KIPS
한국정보처리학회
제10권 제1호
Vol. 10 No. 1

2008

한국 소프트웨어공학 학술대회 논문집

Proceedings of 2008 Korea Conference on Software Engineering

- 일시 : 2008년 2월 20일(수)~22일(금)
- 장소 : 강원도 용평 리조트

주최 : 한국정보과학회
한국정보처리학회

주관 : 한국정보과학회 소프트웨어공학 연구회
한국정보처리학회 소프트웨어공학 연구회
한국전자통신연구원
KAIST 소프트웨어프로세스개선센터

후원 : (주)모아소프트, 삼성SDS(주), (주)쓰리케이소프트
(주)엔쓰리소프트, (주)텔레로직코리아

▶ 소프트웨어 설계 및 아키텍처

▶ 모델

이종 소형 무인 지상 차량 개발을 위한 MDA 기반 자동화 방법 연구†

김우열, 손현승, 김영철

홍익대학교 일반대학원 소프트웨어공학전공
충남 연기군 조치원읍 신안리 300
{john, son, bob}@selab.hongik.ac.kr

요약: 현재 임베디드 소프트웨어 구현 기술은 하드웨어에 종속되기 때문에 이종의 시스템 개발에서는 기 개발된 소프트웨어의 재사용이 매우 어렵다. 이런 문제점 때문에 기존에는 크로스 컴파일러나 미들웨어를 사용하고 있다. 하지만 크로스 컴파일러는 재사용성이 떨어지고 미들웨어는 메모리 공간적 문제점이 있다. 본 논문에서는 위의 문제를 해결하기 위해 기존 임베디드 시스템 개발 방법에 MDA(Model Driven Architecture) 메커니즘 개념을 접목한 자동화 방법을 제안한다. 제안한 자동화 방법은 임베디드 소프트웨어 개발에 적합하도록 변형된 UML(Unified Modeling Language) 프로파일을 사용하여 타겟 독립 모델에서 타겟 종속적 모델들로 변환하고 타겟 의존 코드를 자동으로 발생시킨다. 이 방법은 하나의 메타 모델 이용한 이종의 소프트웨어의 자동 개발 통해 개발기간의 단축과 신뢰성을 얻고자 하는 것이다. 적용사례로서 제안한 방법을 사용하여 이종의 소형 무인 지상 차량 시스템을 개발하였다.

핵심어: 통합 모델링 언어 UML, 모델 기반 아키텍처, 임베디드 소프트웨어, 소형 무인 지상 차량(SUGV: Small Unmanned Ground Vehicle)

1. 서론

소형 무인 지상 차량(SUGV: Small Unmanned Ground Vehicle)[1]은 사람이 들어갈 수 없는 곳이나 위험한 지역을 탐색하는 용도로 쓰고 있다. 이러한 특징 때문에 군사용, 우주용, 구조용으로 연구가 활발히 진행 중이다. 이 시스템은 하드웨어의 제약사항들인 프로세서의 속도, 메모리 용량, 크기의 한계를 가지고 있다. 뿐만 아니라 위험지역 탐색을 위해서는 정밀한 동작을 수행해야 된다. 그렇기 때문에 소프트웨어는 실시간으로 시스템의 동작을 정확히 제어해야 한다.

SUGV 의 활용이 늘어남에 따라 각각의 시스템을 위한 개발환경이 증가하고 상호운용성 문제[11]가 대두되고 있다. 이 결과 이기종 SUGV 시스템을 위한 소프트웨어 재사용성 문제를 해결할 방법이 필요하다. 소프트웨어 공학에서 MDA(Model Driven Architecture)[2,3]는 플랫폼에 독립적인 메타모델을 설계한 후 필요한 기술 모델을 변경하여 그 모델을 통해 코드 생성을 자동화하는 메커니즘이다. 이와 같은 방법으로 모델의 재사용과 관련된 코드의 생산성을 높일 수 있다.

하지만 기존의 MDA 는 원래 임베디드 시스템 개발에 적합하지 않다. 그러나 우리는 기존 임베디드 시스템 개발 메커니즘에 MDA 를 접목하여 자동화 방법을 제안한다. 제안한 방법은 자동화 도구를 통해 타겟 독립 모델을 만든 후 타겟 종속적 모델을 생성할 수 있고 타겟 의존 코드 역시 생성된다. 그리고 각 변환 단계는 UML 프로파일이 적용되어 자동 변환 된다. 이처럼 MDA 를 임베디드 시스템 개발에 적용한다면 개발시간 단축 및 이기종 소프트웨어의 재사용성 문제 해결이 가능할 것 이다.

본 논문의 구성은 다음과 같다. 2 장에서는 관련연구로 기존의 MDA 와 UML 프로파일에 대하여 알아본다. 3 장에서는 제안한 MDA 기반의 임베디드 소프트웨어 개발 자동화 방법을 기술한다. 4 장에서는 적용사례로 SUGV 시스템의 소프트웨어 개발을 보여준다. 마지막 5 장에서는 결론 및 향후 연구를 언급한다.

2. 기존의 모델 기반 아키텍처(Model Driven Architecture)

본 장에서는 관련연구로서 기존의 MDA[4,5,10]에 대해 알아본다. OMG(Object Management Group)에서 제안한 방법으로 S/W 개발 프로세스에 초점이 맞춰져 있다. 이는 모델을 기반으로 시스템이 구축되는 것을 묘사한다. 모델화 된 시스템은 모델과

† 본 연구는 산업자원부와 한국산업기술재단의 지역혁신인력양성사업(2007013011430)으로 수행된 연구결과임

모델간의 변환을 통해 하나 또는 다른 플랫폼으로 배포될 수 있다. 하지만 기존의 MDA 는 임베디드 시스템을 개발하기에 적합하지 않다. 그래서 본 논문에서는 기존 임베디드 시스템 개발 메커니즘에 MDA 를 접목하여 모델 변환 방법을 제안하였다.

2.1 기존 MDA 구성 요소

MDA 는 기존에 OMG 가 발표한 특정 플랫폼에 귀속되지 않은 플랫폼 독립적인 모델을 중심으로 개발과 시스템 통합 및 연동을 수행하는 프레임워크이다. MDA 개발 프로세스에서는 플랫폼에 독립적인 PIM 을 생성하고, PIM 에서는 플랫폼에 종속적인 여러 개의 PSM 으로 변환하며 최종적으로 코드로 변환한다. 모델간의 변환은 기존의 전통적인 개발 프로세스와 다를 것 없지만 MDA 에서는 모델이 UML 프로파일로 작성되기 때문에, PIM 에서 PSM 으로 변환이 모두 매핑 규칙에 의해 자동화 되므로 개발자의 노력을 최소화한다[8].

MDA 는 여러 가지 모델과 표준들에 의해서 구성된다. MDA 모델 들은 MOF 라는 추상적인 메타모델에 기초를 두고 있기 때문에 모두 연관되어 있다. 다시 말해 MDA 모델은 MOF 와 호환성이 있다. 이런 특징은 MDA 에서 이용된 모델들이 다른 MOF 호환 모델들과 호환성을 가진다는 것을 보장한다.

또 한 가지 중요한 부분이 UML 프로파일(profile)[10]이다. UML 프로파일은 UML 을 확장한 것이기 때문에 MOF 와 호환성이 있으며, UML 2.0 에서는 UML 의 다양한 기능적인 사용 방법을 기술할 수 있다. UML 프로파일은 UML 의 확장인 동시에 그 자체로 MOF 메타모델이다. 일부의 MDA 메타모델은 이미 OMG 에서 과거에 정의된 것들이며 일부는 OMG 외부의 그룹 또는 벤더들에 의해 MOF 호환이 되는 형태로 만들어진 것이다. 이러한 OMG 외부의 메타모델들은 비록 공식적인 OMG 표준은 아니지만 MOF 호환성이 있기 때문에 MDA 개발 과정에 사용되는 데 문제가 없다. 대표적인 MDA 모델들과 프로파일들로는 다음과 같은 것들이 있다.

UML 프로파일[10]은 특정 도메인에 대한 UML 모델을 작성하기 위해 일반 확장(generic extension) 메커니즘을 정의하는 것이다. 그러므로 UML 프로파일은 추가적인 스테레오타입(stereotype)과 태그 값(tagged value)이 적용된 엘리먼트(element), 어트리뷰트(attribute), 메소드(method), 링크(link) 등으로 이뤄진다. UML 프로파일은 이러한 확장 컬렉션을 이용해서 특정 도메인에 대한 모델링 문제를 기술하고 해당 도메인의 내용들을 모델링 할 수 있도록 해준다.

현재 CORBA 프로파일, EAI 프로파일, EDOC 프로파일, 리얼타임 컴퓨팅을 위한 스케줄링 프로파일 등이 OMG 에서 정의되었으며 EJB 프로파일은 JCP(Java Community Process)를 통해, 닷넷 프로파일은 마이크로소프트에 의해 정의되고 있다. 본 논문에서는 임베디드 소프트웨어에 적합하도록 운영체제와 프로세서의 속성을 정의한 프로파일을 작성하여 적용하였다.

UML 프로파일은 타겟 독립 모델뿐만 아니라 타겟 종속 모델을 표현하는 데에도 적용된다. 본 논문에서는 SUGV 시스템에 설계하기 위해 운영체제 프로파일과 프로세서 프로파일이 사용 된다.

2.2. MDA 와 Product Line 개발 프로세스 비교

근래 많은 사람들이 소프트웨어 공학[13]을 임베디드 소프트웨어 개발에 적용하려는 노력을 하고 있다. 표 1 에서는 여러 아이디어 중 대표적인 두 가지 개념인 프로덕트 라인[9]과 MDA 를 비교/분석 하였다.

표 1. 프로덕트 라인과 MDA 의 비교 [2]

Product Line	MDA
1) 한 번의 개발 라이프사이클 후에, 핵심 자산 재사용하는 기법 2) 고품질의 임베디드 시스템을 적시에 경제적으로 개발할 수 있는 *단일 제품군 기반 개발 방법* (Product-Line based Development) 3) Feature Driven 4) 커스터마이징 용이 5) UML 2.0 적용	1) 한 번의 개발 라이프사이클 중에, 하나의 메타 모델을 재사용하는 기법 2) 고품질의 임베디드 시스템을 적시에 경제적으로 개발할 수 있는 *이종 제품군 기반 개발 방법* (Heterogeneous Product based Development) 3) xUML + UML 2.0 적용 4) *코드 자동 생성기 필수*

프로덕트 라인과 MDA 를 비교했을 때, 가장 중요시 되는 차이점은 개발의 시기이다. 표 1 에서도 나타나듯, 프로덕트 라인은 한 번의 개발 라이프 사이클이 수행된 후에 생성된 핵심 자산을 재사용하여 새로운 시스템 개발이 가능하다. 이에 반해 MDA 는 한 번의 개발 라이프 사이클 중에 하나의 메타모델을 만든 후, 메타모델을 재사용하여 여러 개의 새로운 시스템 개발이 가능하다. 그리고 두 가지 모두 고품질의 임베디드 시스템을 적시에 경제적으로 개발할 수 있다는 점은 동일하지만 프로덕트 라인은 단일 제품군을 개발하기에 용이한 반면, MDA 는 이종의 제품군 개발에도 용이하다. 그러나 MDA 가 장점만을 가지고 있는 것은 아니다. 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.

우리는 이런 이유들로 MDA 를 임베디드 소프트웨어 개발에 적용하였다.

3. 제안한 MDA 기반 임베디드 소프트웨어 개발

제안한 방법은 임베디드 시스템을 개발하기에 적합하지 않은 MDA 메커니즘을 보완하여, 기존 임베디드 시스템 개발과 MDA 를 접목하였다. 제안한 MDA 기반 변환 프로세스는 타겟 독립 모델(TIM), 타겟 종속 모델(TSM), 타겟 의존 코드(TDC)의 3 가지 단계를 가진다. 타겟 독립적인 모델은 어떠한 모델에도 의존하지 않는 메타모델이다. 요구사항 분석을 거친 후 확장된 xUML[3]을 이용하여 타겟 독립 모델을 설계한다. 이 때 UML 프로파일이 적용된다. 본 논문에서의 타겟 종속 모델은 특정 하드웨어나, OS 에 의존적인 모델을 말한다. 타겟 독립 모델로부터 자동 변환된 타겟 종속 모델에 적합한 기능을 추가하여 완성된 모델을 만든다. 타겟 의존 코드는 타겟 종속 모델로부터 생성된 코드이다. 최종적으로 완성된 이 코드를 컴파일 하여 타겟 시스템에 다운로드 하게 된다.

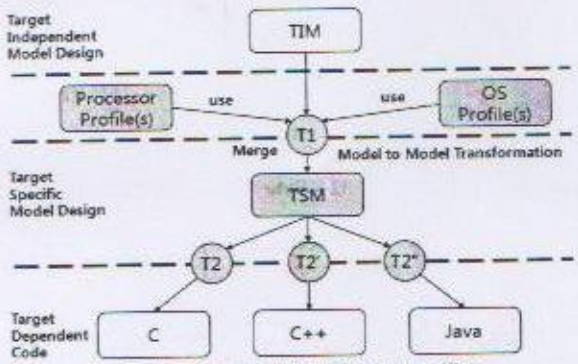


그림 1. MDA 기반 변환 프로세스

그림 1 과 같이 MDA 기반의 개발 자동화 방법은 T1, T2 의 두 가지 변형 단계로 구성된다. T1 단계에서는 미리 정의된 프로세서 프로파일과 운영체제 프로파일을 병합하여 타겟 독립 모델이 타겟 종속 모델로 변환된다. 예를 들어 적용사례로 사용할 임베디드 시스템은 프로세서 프로파일은 Hitachi H8 이고 운영체제 프로파일은 brickOS 이다. T2 단계에서는 타겟 시스템에 종속적인 모델(TSM)이 타겟 의존 코드(TDC)로 변환된다. 생성 가능한 언어는 C, C++, Java 이다. 코드는 모델을 텍스트로 변환하는 방법[6]을 이용하여 자동 생성된다.

3.1. 타겟 독립 모델의 타겟 종속 모델 변환 단계(T1)

적용하려는 SUGV 시스템의 구조도는 그림 2 와 같이 4 계층의 레이어 아키텍처[13]로 구성된다. 각각은 어플리케이션 레이어(AL), 서비스 레이어(SL), 운영체제 레이어(OSL), 프로세서 레이어(PL)이다. 본 논문에서 예제로 삼은 SUGV 시스템은 메모리 용량 상 미들웨어를 탑재할 수 없었지만 추후 시스템에는 미들웨어를 적용하려고 한다.

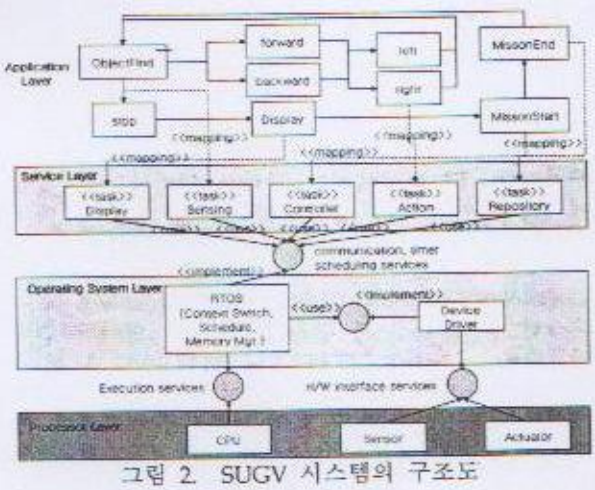


그림 2. SUGV 시스템의 구조도

각 레이어별로 살펴보면 어플리케이션 레이어는 최상위 레이어로 서비스들을 이용하여 시스템을 모델링한다. 어플리케이션 레이어의 행위들은 서비스 레이어의 서비스들과 직접 매핑 된다. 사용자는 서비스 레이어의 서비스들을 사용하여 모델링 하게 된다. 이것은 사용자가 쉽게 가능한 서비스들 사용할 수 있도록 도와주는 인터페이스 역할을 해준다.

서비스 레이어는 운영체제 레이어와 프로세서 레이어의 연결자 역할을 한다. 이 레이어 에서 사용자에게 서비스를 제공해 주게 된다.

운영체제 레이어는 크게 RTOS 의 영역과 디바이스 영역으로 나뉜다. RTOS 영역은 운영체제의 API(Context switching, Scheduling, Memory Management, Timer Service 등)로 구성된다. 이 레이어의 주된 관심사항은 시스템 리소스의 최적화된 활용에 있다. 그리고 디바이스 드라이버는 다양한 하드웨어로부터 시스템이 독립적으로 구성될 수 있도록 한다. 디바이스 드라이버를 사용하여 다양한 하드웨어의 이식성을 높일 수 있다. 이들의 실제적인 구현은 프로세서 레이어에서 이루어진다.

마지막으로 프로세서 레이어는 4 계층 중 최하위 레벨로 하드웨어에 의존적이다. 하드웨어를 작동시키기 위한 부분으로 시스템 작동을 위해 꼭 필요한 레이어이다. 그래서 각 프로세서마다

개별적으로 생성된다.

이러한 구성정보를 UML 프로파일로 만들어 데이터 저장소에 저장하면 T1 단계에서 변환규칙에 의해서 시스템에 맞는 모델로 변환하여 준다. 본 논문에서는 UML 프로파일에 관한 자세한 내용은 지면상 다루지 않는다.

모델 변환의 중요한 요소는 변환 규칙이다. 변환규칙은 UML 프로파일에 있는 공통 매개체를 통해 각 시스템 상황에 맞도록 변경하여 준다.

3.2. 타겟 종속 모델의 타겟 의존 코드 변환 단계(T2)

T1 단계를 통해 생성된 타겟 종속 모델을 실제 수행 가능한 코드로 변환하기 위해서 T2 단계가 수행된다. T2 단계에서는 클래스 다이어그램, 병렬 메시지 다이어그램, 병렬 상태 다이어그램의 메타 모델을 사용하여 언어를 생성 시킨다.

메타모델을 통해서 다이어그램들의 정보들을 모두 저장한 후 그림 3 과 같은 형태의 각 언어별 코드 템플릿을 사용하여 T2 단계의 변환을 하게 된다. 결론적으로 타겟 종속 모델을 메타모델을 사용하여 데이터를 수집하고 그림 3 의 템플릿을 이용하여 T2 단계의 변환을 수행하여 소스코드를 자동으로 생성한다.

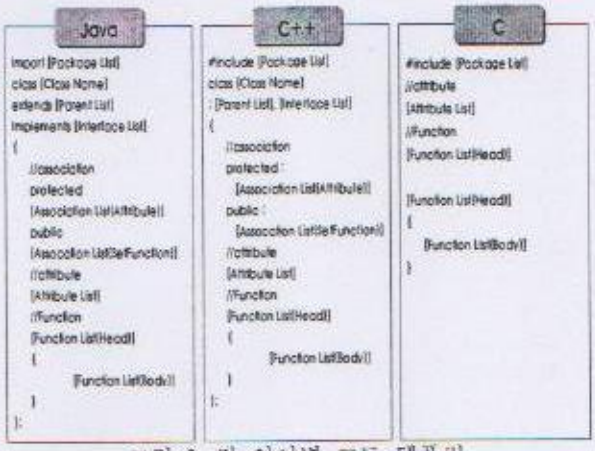
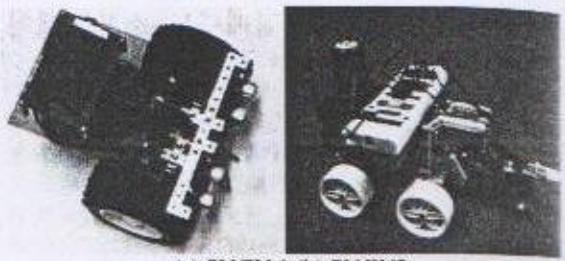


그림 3. 각 언어별 코드 템플릿

4. 적용 사례

그림 4 의 (a)와 (b)는 본 논문에서 예제로 설정한 2 개의 바퀴로 움직이고 듣고 다닐 수 있게 설계된 이종의 SUGV 이다. 이는 어떠한 환경에서든 사람 없이 지형지물을 탐색 가능하게 해 준다. 그리고 향상된 상황 인식 장치로 감시 지역 정찰이 가능하다. 또한 사용자에게 목표 시스템의 관찰을 제공하거나 시야를 벗어난 지역의 탐색을 수행할 수 있다.



(a) SUGV 1 (b) SUGV2
그림 4. 이종의 SUGV

SUGV 는 텍스트 LCD 1 개, 초음파 센서 2 개, 로데이션 서보 모터 2 개, 바퀴 2 개로 구성되어 있다. 텍스트 LCD 는 현재 시스템이 어떠한 상태인지 모니터링 할 수 있게 해주고 센서의 정보를 디스플레이 해준다. 초음파 센서는 대상 물체와의 거리를 측정하여 목표물의 위치 정보를 제공해준다. 서보 모터는 차량을 전, 후, 좌, 우 방향으로 이동할 수 있도록 해준다.

표 2. 이종 SUGV 하드웨어 정보

구분	(a) SUGV1	(b) SUGV2
Microcontroller	Ubicom SX48AC 20MHz	Hitachi H8/3292 16MHz
OS	Javeline	brickOS
RAM	32 KByte	512KByte
EEPROM	32 KByte	16KByte
Sensor	초음파센서 2 개	빛 센서 2 개
Display	Text LCD	Text LCD
Motors	2 개	2 개
JVM	하드웨어	없음
Languages	Java	C/C++

이종의 SUGV 의 하드웨어 정보는 표 2 와 같다. SUGV1 시스템의 프로세서는 Javeline™[7]이 Ubicom SX48AC 에 JAVA 인터프리터를 내장시켜 JAVA 언어를 사용할 수 있도록 만든 제품이다. RAM 과 ROM 의 크기가 32KByte 로 일반시스템에 비하여 매우 작다. 이러한 요소가 시스템 개발의 제약사항으로 요구된다. SUGV2 시스템은 LEGO MindStorms™[12]으로 Hitachi H8 프로세스

사용하고 C 언어 또는 C++로 개발할 수 있는 환경이 제공된다.

T1 단계에서는 타겟 독립 모델을 타겟 종속 모델로 변환한다. 도구를 통하여 타겟 독립 모델을 생성한 다음 타겟 종속 모델으로 변환을 시도한다. 이때 UML 프로파일화 된 프로세서 프로파일과 운영체제 프로파일을 선택하여 변환하게 된다.

Javeline[7]은 Parallax 에서 만든 자바 칩으로 그림 5 와 같이 마이크로 컨트롤러 위에서 하드웨어적으로 자바 바이트 코드를 해석하여 준다.

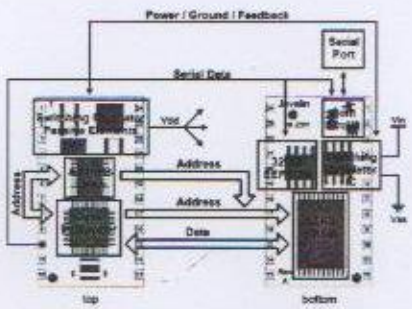


그림 5. Javeline 블록 다이어그램

하지만 기존의 자바와 같은 기능이 되지 않고 아래와 같은 몇 가지 다른 특징을 가지고 있다.

- 단일 쓰레드
- 가비지 컬렉션 없음
- 데이터 타입의 부분집합
- 자바 라이브러리의 부분집합
- ASCII 문자열
- 인터페이스 없음
- 일 차원 배열

기존의 자바의 기능을 사용하지 못하지만 OS 처럼 라이브러리와 API 를 가지고 있기 때문에 OS 로 분류 하였다.

이렇게 각각의 하드웨어 환경에 맞는 프로세서와 운영체제를 선택하여 독립적인 모델을 종속적인 모델로 변환하게 된다. 그리고 본 논문에서 제시한 방법을 적용하여 개발중인 도구에서는 생성한 모델에 대한 검증을 자동으로 하여 사용자의 실수로 인한 오류를 줄여주는 기능이 있다. 또한 병렬 상태 다이어그램의 각 상태를 수행 하여 논리적인 오류가 발생할 수 있는 부분을 체크할 수 있도록 해준다. 병렬 순차 다이어그램과 같이 다이어그램들의 요소를 검사하여 오류 메시지를 출력 창에 보여준다.

T2 단계에서는 타겟 종속 모델을 타겟 의존 코드로 변환 한다. 타겟 종속적인 모델을 설계한 후 각 시스템에 맞는 코드 템플릿을 통하여 이종의 코드를 생성한다. 모든 과정은 자동화 도구를 이용하여 변환된다.

표 3. 도구를 통해 생성된 이 기종의 시스템 코드

SUGV1(Javeline)	SUGV2(MindStorms)

public void Forward()
{
if(m_chPosition == 'f')
CPU.pulseOut(STOP - m_speed,m_nPin);
else CPU.pulseOut(STOP + m_speed,m_nPin);
m_direction = 'f';
}
public void Backward()
{
if(m_chPosition == 'f')
CPU.pulseOut(STOP + m_speed,m_nPin);
else
CPU.pulseOut(STOP - m_speed,m_nPin);
m_direction = 'b';
}
public void Stop() {
CPU.pulseOut(STOP,m_nPin);
m_direction = 's';
}
	public void Forward()
	{
	if(m_chPosition == 'f')
	motor.forward(m_speed);
	else motor.reverse(m_speed);
	m_direction = 'f';
	}
	public void Backward()
	{
	if(m_chPosition == 'f')
	motor.reverse(m_speed);
	else
	motor.forward(m_speed);
	m_direction = 'b';
	}
	public void Stop() {
	motor.brake();
	m_direction = 's';
	}

표 3 은 생성된 코드 중 일부인 Motor 클래스의 Forward(), Backward(), Stop() 함수 이다. Motor 클래스는 서보모터를 작동시키는 클래스로 한 개의 모터와 연결되어 있다. 그렇기 때문에 좌측 모터와 우측모터를 동작시키는 코드가 같다면 서로 반대 방향으로 돌아가게 된다. 그래서 m_chPosition 변수를 이용하여 각자의 위치 상태를 저장하도록 하였다. 도구를 통하여 생성된 3 개의 함수를 비교하여 서로 다른 부분은 밑줄을 이용하여 표시 하였다. 코드를 통해 이종의 시스템에 맞도록 변환된 것을 확인 할 수 있다.

5. 결론

현재 임베디드 소프트웨어 구현 기술은 하드웨어에 종속되기 때문에 이종의 시스템 개발에서는 기 개발된 소프트웨어의 재사용이 매우 어렵다. 본 논문에서는 이 기종 임베디드 시스템 개발 시 소프트웨어의 재사용성 문제를 해결하기 위해 기존 임베디드 시스템 개발 에 MDA(Model Driven Architecture) 메커니즘 개념을 접목한 방법을

제안하였다. 이 방법은 자동화 도구를 통해 타겟 독립 모델을 만든 후, 특정 임베디드 시스템의 환경으로 자동 변환을 위해 UML 프로파일이 사용되어 타겟 종속적 모델들로 변환되고 타겟 의존 코드 또한 개발하는 것이다. 제안한 방법의 적용사례로서 SUGV 시스템을 설계하고 단계별로 하드웨어 및 OS 의 프로파일을 적용해 이종의 코드를 생성하였다.

향후 연구로는 개발한 도구를 통해 설계한 모델의 자동 문서화와 모델 시뮬레이션 기능이 필요하다. 그리고 현재 운영체제와 더욱 많은 프로세서가 지원될 수 있도록 프로파일에 관한 연구를 진행 중이다.

참고문헌

[1] Future Combat System, Small Unmanned Ground Vehicle(SUGV), http://army.mil/fcs/sugv.html/

[2] W. Kim, R. Y. Kim, "Adapting Model Driven Architecture for Modeling Heterogeneous Embedded S/W Components," ICHIT2006, Vol. 2, 2006. 11.

[3] 김우열, 김영철, "A Study on Modeling Heterogeneous Embedded S/W Components based on Model Driven Architecture with Extended xUML, " KIPS Trans., Vol. 14-D, No. 1, 2007. 2.

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

[5] Dong Ho Kim, Woo Yeol Kim, Young Chul Kim, "A Study on Design for Embedded S/W based on Model Driven Architecture," IWIT, Vol. 6, no. 1, 67-74, 06.03.

[6] OMG, MOF Models to Text Transformation Language final adopted specification, OMG Document ptc/06-11-01

[7] Parallax, Javelin Stamp Manual Version 1.0, http://www.parallax.com/, 2002 .

[8] Pierre Boulet, Jean-Luc Dekeyser, Cedric Dumoulin, and Philippe Marquet, "Mda for Soc Design, Intensive Signal Processing Experiment," In FDL'03, Frankfurt, September 2003. ECSI.

[9] Kyo C. Kang, Jaejoon Lee, and Patrick Donohoe, "Feature-Oriented Product Line Engineering", IEEE Software, Vol. 9, No. 4, Jul./Aug. 2002, pp.58-65.

[10] OMG, UML Superstructure, v2.1.1, OMG document formal/07-02-03 .

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

[12] LEGO, Mind Storms, http://mindstorms.lego.com/

[13] B. Bruegge, A. H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns, and Java(2nd Edition), Prentice Hall, 2004