPU, 운용환경, 그리고 사용하는 프로그램 언어까지 상이한 이종의 임베디드 소프트웨어 시스템이다. 다음 절에서는 타겟 독립 모델단계에서 하나의 메타모델을 만든 후, 재사용하여 각각의 타겟 종속 모델이 생성되는 것을 보여줄 것이다.

4.2 타겟 독립 모델

4.2.1 정적 모델의 타겟 독립 모델

(그림 3)는 타겟 독립 모델을 설계한 것이다. 본 논문에서는 하드웨어를 변경하지 않고 소프트웨어의 변경만으로 시스템을 컨트롤 하고자 하였다. 그래서 센서에 닿으면 멈추도록 설계되었던 기존의 모델을 변경하여 좌우로 회전할 수 있는 메소드인

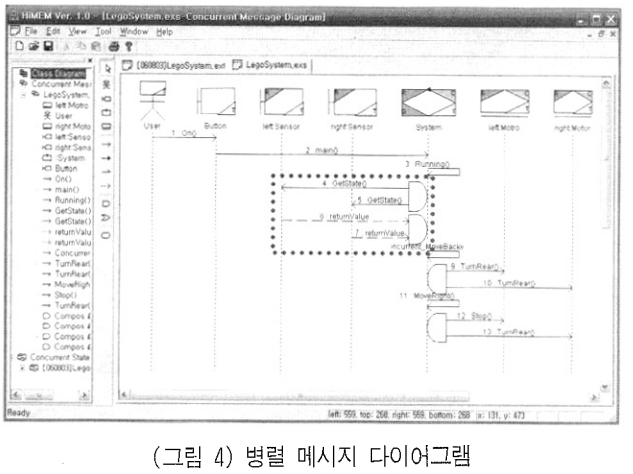


(그림 3) 타겟 독립 모델

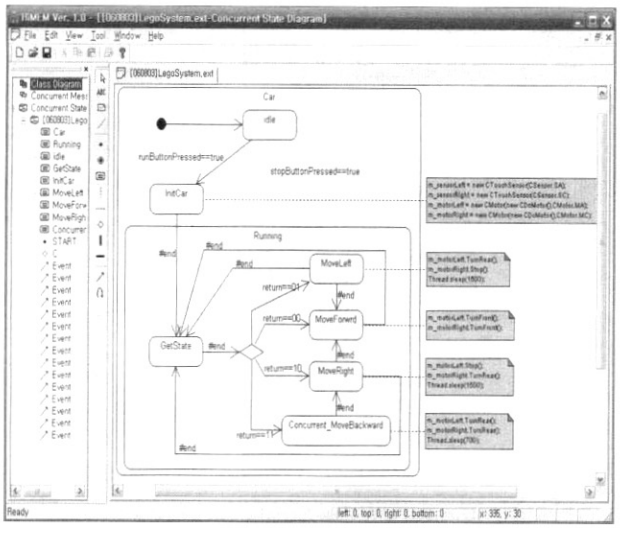
MoveLeft()과 MoveRight()을 추가하였다. 또한 두 센서에 동시에 이벤트가 발생했을 때에는 시스템이 후진할 수 있도록 Backward()를 Concurrent_MoveBackward()로 변경하였다. (그림 4)에서 점선으로 표시해놓은 부분이 동시성을 모델링 한 부분이다. 그리고 각각의 시스템이 사용하는 모터가 상이하기 때문에 모터의 행위를 분리하여 모델링 하였다.

4.2.2 동적 모델의 타겟 독립 모델

본 논문에서는 기존의 동적 다이어그램에서 표현 못하는 점을 보완한 병렬 메시지 다이어그램을 사용하였다. 이는 기존의 시퀀스 다이어그램에 Role과 Rule 개념을 첨가하고, 동시성을 표현 못했던 한계를 보완하여 제안한 노테이션이다. (그림 4)는 시스템을 병렬 메시지 다이어그램으로 표현한 것이다. 점선으로 표시된 부분은 병렬 메시지 다이어그램의 Reverse Fork/Join 동작을 나타낸다. 이것은 두 개의 센서가 동시에 반응했을 때 콘트롤러는 두 모터에 동시다발적으로 제어 신호를 보내는 것을 의미한다. 이처럼 병렬 메시지 다이어그램은 Reverse Fork/Join등 실시간의 동시성까지 표현할 수 있다. 그리고 확장된 xUML에서는 동기 메시지를 기본으로 하기 때문에 리턴 메시지도 함께 표현되어 있다.



(그림 4) 병렬 메시지 다이어그램



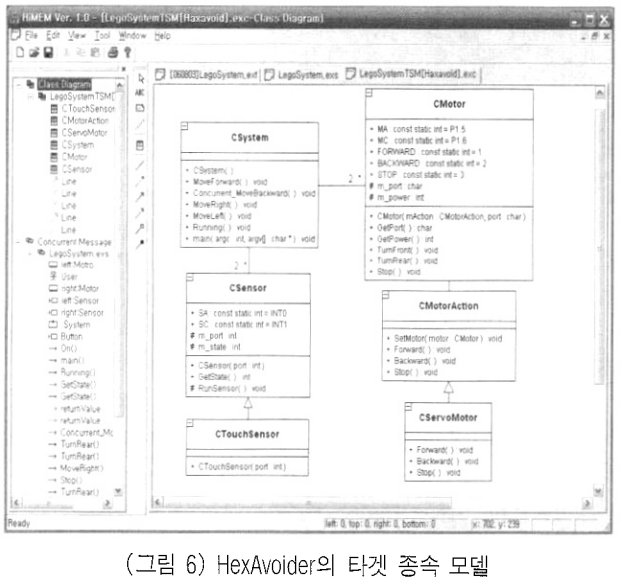
(그림 5) 병렬 상태 다이어그램

(그림 5)는 시스템의 병렬 상태 다이어그램으로서 센서가 동작하는 Polling 알고리즘을 고려하여 표현하였다. 병렬 상태 다이어그램은 상태의 Non-deterministic한 영역까지 확장할 계획이지만 현재로서는 Stochastic 단계 까지만 표현이 가능하다.

4.3 타겟 종속 모델

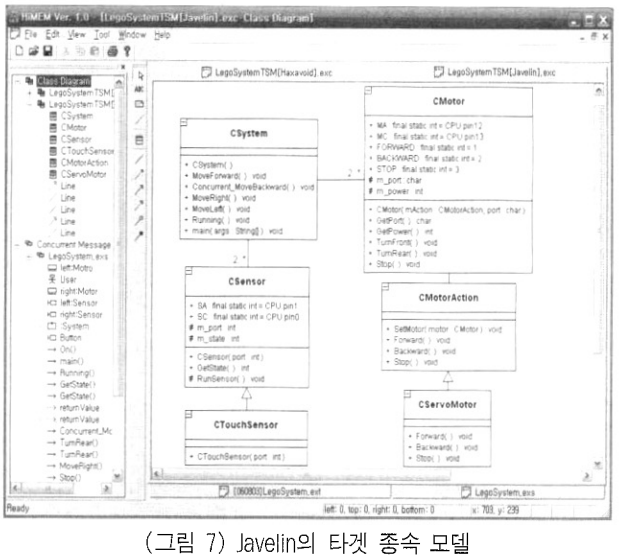
4.3.1 정적 모델의 타겟 종속 모델

(그림 6)은 타겟 독립 모델에서 파생된 HexAvoider의 타겟 종속 모델이다. 이때 CPU와 프로그램 언어를 고려하여 타겟 종속 모델이 파생된다. 본 논문에서는 자동화 도구를 이용하여 이루어지는 모델링을 설명하므로 타겟 독립 모델과의 세부 연관관계는 생략하였다.



(그림 6) HexAvoider의 타겟 종속 모델

(그림 7)은 Javelin의 타겟 종속 모델이다. 이것 역시 (그림 6)과 같이 타겟 독립 모델에서 CPU및 프로그램 언어를 고려하여 파생된 것이다.



(그림 7) Javelin의 타겟 종속 모델

4.4 타겟 종속 코드

이번 절에서는 앞서 생성된 타겟 종속 모델을 사용하여 수작업으로 코드화 하였다. 코드 자동화 도구에 관한 연구가 여전히 진행 중이다. <표 4>와 <표 5>의 코드는 한가지의 병렬 메시지 다이어그램에서 나온 다른 코드이다.

〈표 4〉 HexAvoider의 코드

C

```
void move_forward(void)
{
        left_leg_back_right_leg_front();
        delay(movement_delay);
        if(sense_result) return;

        left_leg_front_right_leg_back();
        delay(movement_delay);
        if(sense_result) return;

        return ;
}
void move_backward(void)
{
        left_leg_front_right_leg_back();
        delay(movement_delay);
        delay(movement_delay);

        left_leg_back_right_leg_front();
        delay(movement_delay);
        delay(movement_delay);
}
void turn_right(void)
{
        left_right_legs_back();
        delay(movement_delay);
        delay(movement_delay);

        left_right_legs_front();
        delay(movement_delay);
        delay(movement_delay);
}
void turn_left(void)
{
        left_right_legs_front();
        delay(movement_delay);
        delay(movement_delay);

        left_right_legs_back();
        delay(movement_delay);
        delay(movement_delay);
}
void main(void)
{
        while(1)
        {
                move_forward();
                int sense_result=0;
                if(!SENSOR_LEFT) { sense_result++; }
                if(!SENSOR_RIGHT) { sense_result+=2; }

                switch(sense_result)
                {
                case 1:
                        turn_right(); break;
                case 2:
                        turn_left(); break;
                case 3:
                        concurrent_move_backward();
                        turn_right(); break;
                } } }
```

〈표 5〉 Javelin의 코드

Java

```
public class CSystem
{
    public static final int SPEED = 120;
    public static void turn_left()
    {
        CPU.pulseOut(115,CPU.pin12);
    }
    public static void turn_right()
    {
        CPU.pulseOut(230,CPU.pin13);
    }
    public static void move_forward()
    {
        CPU.pulseOut(230,CPU.pin12);
        CPU.pulseOut(115,CPU.pin13);
    }
    public static void concurrent_move_backward()
    {
        CPU.pulseOut(115,CPU.pin12);
        CPU.pulseOut(230,CPU.pin13);
    }
    public static void main()
    {
        Timer tSensing = new Timer();
        Timer tLoMotor = new Timer();
        boolean SENSOR_LEFT= true;
        boolean SENSOR_RIGHT = true;
        tSensing.mark();
        tLoMotor.mark();
        while (true)
        {
         if(tLoMotor.timeout(CSystem.SPEED))
         {
            if( (!SENSOR_LEFT && SENSOR_RIGHT) )
            {
                turn_reft();
                CPU.delay(3000);
            }
            else if( (!SENSOR_RIGHT && SENSOR_LEFT) )
            {
                turn_right();
                CPU.delay(3000);
            }
            else if( (!SENSOR_LEFT&& !SENSOR_RIGHT))
            {
                concurrent_move_backward();
                CPU.delay(3000);
                turn_right();
            }
            else
            {
                move_forward();
            }
            tLoMotor.mark();
         }
         if(tSensing.timeout(20))
         {
                left = CPU.readPin(CPU.pin0);
                right = CPU.readPin(CPU.pin7);
                CPU.writePin(CPU.pin1,left);
                CPU.writePin(CPU.pin8,right);
                tSensing.mark();
        } } } }
```

실제적으로 각각의 임베디드 시스템들은 하드웨어 구조에 종속적이기 때문에 소스 코드 레벨에서의 재사용은 불가능하다. 그래서 우리는 MDA 기반으로 모델을 재사용하여 코드를 생성 하고자 한다. 현재는 모델 자동화 도구에서 코드 발생기를 개발 중이다.

5. 결 론

임베디드 시스템은 하드웨어에 종속적이기 때문에 기 개발된 설계와 코드의 재사용이 어렵다고 한다[1]. 본 논문에서는 MDA 기반의 임베디드 개발 방법론인 HiMEM을 소개하였다. 제안한 방법을 통해 임베디드 소프트웨어를 재사용하려 한다. 또한 확장된 xUML을 제안함으로서 임베디드 시스템의 동적인 모델링을 향상시켰다. 기존의 시퀀스 다이어그램에 AND/OR/XOR 등의 개념과 병렬과 Fork/Join등의 개념도 추가함으로서 임베디드 소프트웨어 시스템의 동시 발생 문제등도 모델링이 가능하였고 모델링 도구를 구현하였다. 이는 모델링을 자동화하고 수작업을 통한 코딩이 가능하게 하여 보다 빠르고 안정적으로 시스템을 개발할 수 있었다.

현재는 개발한 모델의 검증과 코드 생성 자동화 도구를 개발 중이다. 그리고 타겟 종속 모델에 적용할 수 있는 하드웨어 프로파일에 관한 연구가 필요하다.

참 고 문 헌

[1] Axel Jantsch, 'Modeling Embedded System and SOCs', Mogan Kaufmann, 2004.

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

[3] W. Kim, R. Y. Kim, "Adapting Model Driven Architecture for Modeling Heterogeneous Embedded S/W Components," ICHIT, Vol. 2, 2006. 11.

[4] Leon Starr, 'Executable UML: How to Build Class Model', Prentice Hall, 2002.

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

[6] Kennedy Carter Ltd., Executable UML: An Online Tutorial, http://www.kc.com

[7] 김인기 역, 'Executable UML: 클래스 모델 만들기, 정보문화사', 2003

[8] Bruce Powel Douglass, 'Real-Time UML: Developing Efficient Objects for Embedded Systems', 2nd Edition, Addison-Wesley, 1999.

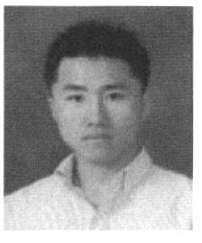
[9] 양영종, 조진희, 하수정, 차진희, "임베디드 시스템 개발 방법론 및 재사용 체계", 전자통신동향분석, 제21권, 제1호, 2006. 2.

[10] Kyo C. Kang, Jaejoon Lee, and Patrick Donohoe, "Feature-Oriented Product Line Engineering," IEEE Software, Vol.9, No.4, Jul./Aug., 2002,

[11] Object Management Group, OMG Unified Modeling Language Specification (draft) Version 1.3, June, 1999.

[12] 김우열, 김영철, "실시간 임베디드 소프트웨어 모델링을 위한 xUML 확장에 관한 연구", KIPS, 춘계학술대회 논문집, Vol.13, No.1, 2006. 5.

김 우 열



e-mail : john@selab.hongik.ac.kr
2004년 홍익대학교 컴퓨터정보통신(학사)
2006년 홍익대학교 소프트웨어공학전공 (석사)
2006년~현재 홍익대학교 박사과정
관심분야 : 상호운용성, 임베디드 소프트웨어 개발 방법론 및 도구 개발, 컴포넌트 시험 및 평가, 리팩토링

김 영 철

e-mail : bob@selab.hongik.ac.kr
2000년 Illinois Institute of Technology (공학박사)
2000년~2001년 LG 산전 중앙연구소 Embedded system 부장
2001년~현재 홍익대학교 컴퓨터정보통신 교수
관심분야 : 소프트웨어 성숙도 모델, Use Case 방법론 및 도구 개발, CBD, BPM, 사용자 행위 분석 방법론