제35권 제12호 2008년 12월 ISSN 1229-6848

정보과학회논문지 : 소프트웨어 및 응용

Journal of KIISE : Software and Applications

VOLUME 35, NUMBER 12, DECEMBER 2008

UML 프로파일 메커니즘을 이용한 이종 소형 무인지상차량 설계 자동화

(Design Automation for Heterogeneous SUGVs with UML Profile Mechanism)

김 우 열 † 손 현 승 †† 김 영 철 †††
(Woo Yeol Kim) (Hyun Seung Son) (R. Young Chul Kim)

요 약 SUGV의 활용이 늘어감에 따라 구성하는 소프트웨어가 복잡해지고 개발 환경 다양화로 인한 상호운용성 문제가 대두되고 있다. 본 논문에서는 이러한 문제점을 해결하고자 기존의 UML 프로파일 메커니즘을 이용하여 SUGV 개발에 MDA가 적용되도록 하였다. 이를 통해 이종 SUGV 소프트웨어 설계시 타겟 독립 모델을 만든 후 UML 프로파일이 적용된 자동화 도구를 이용해 타겟 종속 모델 및 코드를 생성하면 이종 기기의 개발기간과 노력을 절약할 수 있다. 그리고 발생된 코드의 분석을 통해 제안한 방법의 이점 및 신뢰성을 확인할 수 있었다.

키워드 : 소형 무인 지상 차량, 모델 지향 아키텍처, 통합 모델링 언어, 임베디드 소프트웨어

Abstract Today raises its head on the issue of interoperability caused by the complexity of the embedded software and the diverse development environment about SUGV(Small Unmanned Ground Vehicle). So, we propose to adopt the original MDA mechanism for this heterogeneous embedded development. To solve this problem, we apply for developing SUGV with MDA(Model Driven Architecture) using the original UML profile mechanism. Through this method, it can be semi-automatically transformed into TSM(Target specific model) after modeling TIM(Target Independent Model). Then we can also automatically generate the heterogeneous source codes. Therefore it will be reduced the development cycle and effort of the heterogeneous systems. We verify the benefits of our proposed approach and the reliability through analyzing the generated codes.

Key words : Small Unmanned Ground Vehicle, Model Driven Architecture, Unified Modeling Language, Embedded Software

1. 서 론

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, 재난구조용 등으로 다목적으로 사용된다. 이 시스템은 크기가 작고 모바일 형태의 시스템으로 하드웨어의 제약사항들인 프로세서의 속도, 메모리 용량, 배터리 용량의 한계를 가지고 있다. 뿐만 아니라 위험지역 탐색이나 인명구조를 위해서는 정밀한 동작을 수행해야 된다. 그렇기 때문에 소프트웨어는 실시간으로 시스템의 동작을 정확히 제어할 수 있어야 하고 소프트웨어의 신뢰성이 요구된다[2].

SUGV의 활용이 늘어남에 따라 구성하는 소프트웨어가 복잡해지고 개발환경 다양화로 인한 상호운용성 문제가 대두되고 있다. 이 결과 이기종 SUGV 시스템을

· 본 연구는 교육과학기술부와 한국산업기술재단의 지역혁신인력양성사업(2008~2009)으로 수행된 연구결과임

† 정 회 원 : 홍익대학교 전자전산공학과
john@selab.hongik.ac.kr
†† 학생회원 : 홍익대학교 전자전산공학과
son@selab.hongik.ac.kr
††† 정 회 원 : 홍익대학교 컴퓨터정보통신공학과 교수
bob@selab.hongik.ac.kr

논문접수 : 2008년 5월 21일
심사완료 : 2008년 10월 31일

정보과학회논문지: 소프트웨어 및 응용 제35권 제12호(2008.12)

위한 소프트웨어 재사용성 문제를 해결할 방법이 필요하다. 소프트웨어 공학에서 MDA(Model Driven Architecture)[3,4]는 독립적인 플랫폼을 사용하여 의존적인 플랫폼으로 자동변환 하여 상호운영성의 문제 해결이 가능하다.

본 논문에서는 이러한 문제점을 해결하고자 기존의 UML 프로파일[5] 메커니즘을 이용하여 SUGV 개발에 MDA가 적용되도록 하였다. 이는 이종 SUGV 소프트웨어 설계시 타겟 독립 모델을 만든 후 UML 프로파일이 적용된 자동화 도구를 이용해 타겟 종속 모델 및 코드를 생성하는 것이다. 이러한 자동화 방법은 개발 라이프 사이클을 줄여 이종 기기의 개발시간과 노력을 단축시킬 수 있다. 그리고 발생한 코드의 개발 기간 및 품질 분석을 통해 제안한 방법의 이점 및 신뢰성을 확인할 수 있다.

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

2. 관련연구

2.1 MDA 기반 임베디드 소프트웨어 개발 방법론

대표적인 임베디드 소프트웨어 개발방법론으로는 ILogix의 Harmony 프로세스와 한국전자통신연구원의 EMMA(Embedded MARMI)가 있다. 표 1에 Harmony와 EMMA, HiMEM의 특징 및 장단점을 비교하였다[6].

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 자산을 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 자동변환을 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.

2.2 MDA 기반 임베디드 소프트웨어 개발프로세스

임베디드 시스템을 개발하기에 적합하지 않은 MDA 메커니즘을 보완하여, 기존 임베디드 시스템 개발과 MDA를 접목하였다. MDA 기반 변환 프로세스는 타겟 독립 모델(TIM), 타겟 종속 모델(TSM), 타겟 의존 코드(TDC)의 3가지 단계를 가진다. 타겟 독립적인 모델은 어떠한 모델에도 의존하지 않는 메타모델이다. 요구사항 분석을 거친 후 확장된 xUML[6]을 이용하여 타겟 독립 모델을 설계한다. 이 때 UML 프로파일이 적용된다. 본 논문에서의 타겟 종속 모델은 특정 하드웨어나, OS에 의존적인 모델을 말한다. 타겟 독립 모델로부터 자동변환된 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 1과 같이 MDA 기반의 개발 자동화 방법은 T1, T2의 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[7]이고 운영체제 프로파일은 brickOS[8]이다. T2 단계에서는 타겟 시스템에 종속적인 모델(TSM)이 타겟 의존 코드(TDC)로 변환된다. 생성 가능한 언어는 C, C++, Java이다. 코드는 모델을 텍스트로 변환하는 방법[9]을 이용하여 자동 생성된다.

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***

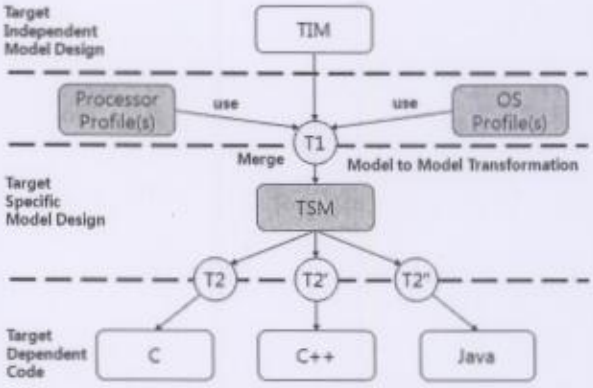

그림 1 MDA 기반 변환 프로세스

2.3 UML프로파일

UML[10]은 매우 일반적인 언어이기 때문에, 특정한 프로그래밍 언어(Java, C++, 등)의 개념을 표현하거나 혹은 특정한 애플리케이션 도메인(금융, 항공우주, 전자상거래, …)의 개념을 표현하기에는 부족하다. 이런 경우 스테레오타입과 확장 속성을 사용할 수 있는데, 이것을 체계적인 형태로 정의해서 하나의 패키지로 만든 것이 표 2의 UML 프로파일이다.

표 2 UML 프로파일 그래픽 표기법

Node Type	Notation
Stereotype	«stereotype» Name
Metaclass	«metaclass» Name
Profile	«profile» Name
Extension	——▶
ProfileApplication	«apply» - - - - - ▷

UML 프로파일은 특정 요소를 여러 가지 관점에서 확장할 수 있도록 해 준다. 스테레오타입(stereotype)을 통해 요소들을 분류할 수 있는 기준을 제공하고, 확장 속성(tagged value)을 통해 요소에 정의되지 않은 또 다른 속성을 정의할 수 있도록 도와준다. 그리고 제약조건(constraint)과 데이타타입(data type)을 추가적으로 정의할 수 있도록 허용한다.

3. 개선된 UML 프로파일

UML을 확장하기위해 스테레오타입과 확장 속성을 사용할 수 있는데, 이것을 체계적인 형태로 정의해서 하나의 패키지로 만든 것이 UML 프로파일(Profile)이다. UML 프로파일을 사용하면 기존의 UML을 다양한 도메인에 맞게 확장 할 수 있다. 하지만 UML 프로파일만으로는 SUGV 시스템을 프로파일화 하기는 힘들다. 본 논문에서는 UML 프로파일처럼 확장개념과 속성을 저장하지만 SUGV 시스템에 맞도록 다른 구조를 제안한다. 이것을 메타 프로파일이라고 부른다.

3.1 제안한 메타구조

타겟 독립 모델이 타겟 종속 모델로 변환 될 때 UML 프로파일 메커니즘이 사용된다. 다양한 하드웨어 및 운영체제에 적용되기 때문에 이들을 통합 할 수 있는 메타 모델이 필요하다. 이를 메타 프로파일이라고 부르고 그림 2와 같다.

특징은 각각의 운영체제나 프로세서에 따라서 따로 구성되는 것이 아니라 하나의 통합된 모델로 구성되는 것이다. 또한 이들 간의 관계를 유지하기 위해서 미들웨어, RTOS, 프로세서 3레이어 계층으로 분할하였다. 최상위 계층은 미들웨어로 미들웨어에 대한 목록과 미들웨어가 지원 가능한 RTOS 목록들이 들어간다. 다음 계층은 RTOS로 RTOS의 목록과 지원 가능한 프로세서 목록이 들어간다. 프로세서에는 목록과 프로세서의 블록 이미지와 프로세서의 정보 그리고 변환된 API 함수 리스트가 들어간다. 이것은 프로파일을 선택할 때 영향을

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:complexType name="targetTagType">
    <xs:sequence>
      <xs:choice>
        <xs:element name="satrtTag" type="xs:string"/>
        <xs:element name="endTag" type="xs:string"/>
      </xs:choice>
      <xs:element name="contents" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="processorType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="img" type="xs:anyURI"/>
      <xs:element name="information" type="xs:string"/>
      <xs:element name="targetTag" type="targetTagType" minOccurs
 ="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="rtosType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="processor" type="processorType"
 maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="MiddlewareType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="rtos" type="rtosType" maxOccurs="
 unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="MetaProfileType">
    <xs:sequence>
      <xs:element name="Middleware" type="MiddlewareType"
 maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="MetaProfile" type="MetaProfileType"/>
</xs:schema>
```

그림 2 메타 프로파일의 구조

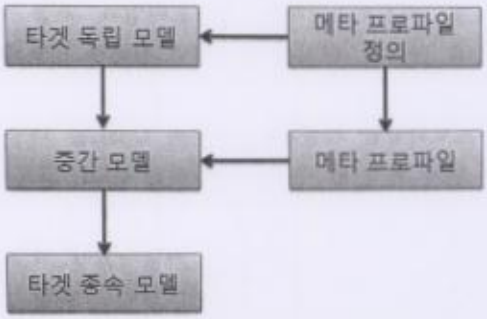

그림 3 제안한 메타 프로파일의 적용

미치게 된다.

메타 프로파일은 타겟 독립 모델에서 타겟 종속 모델로 변환 될 때 사용된다. 그림 3은 제안한 메타 프로파일이 적용되는 것을 도식화 한 것이다. 그림에서 보는 것과 같이 메타 프로파일은 XML형태로 정의하고 정의된 메타프로파일은 타겟 독립 모델에서 사용이 가능하다. 그리고 정의된 메타 프로파일의 내용은 모델변환시 중간 모델에서 불려와 정의된 메타프로파일의 내용을 생성하게 된다. 이때 메타프로파일에는 사용자가 선택한 하드웨어에 맞는 구현부가 생성되어 타겟 종속 모델로 변하게 되는 것이다.

3.2 기존 도구에 적용

제안한 메타 프로파일을 기존 도구에 적용하기 위해서는 그림 4와 같은 절차를 수행해야 된다. 먼저 파일 형태로 되어있는 메타 프로파일을 읽어 문장 해석과 정의된 요소들을 분리한다. 메타 프로파일은 XML을 사용하기 때문에 DOM 파서를 이용하여 문장을 해석하고 요소를 추출한다. 저장된 요소는 미들웨어, 운영체제, 하드웨어로 구분된다. 추출한 미들웨어, 운영체제, 하드웨어를 분석하여 프로파일 테이블을 생성하고 프로파일을 선택할 수 있는 정보를 보여주는 다이얼로그에 출력시켜주게 된다. 사용자가 하드웨어 및 운영체제를 선택하면 프로파일 테이블에서 변환될 요소들을 가져와 작업을 처리하게 된다.

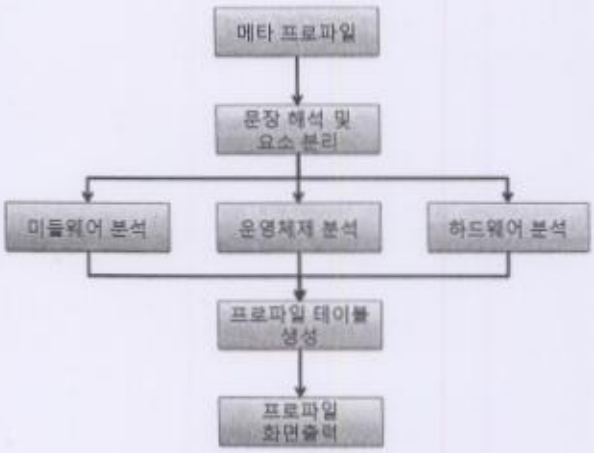

그림 4 메타 프로파일 로드 절차

메타 프로파일을 로드절차를 통해 메타프로파일을 로드하고 이것을 도구에 적용하여 개발하였다.

그림 5는 도구 상에 적용된 메타 프로파일이다. 앞서 살펴본 내용과 같이 미들웨어, RTOS, 프로세서 3계층의 레이어로 나누어지고 그림에서 각각 ①, ②, ③에 해당 된다.

①은 미들웨어로 최상위 단계이고 현재 적용된 모델이 미들웨어가 없기 때문에 "none"으로 표시되어 있다. 프로파일 구조에서 최상위단계는 한개 필요하기 때문에 이것을 "none"으로 표시하여 준 것이다.

②는 RTOS로 중간 단계이고 현재 "brickOS", "uCOS-II"를 선택할 수 있다. 이것은 미들웨어의 "none"의 그룹에 "brickOS", "uCOS-II"를 지원할 수 있는 RTOS들이 들어 있다는 의미이다.

③은 프로세서로 최하위 단계이고 이 단계에 많은 정보들이 들어가게 된다. 현재 "Hitachi h83292"가 표시되어 있는데 이것은 미들웨어가 "none"이고, RTOS가 "brickOS"인 그룹에 가능한 프로세서 목록을 나타낸다. 그리고 여기서 프로세서를 체크하게 되면 그림의 ④와 같이 프로세서의 외부핀 정보와 하드웨어 스펙 정보를 제공하게 된다.

타겟 독립 모델에서 종속 모델로 변환하기위해서는 위에서 설명한 미들웨어, RTOS, 프로세서가 모드 체크되어있어야 변환이 가능하다. 3가지 모두 선택이 되면 프로파일 메타구조에 있는 태그들이 활성화가 되어 모델 변환기에 의해서 이러한 태그 정보들이 타겟에 맞는 정보로 변환이 된다.

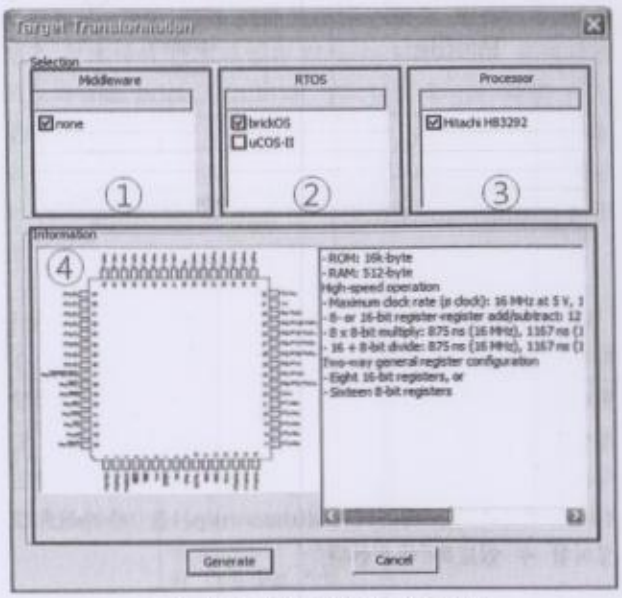

그림 5 도구에 적용된 프로파일

4. 적용사례

4.1 이종 SUGV(Small Unmanned Ground Vehicle) 시스템

그림 6의 (a)와 (b)는 본 논문에서 예제로 설정한 2개의 바퀴로 움직이고 들고 다닐 수 있게 설계된 이종의 SUGV이다. 이는 어떠한 환경에서든 사람 없이 지형지물을 탐색 가능하게 해 준다. 그리고 향상된 상황 인식 장치로 감시 지역 정찰이 가능 하다. 또한 사용자에게 목표 시스템의 관찰을 제공하거나 시야를 벗어난 지역의 탐색을 수행할 수 있다.

SUGV는 텍스트 LCD 1개, 초음파 센서 2개, 로테이션 서보 모터 2개, 바퀴 2개로 구성되어 있다. 텍스트 LCD는 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.

이종의 SUGV의 하드웨어 정보는 표 3과 같다.

SUGV1 시스템의 프로세서는 JavelineTM[11]으로 Ubicom SX48AC에 JAVA 인터프리터를 내장시켜 JAVA 언어를 사용할 수 있도록 만든 제품이다. RAM과 ROM의 크기가 32KByte로 일반시스템에 비하여 매우 작다. 이러한 요소가 시스템 개발 시 제약사항으로 요구된다. SUGV2 시스템은 LEGO의 MindStormsTM[12]으로 Hitachi H8 프로세스를 사용하고 C언어 또는 C++로 개발할 수 있는 환경이 제공된다.

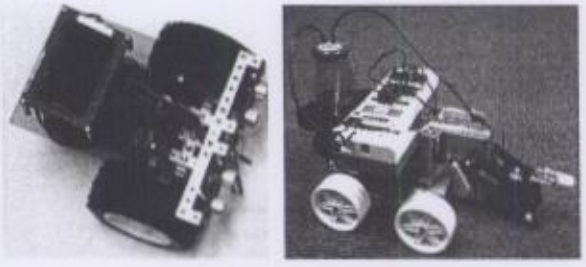

그림 6 이종의 SUGV

표 3 이종 SUGV 하드웨어 정보

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

4.2 타겟 독립 모델(Target Independent Model)

타겟 독립적인 모델은 기능적 요구사항을 시스템에 적용하는 첫 번째 단계이다. 타겟 독립적 모델은 어떠한 하드웨어나 소프트웨어에 의존적이지 않도록 모델링 한다. 이 과정에서 생성되는 모델은 클래스 다이어그램, 병렬 메시지 다이어그램, 병렬 상태 다이어그램이다.

4.2.1 클래스 다이어그램

SUGV 시스템의 정적구조를 모델링하면 그림 7과 같이 구성할 수 있다.

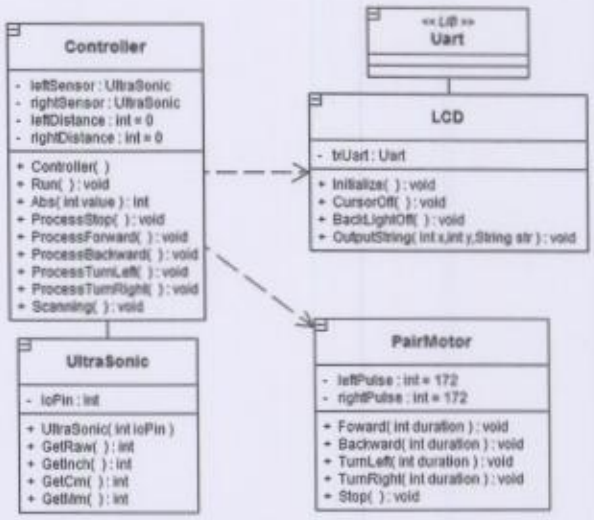

그림 7 TIM의 클래스 다이어그램

사용된 아키텍처는 MVC가 사용되었다. "Controller" 클래스는 센서와, 모터, 그리고 화면장치를 제어하기 위한 컨트롤러이다. "LCD" 클래스는 화면을 제어하기 위한 것이다. "UltraSonic" 클래스는 초음파 센서의 입출력을 담당한다. "PairMotro" 클래스는 모터의 한 쌍을 나타낸다. 모터 클래스를 하나로 만들 수 있지만 모터가 배치 될 때에 같은 방향이 아닌 역방향으로 되어 두 개의 모터를 하나의 클래스로 구성하였다.

4.2.2 병렬 메시지 다이어그램

병렬 메시지 다이어그램은 생성된 객체들 사이의 관계를 나타내게 된다. 그림 8은 외부 물체로부터 데이타를 받은 초음파센서가 컨트롤러에게 전달되어 모터와 LCD를 작동시키는 과정을 모델링한 것이다. 두 개의 초음파센서의 객체의 데이타 값이 컨트롤러에 동시에 전달되기 위해서 AND가 사용되었다.

4.2.3 병렬 상태 다이어그램

병렬 상태 다이어그램은 프로그램이 수행될 때 한 객체에 대한 동적인 상태를 표현하게 된다.

그림 9는 컨트롤 객체에 대한 상태 다이어그램 이다. SUGV의 센서가 고정되어 있다. 그렇기 때문에 앞으로 진행하다가 물체를 만나면 차체를 회전하여 주변 상황

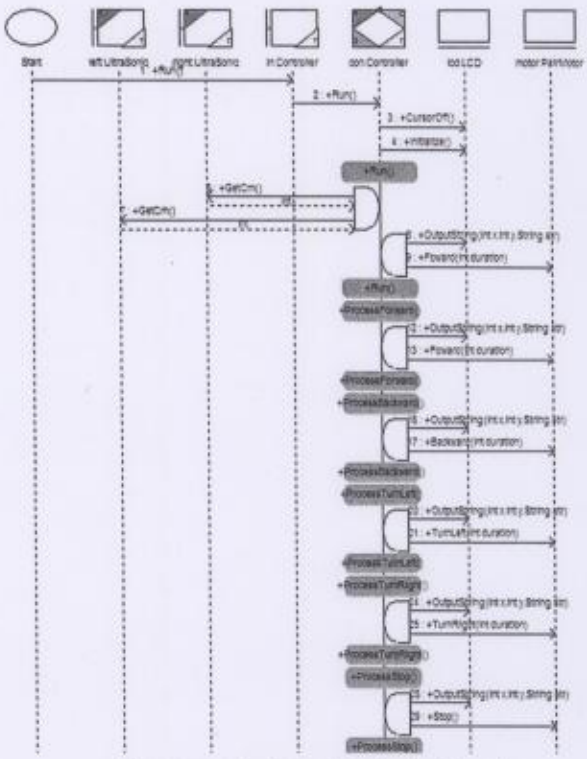

그림 8 TIM의 병렬 메시지 다이어그램

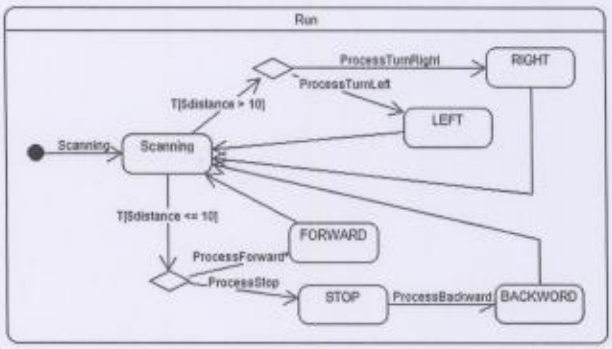

그림 9 TIM의 병렬 상태 다이어그램

을 스캐닝 하여 장애물 지역을 피해가야 한다. 다이어그램을 살펴보면 첫 번째 단계가 스캐닝 단계이고 이 결과를 통해서 수행을 결정하게 된다.

4.3 자동화 도구(HiMEM v1.0)를 사용한 T1 단계 변환

T1단계에서는 타겟 독립 모델을 타겟 종속 모델로 변환한다. 도구를 통하여 타겟 독립 모델을 생성한 다음 타겟 종속 모델로 변환을 시도한다. 이때 그림 10, 그림 11과 같이 프로파일을 선택하여 변환한다.

SUGV1을 위해서는 그림 10과 같이 미들웨어는 "none", RTOS는 "Javeline", 프로세서는 "Ubicom SX48AC"를 선택한 다음 Generate 버튼을 눌러 변환 한다.

SUGV2는 그림 11처럼 미들웨어는 "none", RTOS는 "brickOS", 프로세서는 "Hitachi H8/3292"를 선택 후 'Generate' 버튼을 눌러 변환 한다.

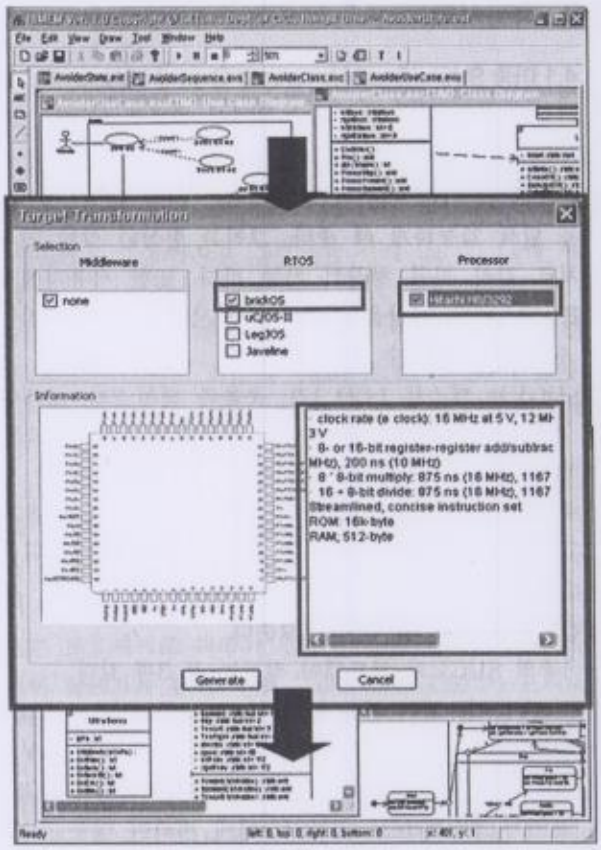

그림 10 SUGV1의 T1 변환

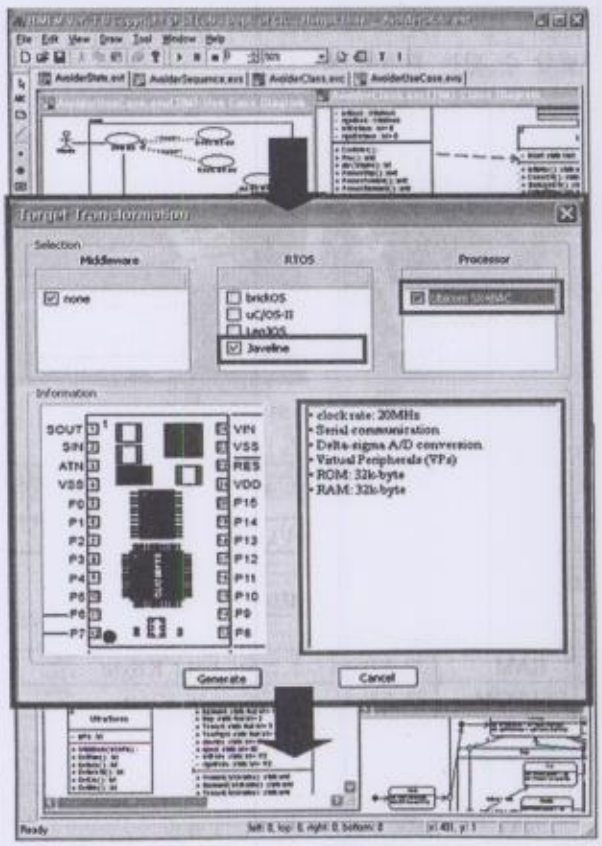

그림 11 SUGV2의 T1 변환

4.4 타겟 종속 모델(Target Specific Model)

타겟 종속 모델은 타겟 독립적인 모델을 미들웨어, 운영체제, 프로세서를 고려하여 생성된 모델이다. 태그를 이용하여 변환하기 때문에 다이어그램의 변화는 없다. 하지만 내부 속성들이 달라졌기 때문에 같다고는 할 수 없다. 이 단계는 자동화도구로는 불가능한 부분을 수작업을 통해서 완성한다. 독립적으로 구성할 수 없었던 것들을 추가로 모델링 하게 된다.

4.4.1 SUGV1

타겟 독립 모델을 자동화 도구를 통해 MindStorm 형태로 변환한다. 변환된 모델은 brickOS의 속성으로 변환된다. 타겟 독립 모델의 구조 그대로 타겟 종속 모델로 변환된다. 이때 프로파일 정보들은 도구 내부에 숨겨져 있으므로 외관상으로는 확인하기 어렵다. 실제적인 구분은 타겟 의존 코드에서 확인이 가능하다.

표 4는 타겟 독립 모델에서 사용한 태그들이 SUGV1에 맞게 변환되는 것을 표로 정리한 것이다. 태그의 내용은 모터를 컨트롤하는 것과 센서의 값을 불러오는 것으로 되어 있다. SUGV1 시스템은 내부 라이브러리가 없고 모터를 직접 펄스를 입력시켜서 모터를 제어하는 것을 확인할 수 있다.

표 4 프로파일에 변환되는 SUGV1

태그	변환
MOTOR_FORWARD	pluseOut(127-speed)
MOTOR_BACKWARD	pluseOut(127+speed)
MOTRO_STOP	pluseOut(127)
SENSOR_TOUCH_GET	getCm()

4.4.2 SUGV2

EMPOS-II시스템에 맞도록 타겟 독립 모델을 변형한다. 타겟 독립 모델에서 타겟 종속 모델로 변환될 때 자동화 도구로 불가능 한 부분은 사용자가 추가한다. 프로그램이 시작되는 메인 부분에 OS_STK TaskStk [N_TASKS][STK_SIZE], OS_STK TaskStartStk[STK_SIZE]를 추가 한다.

표 5는 타겟 독립 모델에서 사용한 태그들이 SUGV2에 맞게 변환되는 것을 표로 정리한 것이다. 태그의 내용은 모터를 컨트롤하는 것과 센서의 값을 불러오는 것으로 되어 있다. SUGV2는 내부 라이브러리를 가지고 있어서 모터를 직접 제어가 아니라 함수를 이용해서 쉽게 제어하는 것을 확인 할 수 있다. SUGV1과 다른 형태로 구성되는 것을 확인할 수 있다.

4.4.3 타겟 종속 모델 검증

도구에서 생성한 모델에 대한 검증을 자동으로 하여 사용자의 실수로 인한 오류를 줄여준다. 그림 12는 병렬

표 5 프로파일에 변환되는 SUGV2

태그	변환
MOTOR_FORWARD	forward(speed)
MOTOR_BACKWARD	revers(speed)
MOTRO_STOP	brake()
SENSOR_TOUCH_GET	get()

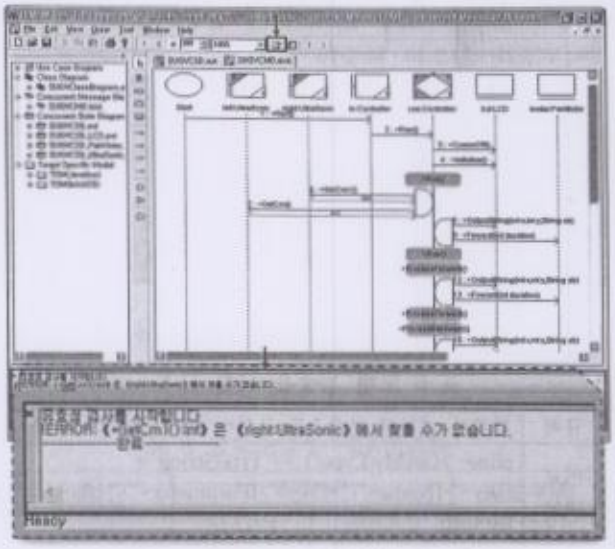

그림 12 병렬 순차 다이어그램의 검증

순차 다이어그램을 모델링할 때 모델 검증 버튼을 누른 화면이다.

위쪽에 빨간 상자에 있는 툴바의 버튼을 누르면 미리 정의된 규칙에 의해서 각각의 모델을 자동으로 검사하여 아래쪽은 빨간 상자와 같이 오류 메시지를 출력하여 준다. 설계단계에서 모델 검증을 통하여 코드를 보다 안정하고 정확하게 생성할 수 있다.

또한 도구는 병렬 상태 다이어그램의 각 상태를 수행하여 논리적인 오류가 발생할 수 있는 부분을 체크한다. 그림 13의 위의 빨간 작은 상자는 상태 다이어그램을 테스트를 수행 하는 버튼 이고 중앙의 부분은 수행되는 모습을 눈으로 확인할 수 있다. 병렬 순차 다이어그램과 같이 다이어그램들의 요소를 검사하여 동적 패스를 출력 창에 보여준다.

다이어그램의 검증은 메타모델을 사용하여 모델규칙을 적용하고 이것을 수행하는 방법으로 사용된다. 사용된 규칙의 내용은 표 6과 같다.

모델 기반의 검증을 통해서 사용자로 하여금 일정수준의 설계를 가능하게 하여 코드생성의 품질을 높일 수가 있다.

4.5 자동화 도구를 사용한 T2 단계로의 변환

그림 14는 변환된 타겟 종속 모델을 나타낸 것이다. 빨간 부분은 타겟 독립 모델, 파란 부분은 타겟 종속 모

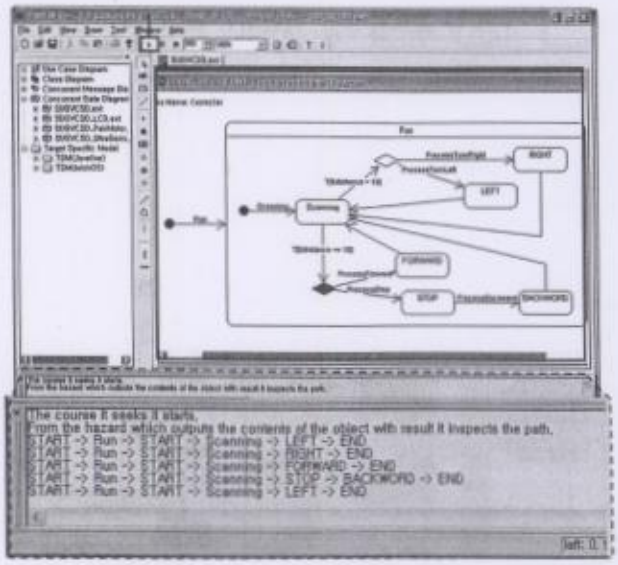

그림 13 병렬 상태 다이어그램의 수행 및 점검

표 6 모델 검증을 위한 규칙

규칙	객체와 연결된 클래스를 조하여 메서드를 검사한다.
비교 대상	pline->GetMyType() == typeString + fkey->fName+"("+fkey->fParameter+"):"+fkey->fReturn;
에러 출력	!ERROR: 《"+pline->GetMyType()+"》 은 《"+pClass->GetMyName()+"》 에서 찾을 수가 없습니다.
규칙	모든 객체와 라인을 조사하여 잘못 연결된 곳을 찾아낸다.
비교 대상	prect == pline->GetEndObject()
에러 출력	!ERROR: 《"+pline->GetStartObject() ->GetMyName()+"("+ pline->GetMyType() +")》 은 《"+prect->GetMyName()+"》 에 잘못 연결되었습니다.

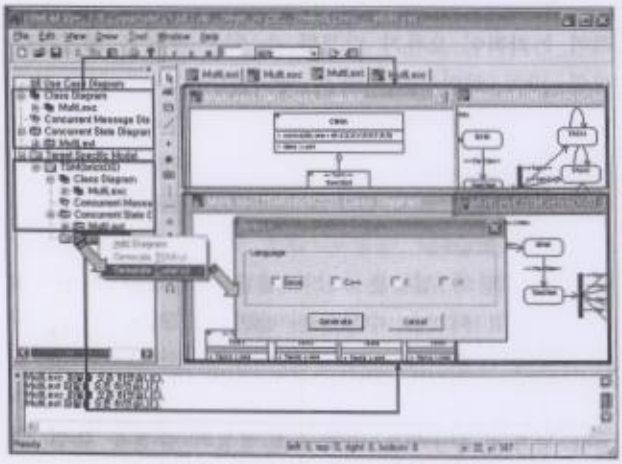

그림 14 TSM의 코드생성 단계

델이다. 타겟 종속 모델을 완성하면 오른쪽 트리메뉴에 있는 "Generate Code(s)"를 이용하여 C, C++, Java 중

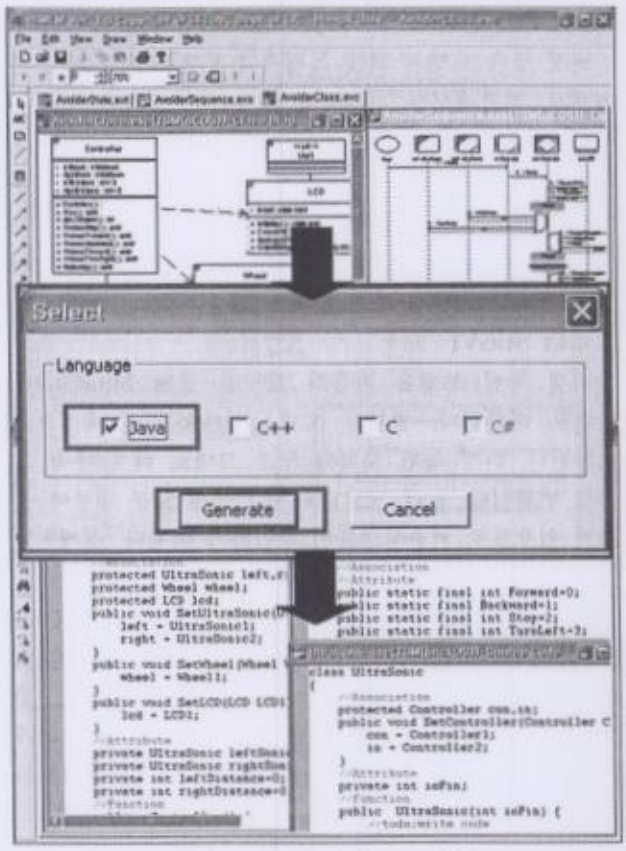

그림 15 SUGV1의 T2 변환

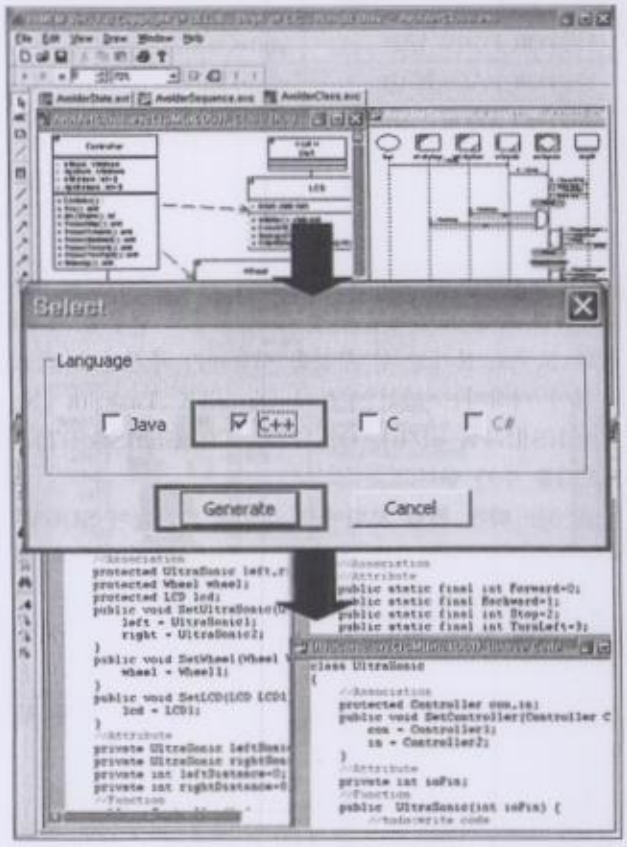

그림 16 SUGV2의 T2 변환

원하는 코드를 생성할 수 있다. 다음은 생성된 타겟 종속 모델을 살펴 볼 것이다.

T2단계에서는 타겟 종속 모델을 타겟 의존 코드로 변환한다. 타겟 종속적인 모델을 설계한 후 각 시스템에 맞는 코드 템플릿을 통하여 그림 15, 그림 16과 같이 코드로 변환 한다. 모든 과정은 자동화 도구를 이용하여 변환된다. SUGV1은 Java를 사용하기 때문에 Java를 선택하고 Generate버튼을 클릭하여 Java로 코드를 생성시킨다.

SUGV2는 C++을 사용하기 때문에 그림 16과 같이 C++을 선택한 다음 Generate버튼을 클릭하여 코드를 생성한다.

4.6 타겟 종속 코드 생성

RCX는 커널과 Application 코드가 분리된 방식을 사용한다. 그래서 먼저 커널을 RCX에 포팅하여 brickOS가 들어가 수행되고 있어야 한다. Application 프로그램을 처리하기 위해서 brickOS내부에 idle 태스크를 포함하고 있다. 표 7은 생성된 코드를 비교한 것이다. 생성된 코드의 컴파일은 gcc 크로스 컴파일러를 사용하기 때문에 make 유틸리티를 사용하여 컴파일 한다.

uC/OS-II는 brickOS와 다르게 kernel과 응용 코드가 같이 포함되어 있는 이미지로 실행된다. brickOS와 마찬가지로 gcc 크로스 컴파일러를 사용하기 때문에 make 유틸리티로 컴파일 할 수 있다.

표 7은 Motor 클래스의 Forward(), Backward(), Stop() 함수이다. 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_ch Position 변수를 이용하여 각자의 위치 상태를 저장하도록 하였다. 도구를 통하여 생성된 3개의 함수를 비교하여 서로 다른 부분은 밑줄을 이용하여 표시 하였다. 코드를 통해 이종의 시스템에 맞도록 변환된 것을 확인할 수 있다.

표 7 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';
	}

4.7 상호운영성 평가

SUGV1과 SUGV2에 표 7의 코드를 컴파일하고 바이너리 코드를 탑재하여 전진, 후진, 멈춤, 좌회전, 우회전을 수행해 보았다. 그 결과 이종의 시스템 SUGV1과 SUGV2는 동일한 동작을 보여주었다. 하나의 타겟 독립 모델을 메타 프로파일을 이용하여 서로 다른 시스템의 동작을 수행할 수 있어 상호운용성을 충족시켜 줬다. 표 8은 두 시스템간의 상호운용성을 평가하기위한 체크리스이다. 평가항목을 기준으로 동일한 수행을 했을 때는 최대 5점으로 표기하고 정도에 따라 차등을 두어 점수로 표기하였다. 전체 시스템을 비교한 결과 30점 만점에 25점으로 83%의 정확도를 확인하였다.

표 8 상호운영성 체크리스트

평가항목	동작여부		동일수행 (0~5점)
	SUGV1	SUGV2	
전진기능	O	O	5
좌회전기능	O	O	3
후회전기능	O	O	3
후진기능	O	O	5
멈춤기능	O	O	5
센싱기능	O	O	4
합계			25

5. 분석

5.1 개발 기간 비교

소프트웨어의 개발은 개발자의 능력에 따라 수행시간이 다르게 작용되어 실험하기 매우 곤란하다. 그래서 비교 대상을 자동화도구가 전체 코드 중 얼마나 많은 양을 생성해줄 수 있는지를 확인하면 그 코드만큼의 개발시간이 줄어들기 때문에 이점을 활용하였다. 코드생성량을 이용하여 개발시간은 COCOMO 모델[13]을 사용하여 계산하였다.

표 9는 자동 생성된 코드를 비교한 것이다. brickOS와 uC/OS-II의 생성된 코드 비율을 살펴보면 각각 86%,

표 9 자동 생성된 코드수 비교

구분	SUGV1	SUGV2
자동 생성 코드	605 LOC	745 LOC
수동 입력 코드	97 LOC	110 LOC
전체 코드	702 LOC	855 LOC
자동 생성 코드비율	86%	87%

표 10 COCOMO 모델 상수표

프로젝트 유형	a	b	c	d
유기적 모드	2.4	1.05	2.5	0.38
반결합 모드	3.0	1.12	2.5	0.35
내장 모드	3.6	1.20	2.5	0.32

표 11 COCOMO 모델을 이용한 개발기간

구분		SUGV1	SUGV2
전체	E	1.6553	2.0360
	D	3.0277	3.2755
자동생성	E	1.4160	1.7619
	D	2.8533	3.1004

87%의 높은 비율로 생성된 것을 확인할 수 있다. 이 자료를 이용하여 개발 기간을 산정하기 위해서 COCOMO 모델을 사용하였다. COCOMO 모델의 수식은 식 (1)과 (2)를 이용하여 간단하게 계산하여 개발 기간을 산출할 수 있는 모델이다.

$$E = a * (KLOC) \exp b \ (E : 노력) \quad (1)$$

$$D = c * (E) \exp d \ (D: 개발기간) \quad (2)$$

식을 대입하기 위해서는 경험적인 상수 값을 사용하는데 상수 값은 표 10에 정리하였다. 상수 값들은 프로젝트의 유형에 따라 유기적, 반결합, 내장 모드로 구분된다. 유기적 모드(Organic mode)는 비교적 엄격하지 않은 요구사항을 기반으로 상대적으로 단순한 소프트웨어에 적용 한다. 반 결합 모드(Semidetached mode)는 크기와 복잡성 면에서 중간 정도의 소프트웨어 프로젝트에 적용한다. 내장 모드 프로젝트(Embedded mode)는 엄격한 제약조건 내에서 개발되어야 하는 소프트웨어 프로젝트에 적용 한다. 본 예제의 규모는 매우 간단한 시스템이므로 유기적 모드의 상수를 사용하여 계산하였다.

식 (1)과 (2)를 적용하여 계산하면 표 11과 같은 결과를 얻을 수 있다.

brickOS는 전체 3.0277(Month)중에 2.8533(Month)의 개발기간을 절약할 수 있다.

uC/OS-II는 전체 3.2755(Month)중에 3.1004(Month)의 개발기간을 절약할 수 있다.

5.2 품질 비교

생성된 코드의 신뢰성 체크를 위해 품질 메트릭[14]을 사용하여 수치화 하였다. 점검한 품질은 예측된 결함 빈도, 결함 발생률이고 계산 공식은 아래의 식 (3),(4)와 같다.

결함예측은 품질 메트릭에 표기된 수식 중 예측된 결함 빈도와 결함 발생률을 그대로 이용하였다. 예측된 오류는 설계 단계의 오류를 예상하여 기술하고 자동화도구를 통해 나온 최종 코드로 검증 하였다. SUGV1과

표 12 생성된 코드의 신뢰성

구분	SUGV1	SUGV2
예측된 오류	2	2
발견된 오류	3	5
제품크기	702 LOC	855 LOC
예측된 결함 빈도	0.00142	0.00351
발견된 결함 수	5	8
수행된 테스트 수	38	45
결함 발생율 (백분율)	0.13157 13%	0.17778 17%

SUGV2에서 예측 가능한 오류는 센서를 2개를 사용하기 때문에 사각지역에 들어갔을 경우 빠져나가기 어렵다는 것과 갑자기 물체가 튀어 나왔을 때 회피할 수 없을 것이라고 예상하였다. 이렇게 예상하고 실제 코드를 가지고 수행해보았더니 SUGV1은 3개의 오류, SUGV2는 5개 오류가 나왔고 이것을 물체에 위치에 따라 SUGV가 빠져나올 수 있는 것을 가지고 수차례 테스트 하였다.

- 예측된 결함 빈도

$$X = ABS(예측된\ 오류 - 발견된\ 오류)/제품\ 크기 \quad (3)$$

- 결함 발생율

$$X = 발견된\ 결함의\ 수/수행된\ 테스트\ 수 \quad (4)$$

실험을 통해서 나온 데이타를 식 (3)과 (4)를 적용하여 계산하면 표 12와 같은 결과를 얻을 수 있다.

테스트 데이타는 분기 적합성 기준을 사용하여 생성하였고 코드에 데이타 값을 직접 삽입하여 수행 하였다.

brinkOS와 u/COS-II의 결함 발생률이 각각 13%, 17%로 비교적 낮은 수치를 나타냈다. 이것은 생성된 코드의 오류 보다는 설계에서 고려하지 못한 요소가 문제가 되는 경우가 대부분 이였다. 설계의 검증을 더 보강한다면 결함 발생율은 크게 떨어질 것이다.

6. 결 론

SUGV의 활용이 늘어감에 따라 구성하는 소프트웨어가 복잡해지고 개발 환경 다양화로 인한 상호운용성 문제가 대두되고 있다. 본 논문에서는 이러한 문제점을 해결하고자 기존의 UML 프로파일 메커니즘을 이용하여 SUGV 개발에 MDA가 적용되도록 하였다. 이를 통해 이종 SUGV 소프트웨어 설계시 타겟 독립 모델을 만든 후 UML 프로파일이 적용된 자동화 도구를 이용해 타겟 종속 모델 및 코드를 생성하였고 또한 이종 기기의 개발기간과 노력을 단축시킬 수 있었다. 그리고 자동 발생된 코드의 적용을 통해 SUGV 소프트웨어가 기존의 성능을 유지하면서도 개발기간이 단축됨을 확인할 수 있었다.

향후 연구로는 개발한 도구를 통해 설계한 모델의 자

동 문서화와 모델 시뮬레이션 기능이 필요하다. 그리고 미들웨어의 지원뿐만 아니라 더욱 많은 운영체제와 프로세서가 지원될 수 있도록 프로파일에 관한 연구를 진행 중이다.

참 고 문 헌

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

[2] 서장원, "국방사업의 소프트웨어 개발 프로세스와 소프트웨어 정부품질보증", 항공우주산업기술동향, Vol.4, No.2, 76-85, 2006.

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

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

[5] OMG, http://www.omg.org/technology/documents/modeling_spec_catalog.htm

[6] 김우열, 김영철, "확장된 xUML을 사용한 MDA기반 이종 임베디드 소프트웨어 컴포넌트 모델링에 관한 연구", 정보처리학회논문지, Vol.14-D, No.1, 2007. 2.

[7] Renesas, 'H8/300 Series', http://kr.renesas.com/

[8] brickOS, http://brickos.sourceforge.net/

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

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

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

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

[13] Barry W. Boehm, 'Software Engineering Economics', Prentice-Hall, 1981.

[14] ISO/IEC 9126, 'Software engineering : Product quality', http://www.iso.org

김 우 열

2004년 홍익대학교 컴퓨터정보통신공학과(학사). 2006년 홍익대학교 일반대학원 소프트웨어공학전공(석사). 2006년~현재 홍익대학교 일반대학원 소프트웨어공학전공 박사과정. 관심분야는 상호운용성, 임베디드 소프트웨어 개발 방법론 및 도구 개발, 컴포넌트 시험 및 평가, 리팩토링

손 현 승

2007년 홍익대학교 컴퓨터정보통신공학과(학사). 2007년~현재 홍익대학교 일반대학원 소프트웨어공학전공(석사). 관심분야는 임베디드 소프트웨어 자동화 도구 개발, 임베디드 RTOS 개발, 임베디드 MDA (Model Driven Architecture) 연구, 모델 검증 기법 연구

김 영 철

2000년 Illinois Institute of Technology (공학박사). 2000년~2001년 LG 산전 중앙연구소 Embedded system 부장. 2001년~현재 홍익대학교 컴퓨터정보통신공학과 부교수. 관심분야는 테스트 성숙도 모델(TMM), 임베디드 S/W 개발 방법론 및 도구 개발, 모델 기반 테스팅, CBD, BPM, 사용자 행위 분석 방법론