KIPS
한국정보처리학회
19권 제1호
ol. 9 No. 1

2007

한국 소프트웨어공학 학술대회 논문집

Proceedings of 2007 Korea Conference on Software Engineering

- 일시 : 2007년 2월 22일(목)~24일(토)
- 장소 : 강원도 용평 리조트

주최 : 한국정보처리학회
한국정보과학회
한국전자통신연구원

주관 : 한국정보처리학회 소프트웨어공학 연구회
한국정보과학회 소프트웨어공학 연구회

후원 : (주)비트컴퓨터
(주)소프트피아
(주)한국 썬마이크로시스템즈
KAIST 소프트웨어 프로세스 개선 센터

목 차

이종 임베디드 소프트웨어를 위한 코드 생성 메커니즘 및 지원도구

손현승, 김우열, 서윤숙, 김동호, 김동우, 김재수, 김영철

홍익대학교 컴퓨터정보통신공학 소프트웨어공학 연구실
충남 연기군 조치원 홍익대학교
{sonhs, john, jyun, ray, dwkim, jaesoo, bob}@selab.hongik.ac.kr

요약: 사용자 요구사항의 증가와 하드웨어의 발달에 따라 임베디드 시스템의 복잡성이 증가되고 있다. 이렇게 복잡한 기능을 제공하는 임베디드 소프트웨어를 좀 더 빠르고 효율적으로 개발하기 위해 소프트웨어의 재사용은 필수다. 하지만 임베디드 소프트웨어는 하드웨어에 의존적이기 때문에 기 개발된 설계와 코드의 재사용이 어렵다고 한다. 본 논문에서는 MDA(Model Driven Architecture) 기반으로 이종 임베디드 소프트웨어를 위한 코드생성 메커니즘을 제시하고 이를 이용하여 자동 코드 생성기를 구현하였다. 적용사례로서 자동화 도구를 사용한 이종 임베디드 시스템의 코드 생성을 보여주었다.

핵심어: 임베디드 시스템, 자동화 도구, Executable UML, MDA(Model Driven Architecture)

1. 서론

사용자 요구사항의 증가와 하드웨어의 발달에 따라 임베디드 소프트웨어 시스템도 복잡성이 증가되고 있다. 이렇게 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].

소프트웨어 공학에서 MDA(Model Driven Architecture)[2]는 플랫폼에 독립적인 메타모델을 설계한 후 필요한 기술 모델을 변경하여 그 모델을 통해 코드 생성을 자동화하는 메커니즘이다. 만약 응용 프로그램이 다른 구현 환경에서 필요하다면 그 환경에 대한 모델을 선택하고 다시 코드를 생성하면 된다. 이때 응용 프로그램 모델을 수정할 필요는 없다. 이와 같은 방법으로 모델의 재사용과 관련된 코드의 생산성을 높일 수 있다. 그러므로 임베디드 소프트웨어 개발 방법과 MDA 메커니즘이 접목된다면 설계를 재사용할 수 있고, 소프트웨어를 정량화 할 수 있으며, 개발 시간 또한 단축시킬 수 있을 것이다. 또한 모델의 재사용으로 인하여 하드웨어에 독립적으로 소프트웨어 개발 또한 가능해질 것이다[3].

본 논문에서는 임베디드 소프트웨어를 위한 코드 생성 메커니즘을 제시하고 이를 이용하여 구현한 자동화 도구를 소개한다. 적용사례로서 자동화 도구를 사용하여 다른 언어로 동작하는 이종 임베디드 시스템의 코드 생성을 보여준다.

본 논문의 구성은 다음과 같다. 2 장에서는 관련연구로서 임베디드 소프트웨어를 위한 MDA 기반 개발 프로세스와 확장된 xUML 노테이션 대해 알아본다. 3 장에서는 이종의 임베디드 시스템의 코드를 개발하기 위한 방법을 제안한다. 4 장에서는 적용사례로서 도구를 이용한 이종 임베디드 시스템의 모델링 및 자동 코드 생성을 보여준다. 5 장에서는 임베디드 소프트웨어를 개발할 수 있는 자동화 도구들을 비교 분석하였다. 마지막으로 6 장에서는 결론 및 향후 연구를 언급한다.

2. 관련연구

2.1 임베디드 소프트웨어 개발 프로세스

MDA 기반의 임베디드 소프트웨어 개발 프로세스[3,13]를 도식화 하면 그림 1 과 같다. 이는 요구사항 및 분석을 거쳐 확장된 xUML 로 타겟 독립 모델을 설계한다. 다음으로는 운영체제 및 프로세서를 고려한 프로파일을 적용하여 각자 타겟에 종속적인 모델인 타겟 종속 모델로 변환되고, 타겟 종속 모델 또한 자동 코드화 된다.

이후 생성된 소스코드를 우리의 타겟 모델에 탑재하는 작업이 바로 개발한 소프트웨어를 임베디드 소프트웨어 시스템에 적용시키는 것이다. 이때 생성된 소스코드와 모델은 재사용 저장소에 저장되고 후에 기능을 변경하거나 추가하여 새로운 임베디드 시스템을 구축하고자 한다면 저장된 모델을 재사용하여

개발 시간을 단축시킬 수 있다. 최종적으로는 모델 단계에서의 재사용을 통해 우리의 최종 목표인 상호 운용성 또한 높일 수가 있다.

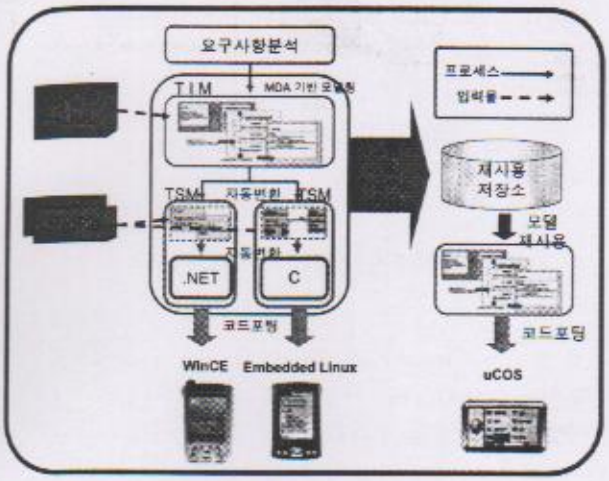

그림 1. MDA 기반 개발 프로세스[3,13]

2.2 확장된 xUML

Executable UML[4,5,6,7]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 UML 은 복잡한 실시간(realtime) 분산(distributed) 임베디드(embedded) 시스템의 스펙을 서술하기에 적합하며 많이 이용된다.

하지만 이러한 xUML 조차도 표현하지 못하는 부분이 있기에 xUML 을 확장[12]하였다. 표 1 과 같이 확장된 xUML 에서 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 ERC 의 구조를 가진다. 그리고 제어 객체는 인터페이스의 기능뿐만 아니라 결정(Decision)을 내리고 수행(Transaction)하는 ERCDT 의 모든 구조를 가질 수 있다. 또한 서비스 객체는 이벤트(Event)가 들어오면 수행(Transaction)을 하게 되는 ET 의 구조를 가지게 된다. 이벤트가 들어오면 객체 내부에서 조건에 맞는지 검사를 한 후, 조건에 맞으면 액션을 수행하게 되고 맞지 않으면 예외처리를 하게 된다.

표 1. 오브젝트 관련 노테이션

Element	Notation	Element	Notation
Actor		Object's Role (ERCDT)	
Object			
Interface (Boundary) Object		Object's Rule (ECA)	
Control Object			
Entity (Service) Object			

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. 병렬 메시지 다이어그램 관련 노테이션

Element	Notation	Element	Notation
Event		Message	
Incoming		Outcoming	
Choice		Multiple Choice	
Fork/Join		Reverse Fork/Join	
Communi-cation		Broad-casting	
Time Delay	T	...	...

표 2 는 확장된 xUML 의 노테이션을 묘사한 것이다. 우선 확장된 xUML 은 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.

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[12,13].

3. 코드 생성 메커니즘

모델링 한 3 가지 다이어그램 즉, 클래스 다이어그램, 병렬 메시지 다이어그램, 병렬 상태 다이어그램을 통해 소스코드를 생성하기 위해서는 이 각 다이어그램 간의 유기적인 연결과 공통의 데이터 구조를 사용해야 한다. 서로 다른 데이터구조를 가지게 되면 소스코드를 생성하기에 매우 복잡한 구조를 가지게 되기 때문이다. 그림 2 는 메모리 상에 실제 생성되는 구조를 도식화한 모델이다.

<table>
<tr><td colspan="2">Class Name: string</td></tr>
<tr><td colspan="2">Package List: List</td></tr>
<tr><td colspan="2">Parent List : List</td></tr>
<tr><td colspan="2">Interface List : List</td></tr>
<tr><td colspan="2">Association List: List</td></tr>
<tr><td rowspan="2">Association List</td><td>Attribute</td></tr>
<tr><td>SetFunction</td></tr>
<tr><td colspan="2">Attribute List: List</td></tr>
<tr><td colspan="2">Function List: List</td></tr>
<tr><td rowspan="2">Function List</td><td>Head</td></tr>
<tr><td>Body</td></tr>
</table>

그림 2. 소스코드를 생성하기 위한 메타 모델

그림 2 의 메타 모델을 차례대로 설명하면 Class Name 은 클래스 이름을 나타낸다. Package List 는 문자열 리스트 이고 C/C++ 에서는 include 부분이 되고 Java 에서는 import 가 되는 부분이다. Parent List 는 문자열 리스트이고 부모 클래스를 나타낸다. 여기서 문자열 리스트로 나타낸 것은 C++ 에서의 복수 상속성 때문에 리스트구조를 가졌다. Interface List 는 Interface 를 연결되는 부분으로 Java 를 위해서 따로 분리 하였다. Association List 는 객체간의 연관관계를 나타내는데 자동적으로 Set 함수를 만들어주어 자동 생성함으로 사용자가 연결하는 함수를 만들지 않아도 되도록 하였다. Attribute List 와 Function List 는 속성과 메소드의 리스트를 나타낸다. 메타 모델이 어떻게 다이어그램과 매핑되는지를 각 다이어그램을 보면서 설명하겠다.

3.1 클래스 다이어그램

클래스 다이어그램은 소프트웨어의 정적 구조를 설계한다. 이는 속성, 메소드, 연관관계를 보여준다.

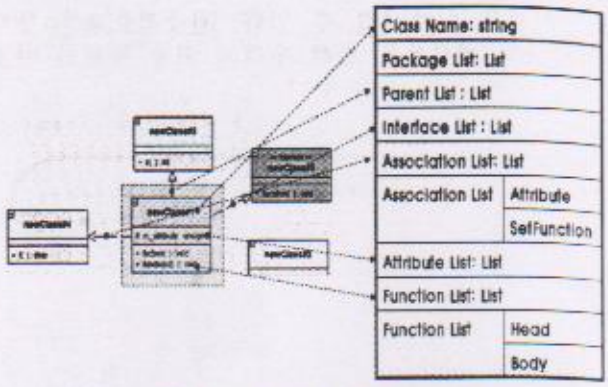

그림 3. 클래스 다이어그램과 메타 모델

그림 3 은 클래스 다이어그램에서 서로 영향을 받는 부분을 화살표를 이용하여 표시하였다. 클래스 이름은 Class Name 에 저장된다. Parent List 는 모델에서 상속 관계를 가질 때 데이터가 추가된다. Interface List 는 인터페이스 클래스와 관계를 가질 때 데이터가 추가 된다. Association List 는 연관관계를 가질 때 데이터가 추가된다. Attribute List 와 Function List 는 변수와 함수가 추가될 때 데이터가 입력된다.

3.2 병렬 메시지 다이어그램

병렬 메시지 다이어그램은 어떤 객체와 연결되는지 객체가 얼마나 생성되어야 하는지를 결정해주는 역할을 해준다. 그림 4 와 같이 실제 메소드의 내용을 채운다. 하지만 여기서 생성된 코드는 최종 코드에서 변형될 수 있다. 최종 함수의 결정은 병렬 상태 다이어그램을 통해서 이루어지기 때문이다. 결국 여기서 생성된 메소드의 내용이 병렬 상태 다이어그램에서의 내용과 결합되어 완벽한 메소드를 이루게 된다.

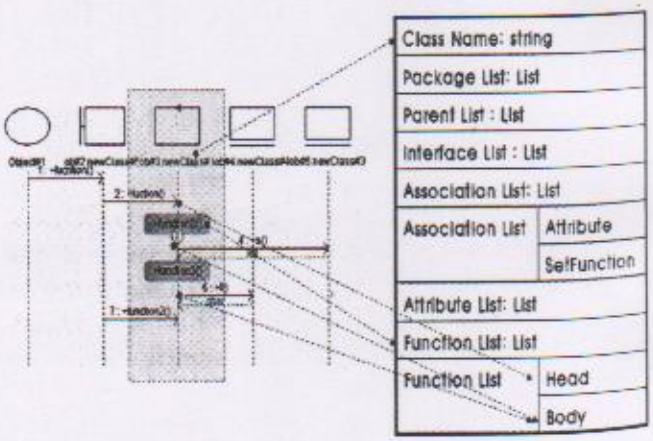

그림 4. 병렬 메시지 다이어그램과 메타 모델

3.3 병렬 상태 다이어그램

클래스 다이어그램과 병렬 메시지 다이어그램이 뼈대를 만들어 주었다면 병렬 상태 다이어그램은 살을 붙이는 과정이다. 그렇기 때문에 실제 코드와 많은 부분이 유사한 구조로 되어있고 실제코드와도 직접 매핑 된다. 그림 5 에 나타난 것처럼 이미 다이어그램을 통해 데이터가 채워졌기 때문에 수정을 통해서 완벽한 코드를 생성한다.

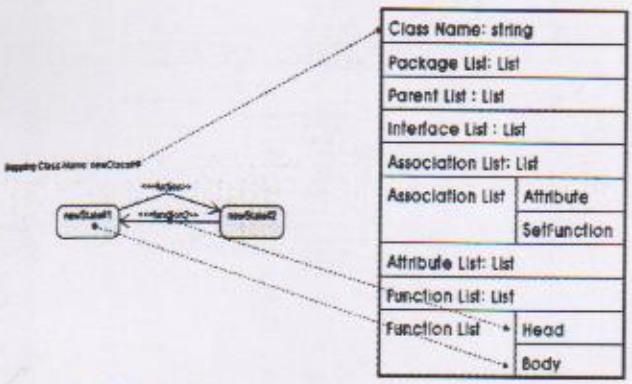

그림 5. 병렬 메시지 다이어그램과 메타 모델

4. 개발 지원도구의 사례

설계 자동화 도구는 임베디드 소프트웨어를 개발할 때 시스템을 분석 설계하고 코드를 자동 생성한 후에 최종 시스템을 구축할 수 있도록 지원해 주는 기술이다. 본 장에서는 설계 자동화 도구의 설계 모델과 주요 메커니즘을 소개하고, 도구를 통하여 코드를 생성한 사례를 보여준다.

4.1 도구의 설계 모델

본 절에서는 자동화 도구의 설계와 코드생성 메커니즘에 대하여 설명한다.

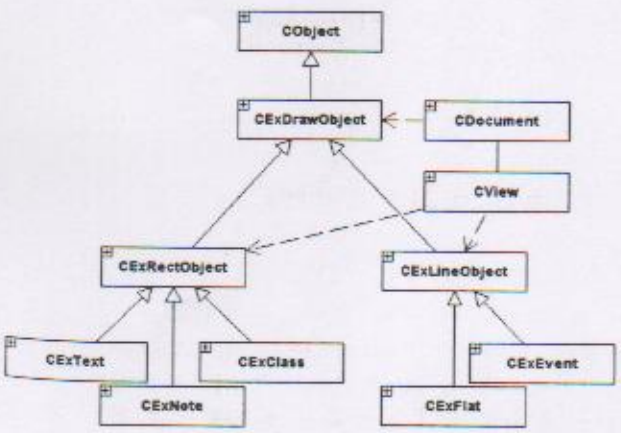

그림 6. 자동화 도구의 클래스 다이어그램

그림 6 은 자동화 도구의 클래스 다이어그램 이다. 도구는 크게 두 가지 부분으로 나눌 수 있다. 설계모델을 생성하는 모듈과 코드를 생성하는 모듈이다. 이는 두 가지 모듈로 분리되어 있다.

첫 번째는 설계모델을 생성하는 모듈로 모델을 화면에 표현할 수 있는 기술이다. 구현된 도구의 클래스는 124 개로 이것을 모두 표현한다면 너무 복잡하여 본 논문에서는 중요한 부분만 간략하게 표현 하였다. CObject 는 가장 기본이 되는 클래스이다. 이렇게 CObject 를 상속받게 되면 기본적으로 몇 가지 편리한 기능을 사용할 수 있는 장점이 있다. 첫 번째는 동적으로 클래스의 이름을 알아낼 수 있다. 두 번째는 순서화 되어 파일을 저장할 때 편리하게 사용할 수 있다. 각 클래스들의 역할은 표 3 과 같다.

표 3. 클래스의 역할

클래스 이름	역 할
CExDrawObject	CObject 다음으로 가장 높은 부모 클래스 이다. 여기에 객체들이 가져야 될 가장 기본적인 정보를 가지고 있다.
CExRectObject	이것은 사각형 모양을 가질 수 있는 모든 객체를 포함한다. 모양은 원이여도 표현을 사각형으로 표현할 수 있으면 모두 이 클래스를 상속 받는다.
CExText	사각형 특성을 가지고 여기에 텍스트를 입력 받는 기능을 포함한다. 이 객체는 아무 곳에나 글을 입력하여 객체를 이동시켜 놓을 수 있다.
CExNote	CExText 와 비슷하지만 CExNote 는 배경과 테두리를 가지고 있다.
CExClass	클래스 다이어그램에서 사용되는 Class 이다. 이외에 다른 다이이그램에서 사용되는 객체 중 사각형 특성을 가지는 객체는 모두 CExRectObject 의 형제가 될 수 있다.
CExLineObject	이 클래스는 특성이 선인 객체들을 포함한다. 이 클래스에는 기본적으로 점 편집기능과 꺾은 선을 그릴 수 있도록 되어있다. 그래서 선을 그릴 때는 이 클래스를 이용하면 쉽게 표현이 가능하다.
CExFlat	CExLineObject 의 자식 클래스로 모양만 틀리고 대부분의 기능이 같다.
CExEvent	CExLineObject 의 자식 클래스로 모양만 틀리고 대부분의 기능이 같다.
CDocument	MFC 의 기본 클래스로 데이터의 저장과 자료구조를 가지고 있다.
CView	MFC 의 기본 클래스로 화면 처리하는 윈도우 이다.

두 번째는 코드를 생성하는 모듈이다. 코드생성은 모델의 메타모델을 그대로 문자열로 바꾸어 파일로 만들면 된다. 그림 7 은 메타 모델의 내용을 그대로 코드에 적용하는 것을 보여준다.

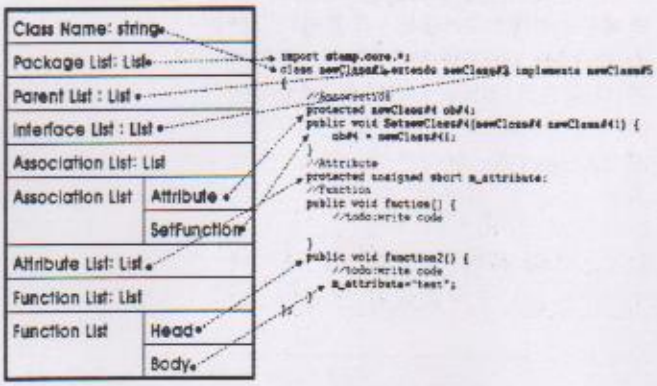

그림 7. 메타 모델을 이용한 소스 코드 생성

C++, 자바의 클래스 선언 부를 살펴보면 일정한 규칙을 찾을 수 있다. 이 규칙을 전 장에서 생성한 메타모델과 매핑을 하면 된다. Class Name 은 클래스의 이름이 되고, Package List 는 포함될 패키지를 얻어오게 된다. 그림 7 의 화살표와 대응되는 위치를 통해 확인 할 수 있다.

4.2 도구의 적용 사례

본 절에서는 확장된 xUML 을 사용하여 모델링 한 후 자동화 도구에 의해 코드가 자동으로 생성되는 사례를 보여준다. 이종의 시스템의 코드생성을 위해 사용된 제품은 Parallax 사의 Javeline 과 LEGO 의 MindStorm 이다.

표 4 는 Javelin 과 MindStorm 의 하드웨어 기본정보를 보여 준다. Javelin 의 Microcontroller 는 Ubicom 사의 SX48AC 이다. 이 CPU 는 32Kbyte 의 RAM 용량과 32 kbyte 의 EEPROM 을 가지고 있다. 그리고 시리얼 포트를 사용하여 손쉽게 프로그램을 지우고 써 넣을 수 있다. Javelin 은 Java 언어로 동작한다.

표 4. Javelin 의 구성 요소

	Javelin	MindStorm
Micro-controller	Ubicom SX48AC 20MHz	Hitache H8/3292
RAM	32 KByte	32 KByte
EEPROM	32 KByte	16 KByte
센서	터치센서 2 개	터치센서 2 개
서보 모터	2 개	2 개
개발 언어	Java	C++

이에 비해 LEGO MindStorm 은 Hitache 사의 Microcontroller 를 사용한다. 프로세서는 16K 의 내부 ROM 과 512Kbyte 의 RAM 을 갖는다. RCX 에는 또한 32Kbyte 의 추가 RAM 이 있다. LEGO 는 C++ 언어로 동작한다.

4.2.1 클래스 다이어그램

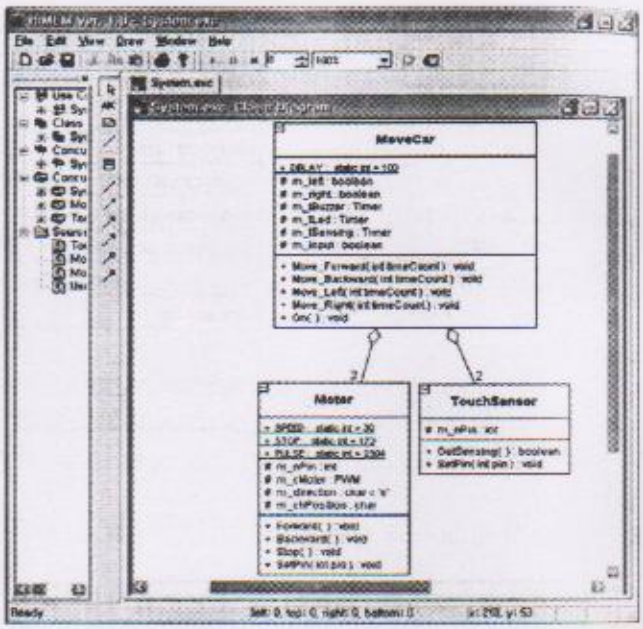

그림 8. 클래스 다이어그램

2 개의 터치센서에 반응하여 움직이는 자동차를 만들려고 한다. 그러기 위해서는 먼저 클래스 다이어그램을 이용하여 정적 구조를 설계한다. 그림 8 은 만들려고 하는 자동차를 클래스 다이어그램을 이용하여 표현한 것이다. 자동차의 객체로 구성될 수 있는 것은 Motor 와 TouchSensor 로 생각할 수 있다. Motor 클래스는 앞으로, 뒤로, 정지 3 가지 기능을 가지고 있지만 두 개의 Motor 가 모여서 전진, 후진, 좌, 우, 정지 5 가지 상태를 만들 수 있다. TouchSensor 는 현재 센서에 값을 전달하는 역할을 한다. Motor 와 TouchSensor 클래스를 MoveCar 클래스가 제어한다.

4.2.2 병렬 메시지 다이어그램

클래스 다이어그램으로 정적인 구조를 설계하였다면 병렬 메시지 다이어그램은 객체간의 상호작용을 모델링 한다. 그림 9 는 병렬 메시지 다이어그램의 표기법을 이용하여 표현한 것이다. 사용자로부터 전원이 켜지게 되면 자동차는 센서를 계속해서 점검하면서 값이 들어오는지를 체크한다. 초기에는 앞으로

만 간다. 그리고 왼쪽 센서에 물체가 감지되면 왼쪽 모터를 정지시키고 오른쪽 모터를 뒤로 후진하여 오른쪽으로 방향을 전환한 후 앞으로 전진한다. 오른쪽 센서에 물체가 부딪치면 오른쪽 모터를 정지시키고 왼쪽 모터를 뒤로 후진하여 왼쪽 방향으로 전환하여 앞으로 간다. 두 개의 센서에 동시에 부딪치면 뒤로 후진하였다가 오른쪽으로 방향은 전환하여 전진한다.

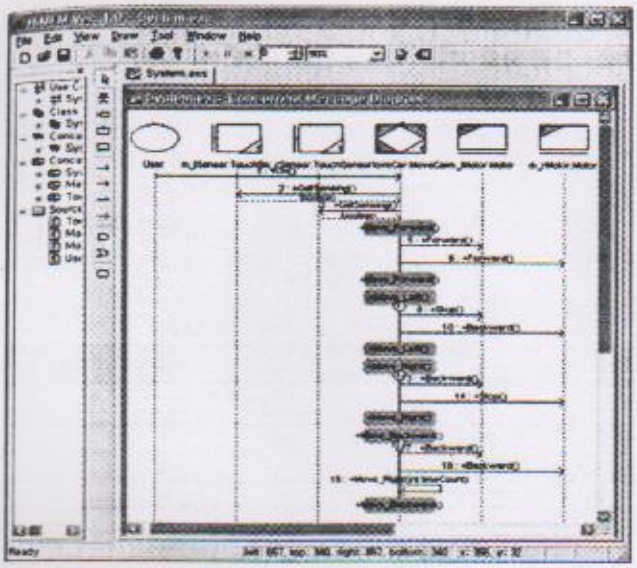

그림 9. 병렬 메시지 다이어그램

4.2.3 병렬 상태 다이어그램

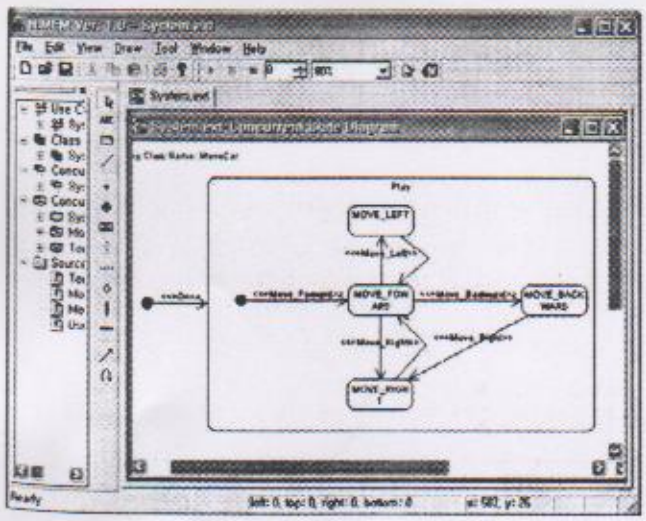

그림 10. 병렬 상태 다이어그램

병렬 상태 다이어그램은 각 기능들이 어떻게 수행되어야 하는지를 정의한다. 그림 10 은 MoveCar 클래스의 병렬 상태 다이어그램을 표현한 것이다. 사용자로부터 전원이 들어오면 On 이라는 함수를 호출하여 Play 상태로 전환된다. Play 상태는 Move_Forward 메시지가 들어오면 MOVE_FORWARD 상태로 전이된다. 이 상태에서 왼쪽 인지 오른쪽인지에 따라 상태가 전환되고 자동적으로 MOVE_FORWARD 상태로 다시 전환된다. 2 개의 센서에 동시에 부딪쳤을 때는 Move_Backward 메시지가 발생되고 MOVE_BACKWARD 상태로 전이된다. 이 상태에서 Move_Right 메시지를 발생시키고 MOVE_RIGHT 상태로 전이된 후 MOVE_FORWARD 상태로 전이된다.

4.2.4 코드 생성

클래스 다이어그램, 병렬 메시지 다이어그램, 그리고 병렬 상태 다이어그램을 이용하여 모델링을 완성한 후 그림 11 과 같이 붉은색 원으로 표시된 아이콘을 클릭하면 프로세서와 생성될 언어를 선택하는 대화상자가 나타난다. 그리고 원하는 프로세서와 언어를 선택한 후 Generate 버튼을 누르면 코드가 생성된다. 현재 Ubicom SX48AC 와 Hitache H8/3292 의 프로파일만 지원이 가능하다.

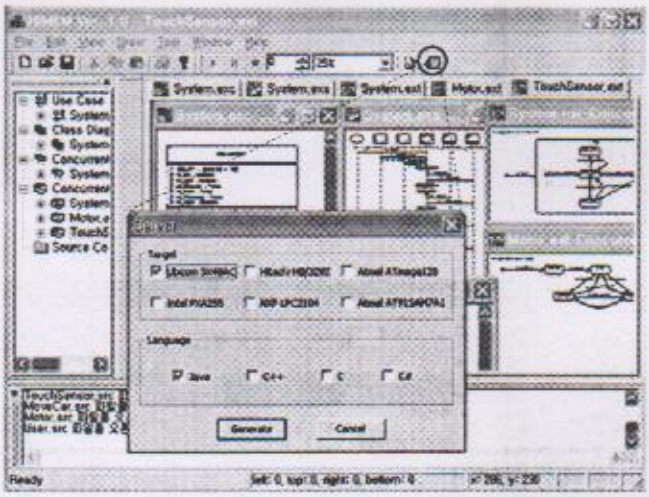

그림 11. 코드의 생성

그림 12 는 Java 언어를 선택하여 생성된 Javelin 의 코드이다. 도구에서 생성된 코드를 보면 Java 문법이 적용된 것을 확인 할 수 있다. 모터의 하드웨어 특성과 관련 있는 것은 m_nPin, SPEED, STOP, PULSE 이다. m_nPin 은 모터와 연결된 CPU 의 외부 핀 번호를 저장하는 곳이다. 서보모터를 구동하기 위해서는 일정한 주기 마다 신호를 받는데 그 값이 중간 값보다 많으면 앞으로 적으면 뒤로 돌아가는 특징을 가지고 있다. STOP 은 그 중간 값이고 SPEED 는 STOP 을 덧셈과 뺄셈을 통해 방향의 속도를 결정 짓게 된다. PULSE 는 신호를 주게 되는 주기 이다. 또한 서보모터가 놓이는 위치에 따라 방향이 틀려지는데 이것은 한쪽 서보모터를 180 도 회전하면 그 방향이 반

대가 되기 때문이다. 여기에 사용되는 변수가 m_chPosition 인데 상태 플래그를 'l' , 'r' 를 가진다. 'l' 은 Left, 'r' 은 Right 를 나타낸다. 마지막으로 m_direction 은 Motor 클래스의 전진, 후진, 멈춤 함수가 중복 실행되는 것을 방지하기 위해 현재 수행되고 있는 방향에 대한 상태를 가지고 있다. 's' 는 Stop, 'f' 는 Forward, 'b' 는 Backward 를 나타낸다.

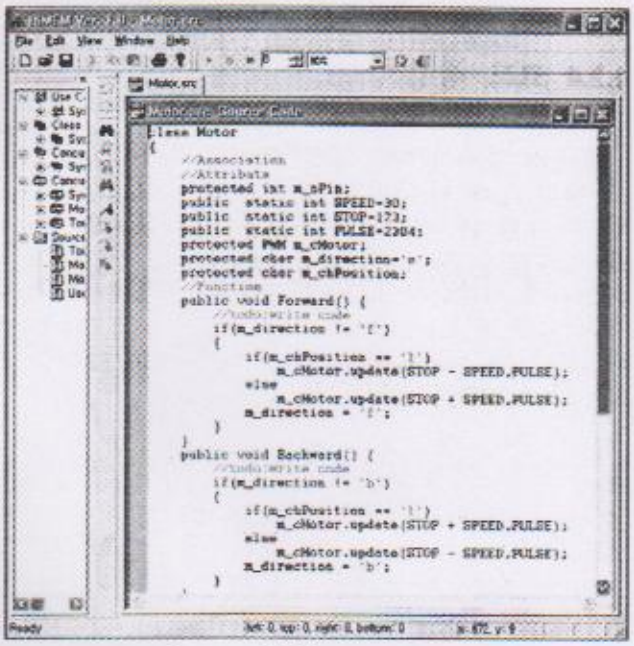

그림 12. 생성된 JAVA 코드

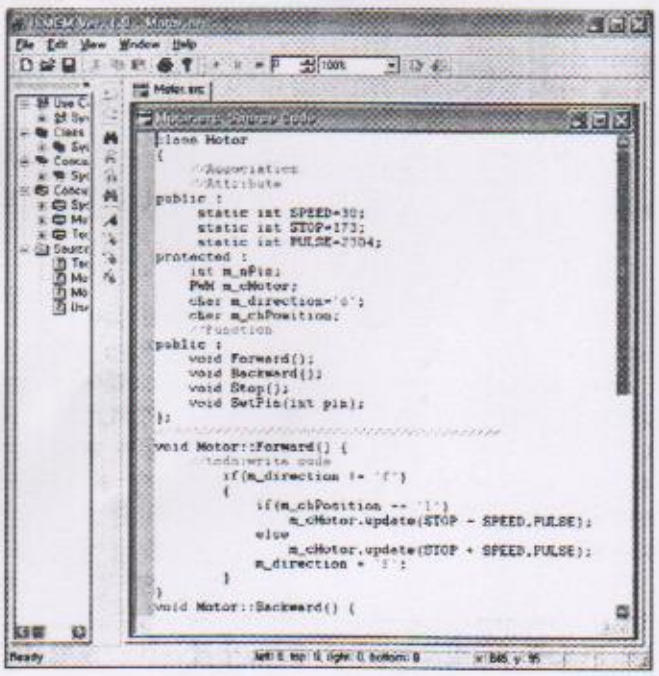

그림 13. 생성된 C++ 코드

그림 13 은 C++언어를 선택하여 생성된 MindStrom 의 코드이다. 도구의 생성된 코드를 보면 C++문법으로 생성된 것을 확인할 수 있다. 앞에서 살펴본 Java 코드와 비교하여 보았을 때 유사하다는 것을 확인 할 수 있다. 다른 점을 살펴보면 C++은 헤더파일과 구현 파일이 분리 시킬 수 있다. 그래서 "////////" 를 경계로 헤더와 구현을 분리 하였다. 또한 접근자가 그룹으로 지정되어 함수 마다 접근자가 표기되는 것이 아니라 ":" 키워드를 사용하여 같은 접근자를 가지는 함수를 한곳에 모아 두었다.

5. 비교 분석

표 5 는 구현한 도구의 장점을 파악하기 위해 다른 자동화 도구와 비교해 놓은 것이다.

표 5. 자동화 도구의 비교[14]

Capability	ROSE	Rhapsody	HiMEM v1.0
Primary Market	Commercial/business applications	Embedded, Real-time	Embedded
분석 능력	X	O	O
이벤트를 모델링 할 수 있는가	X	O	O
구현 단에 설계 변화를 분석할 수 있는가	X	O	O
설계 능력	O	O	O
Reverse Fork/Join 기법을 가지는가	X	X	O
동적 모델링에서 Concurrency 모델링이 가능한가	X	O	O
Nondeterministic 관점을 고려하는가	X	X	X
설계 변경이 쉬운가	X	X	O
여러 타겟에서의 설계 재사용이 가능한가	X	O	O
설계 단계에서 문제를 바로잡을 수 있는가	X	X	O
구현 능력	X	X	O
코드 생성	스켈레톤 코드	VM 기반	동적 요소를 포함한 완벽한 코드
모델과 코드가 연관되어져 있는가	X	O	O
실시간/임베디드 시스템의 구현방법을 가지고 있는가	X	O	O
테스트/디버그 능력	X	X	O
실행 가능한 모델인가	X	X	O
디자인 레벨에서 디버깅이 가능한가	X	X	O

자동화 도구는 크게 분석 능력과 설계 능력, 구현 능력, 그리고 테스트/디버깅 능력으로 나누어 비교할 수 있다. 우리가 개발한 도구(HiMEM v1.0)는 다른 도구와 비교해볼 때 우선 분석 능력에서 이벤트를 모델링하고 구현 전에 설계 변화를 분석할 수 있다는 장점이 있다. 설계 능력으로는 동적 모델링에서 중요시 되는 동시발생 문제(Fork-Join/ Reverse Frok-Join 등)를 표현하고 모델과 구현의 일관성이 유지된다는 장점이 있다. 또한 설계 변경이 용이하므로 여러 플랫폼(타겟)에서 재사용이 가능하고, 설계 도중 문제를 발견 및 수정할 수 있다. 구현 능력으로는 기존의 스켈레톤 코드가 아닌 동적 요소를 포함한 완벽한 코드가 발생된다는 것이다. 마지막으로 테스트/디버깅 능력은 xUML(Executable UML)[4,5,6,7]실행 가능한 모델이기 때문에 디자인 레벨에서의 디버깅이 가능하다.

6. 결 론

사용자 요구사항의 증가와 하드웨어의 발달에 따라 임베디드 소프트웨어 시스템도 복잡성이 증가되고 있다. 이렇게 급변하는 임베디드 시장에서 맞추어 소프트웨어를 빠르게 개발하기 위해서는 소프트웨어의 재사용은 필수다. 하지만 임베디드 소프트웨어는 하드웨어에 의존적이기 때문에 기 개발된 설계와 코드의 재사용이 어렵다고 한다. 본 논문에서는 임베디드 소프트웨어를 위한 코드생성 메커니즘을 제시하고 이를 이용하여 자동화 도구를 구현하였다. 이는 모델링과 코드 생성을 자동화하여 보다 빠르고 안정적으로 시스템을 개발할 수 있었다.

향후 연구로는 개발한 모델의 검증을 위한 도구가 필요하다. 그리고 현재 운영체제와 더욱 많은 프로세서가 지원될 수 있도록 프로파일에 관한 연구를 진행 중이다.

참고문헌

[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, 제 21 권, 제 1 호, 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", 한국정보처리학회, 춘계학술대회 논문집, Vol. 13, No. 1, 2006. 5.

[13] 김우열, 김영철, "확장된 xUML 을 사용한 MDA 기반 이종 임베디드 소프트웨어 컴포넌트 모델링에 관한 연구", 정보처리논문지, 제 14-D 권 제 1 호, 2007. 2.

[14] 조수란, 강성원, "소프트웨어 설계 툴의 코드자동생성능력 비교 연구", 한국정보과학회, 가을학술대회발표논문집, Vol. 33, No. 2(c), 2006.10.