모델 기반 접근 방법을 이용한 임베디드 S/W를 위한 API 변환기의 개발

Development of a Model-Driven Approach Based API Translator for Embedded Software

박병률*, 맹지찬**, 이종범*, 유민수**, 안현식*, 정구민*
Byeong-Ryul Park*, Ji Chan Maeng**, Jong-Bum Lee*, Minsoo Ryu**, Hyun-Sik Ahn*, Gu-Min Jeong*

요 약

본 논문에서는 모델 기반 접근 방법을 이용하여 임베디드 소프트웨어를 개발하기 위한 자동화된 API 변환기에 대하여 기술한다. MDA(Model Driven Architecture)가 임베디드 소프트웨어의 개발에 관해서는 지원이 매우 미약하기 때문에, 우리는 MDA의 장점인 구현 공정의 자동화를 포함하면서도 실시간 오버헤드 문제를 해결할 수 있는 새로운 접근 방법을 제시한다. 첫 번째로, 어느 특정 RTOS에 의존하지 않지만 전형적인 RTOS 서비스들의 대부분을 제공할 수 있도록 Generic API를 정의한다. 우리는 이 Generic API를 이용하여 타겟 응용프로그램의 RTOS와 관련된 행동을 CIC(Common Intermediate Code)에서 표현할 수 있다. 그 다음에, Generic API를 이용하여 기술된 CIC를 특정 RTOS에서 수행할 수 있는 C-코드로 변환할 수 있는 변환 툴을 제안한다. 제안된 API 변환기는 외부에 정의되어 있는 XML 변환 규칙을 이용하여 코드를 변환한다. 따라서 API 변환기는 이 변환 규칙을 수정하거나 추가하여 다른 RTOS로 적용할 수 있도록 확장이 가능하다. 실험을 통하여 제안된 방법을 확인한다.

Abstract

In this paper, we present an automated API translator for embedded software development based on a model-driven approach. Since MDA(Model Driven Architecture) provides little support for the development of embedded software, we propose a new method which contains the MDA's advantage, automation of implement process, and can solve the problem of real-time overhead. First, we define 'generic APIs' which do not depend on any RTOS's but provide most of typical RTOS services. We can describe RTOS-related behaviors of target application using these generic APIs in a CIC(Common Intermediate Code). Then, we propose a transformation tool for translating a CIC using generic APIs into a C-code for specific RTOS. The proposed API translator converts them into C-code using XML transformation rule which is defined outside. It indicates that an API translator extends to other RTOS's by modifying or adding the transformation rule. From the experiment, we validate the proposed method.

Keywords :API Translator, MDA(Model Driven Architecture), RTOS, Embedded Software Development, POSIX.

I. 서 론

하드웨어 개발이 진보됨에 따라 소프트웨어 개발 또한 더 복잡하고 거대한 시스템으로 바뀌게 되었고, 이로 인해 모델 기반 접근(Model-Driven Approach) 방법을 이용한 소프트웨어 개발 방법이 등장하였다. 그리하여 개발자는 특정 하드웨어와 소프트웨어에 종속되지 않는 추상 모델을 만드는데 더 중점을 두게 되었고, 이 모델들은 자동화된 툴에 의해 실행 가능한 코드로 변환된다 [1-2]. 더불어 이러한 모델 기반 접근 방법은 생산성, 호환성 및 유지보수 측면에서 많은 이점들을 제공한다.

MDA(Model Driven Architecture)는 소프트웨어의 설계를 돕고 그 효율성을 향상시키기 위해 객체 관리 그룹(Object Management Group)에 의해 제안되었다 [3-7]. MDA의 중요한 특징은 어플리케이션의 구체적인 구현 사항으로부터 추상적인 행동과 기능을 분리하는 것이다. 이를 위해 MDA는 다음의 3가지 중요한 모델을 제공한다.

- **PIM(Platform Independent Model)**: PIM은 현재 구현된 어떠한 기술에도 종속되지 않는 모델이다.
- **PSM(Platform Specific Model)**: PSM은 현재 구현된 어느 특정 기술을 포함하는 모델이다.
- **Code**: PIM으로부터 변환된 PSM들은 마지막으로 코드로 변환된다.

* 국민대학교 전자공학부
** 한양대학교 전자통신컴퓨터공학부
논문 번호 : 2007-3-6 접수 일자 : 2007. 6. 14
심사 완료 : 2007. 7. 26

MDA의 개발 과정은 그림 1에서 보는 바와 같다. PIM은 하나 또는 그 이상의 PSM으로 변환되고 마지막으로 수행 가능한 코드로 변환된다. 이러한 변환 과정은 변환 툴에 의해 자동으로 수행된다. 그러나, 현재의 MDA는 주요 대상이 EJB, Microsoft .NET 및 CORBA 등 타겟 플랫폼 위의 미들웨어이기 때문에 임베디드 소프트웨어 개발에 대한 지원이 미약하다. 미들웨어에서는 가상머신이 운영체제와 컴퓨터 플랫폼 사이에서 가상적인 환경을 제공해 주므로, 응용 프로그램은 이러한 가상머신 위에서 수행된다. 하지만, 대부분의 임베디드 시스템은 실시간성과 제한된 자원 등의 요구조건을 필요로 한다. 이러한 이유로 임베디드 환경에서는 실시간 오버헤드를 고려해야만 하는 미들웨어 플랫폼 대신에 RTOS(Real-Time Operating System)가 요구된다.

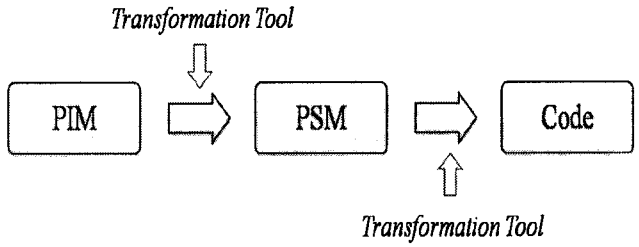

그림 166. MDA 개발 과정.
Fig. 1. MDA development process.

현재 산업 현장에서는 모델 기반 소프트웨어 개발을 위해 Rose [8], Rhapsody [9] 및 Tau [10] 등과 같은 다양한 CASE(Computer-Aided Software Engineering) 툴들이 사용되고 있다. 개발자는 이 툴들을 이용해 응용 프로그램의 모델을 만들고 변환 툴을 이용해 최종 코드를 생성한다. 하지만 이러한 툴들 역시 플랫폼과의 독립성을 이루기 위해 가상머신을 이용하기 때문에 임베디드 소프트웨어를 개발하는데 적합하지 않다..

본 논문에서는 임베디드 소프트웨어 개발을 위해 모델 기반 접근 방법을 제시하고 자동화된 변환 툴에 대하여 기술한다. 본 연구는 HOPES (Hopes of Parallel Embedded Software) 프로젝트 결과물의 일부분이다 [11-13]. HOPES는 모델 기반 임베디드 소프트웨어 개발을 위한 통합 환경 툴을 개발하는 프로젝트이다. 제안된 방법은 RTOS 플랫폼을 고려한다는 것을 제외한다면 MDA와 매우 유사하다. 모델링 단계에서 응용 프로그램의 RTOS와 관련된 행동들을 정의하고, RTOS에서 수행 가능한 최종 코드를 생성해낸다. 먼저 전형적인 RTOS 서비스들을 표현하고 추상화시킬 수 있는 Generic API (Application Programming Interface)들을 정의한다. 이를 위해 우리는 호환성이 우수한 POSIX (Portable Operating System Interface for UNIX) 표준을 이용한다 [14-15]. 또한 Generic API를 사용하여 작성된 중간 형태의 코드를 특정 RTOS에 종속되는 C-코드로 변환할 수 있는 API 변환기를 개발하였다. 외부에 정의된 변환 규칙을 수정함으로써 다른 RTOS로도 확장이 가능하다. 또한 API 변환이 중심이 되기 때문에 변환 시 실시간 오버헤드를 줄일 수 있다.

기본 개념은 이미 이전 연구에서 제안되었다 [16]. API 변환기가 사용하는 변환 규칙이 [16]에서는 패턴, 심벌 및 룰로 나누어져 있었다. 이로 인해 서로 중복되는 사항이 존재하고 관리도 어려워 본 논문에서는 XML을 이용해 이들을 통합하였다. 이로써 변환규칙은 기존의 코드에서 모델로의 역변환을 수행할 수 있도록 확장이 가능하게 되었다.

본 논문에서는 다음과 같이 구성된다. 2장에서는 Generic API를 정의하고 어떻게 RTOS와 관련된 행동들을 표현할 수 있는지 기술한다. 3장에서는 API 변환기의 설계 및 구현에 관하여 기술하고 4장에서 API 변환기의 변환 실험 결과를 기술한다. 마지막으로 5장에서 결론을 맺고 향후 연구되어야 할 과제를 제시한다.

II. Generic API를 이용한 모델 기반 소프트웨어 개발

본 연구는 HOPES 프로젝트의 결과물 중 일부이다 [11]. HOPES 프로젝트의 목표는 병렬 임베디드 소프트웨어를 개발하기 위한 환경을 제공하는 소프트웨어를 개발하는 것이다. 이 프로젝트에서 우리는 멀티프로세서용 OS API 변환 툴을 개발하는 것이 목표였다.

2.1 Generic API의 정의

전형적인 RTOS들은 개발자가 실시간 임베디드 시스템에서 응용프로그램을 개발할 수 있도록 서비스들을 제공한다. 이러한 서비스들은 태스크 관리, 메모리 관리, 파일 시스템 관리 및 I/O 관리 등을 포함하는 API 형태로 이루어져 있다. 우리는 이러한 API들을 어떠한 플랫폼에도 종속적이지 않도록 추상화 시키고, 이를 'Generic API'로 정의한다. 이를 위해 POSIX 표준 1003.1-2004 [14]이 사용 되었다. 이 표준에서는 응용프로그램이 운영체제로부터 기본적인 서비스들을 획득할 수 있는 방법이 정의된다.

Generic API를 정의하기 위하여, 먼저 서로 관련 있는 API들끼리 그룹화 하였다. 그리고 서로 비슷한 API들, 또는 RTOS와 직접적으로 관련이 없는 API들을 제거하였다. POSIX 표준 API에는 비슷한 기능을 가진 API들이 많이 존재한다. 예를 들어, printf(), fprintf()와 sprintf()는 출력 대상이 되는 타겟이 다를 뿐 모두 출력을 담당하고 있는 API들이다. 우리는 Generic API들을 RTOS가 제공하는 서비스 그룹을 중심으로 분류하였고, 이에 네트워크 분야와 실시간 분야를 추가하였다. 표 1은 Generic API들과 그 분류를 보여주고 있다.

표 1. Generic RTOS API들과 분류.
Table 1. Generic RTOS APIs and those categories.

분 류	Generic RTOS APIs (총 78개)	개수
Task	EXEC, EXIT, FORK, GETPID, KILL, NICE, SIGNAL, SLEEP, STAT, USLEEP, WAITPID, YIELD, THREAD_MUTEX_LOCK, THREAD_MUTEX_UNLOCK, THREAD_COND_SIGNAL, THREAD_COND_WAIT, THREAD_CREATE, THREAD_EXIT, THREAD_JOIN, THREAD_KILL.	21
Real-Time	MQ_GETATTR, MQ_NOTIFY, MQ_RECEIVE, MQ_SEND, MQ_SETATTR, SEM_WAIT, SEM_POST, THREAD_BARR_WAIT, THREAD_SPIN_LOCK, THREAD_SPIN_UNLOCK, SCHEDGETPARAM, SCHEDSETPARAM, SEM_GETVAL, SHM_OPEN, SHM_UNLINK.	15
Network	SOCK_ACCEPT, SOCK_BIND, SOCK_CONNECT, SOCK_GETOPT, SOCK_LISTEN, SOCK_RECV, SOCK_SEND, SOCK_SETOPT, SOCKET.	9
Memory	MALLOC, FREE, MMAP, MUNMAP.	4
File	CHDIR, CLOSE, EOF, FILENO, MKDIR, OPEN, PRINT, READ, REMOVE, RENAME, RMDIR, SCAN, SEEK, SETBUF, TELL, LINK, UNLINK, WRITE.	18
Device	IOCTL, POLL, SELECT.	3
Others	CTIME, GETITIMER, GETTIMEOFDAY, MKTIME, PERROR, RAND, TIME, SETITIMER.	8

2.2 Generic API를 이용한 RTOS와 관련된 행동 기술

그림 2는 Generic API를 이용한 모델 기반 접근 방법을 보여준다. 제안된 방법은 3단계로 구성된다. 첫 단계에서는 응용프로그램의 데이터 흐름 모델과 Generic API들을 이용하여 ABM(Application Behavior Model)을 생성한다 [12]. 다음 단계에서는 코드 합성기를 통하여 Generic API들을 포함하는 중간 형태의 C-코드인 CIC (Common Intermediate Code)로 변환된다. 마지막 단계에서 대상이 되는 RTOS가 정해지고, API 변환기는 타겟 플랫폼에서 실행 가능한 완전한 C 코드로 변환한다.

Generic API를 이용하여 RTOS와 관련된 행동을 구체적으로 어떻게 기술하는지 다음의 예를 이용하여 설명한다. 그림 3은 쓰레드 동기화에 대한 예를 보여주고 있다. 멀티 쓰레드 프로그램에서 공유 데이터를 동기화하기 위해서는 크리티컬 섹션이 사용된다. 우리는 간단히 이 예제를 구현하기 위해 뮤텍스(Mutual Exclusion)를 이용한다. 이 예제에서는 3개의 쓰레드가 사용된다. 응용프로그램의 주 쓰레드는 공유 데이터를 이용하는 2개의 쓰레드를 생성한다. 그중에 하나는 공유 데이터 값을 증가시키고, 나머지 하나는 공유 데이터 값을 감소시킨다. 주 쓰레드는 2개의 Generic API, THREAD_CREATE()와 THREAD_JOIN()을 호출한다. 주 쓰레드에서 THREAD_CREATE()는 새로운 쓰레드를 생성할 수 있고, THREA_JOIN()는 쓰레드가 종료될 때 까지 대기할 수 있다. 비슷하게, 다른 2개의 쓰레드에서도 RTOS와 관련된 행동을 표현하기 위해 Generic API를 호출할 수 있다. MUTEX_LOCK()과 MUTEX_UNLOCK()을 호출함으로

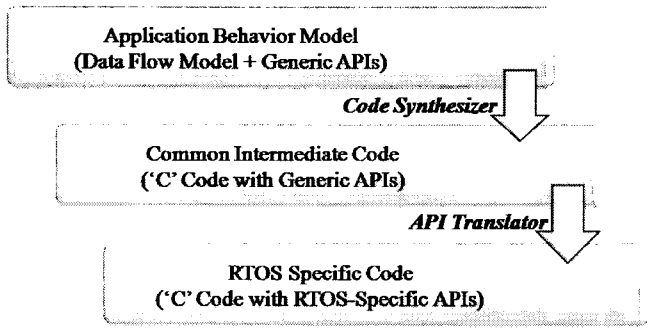

그림 167. 제안된 방법의 개발 과정.
Fig. 2. Development process of the proposed method.

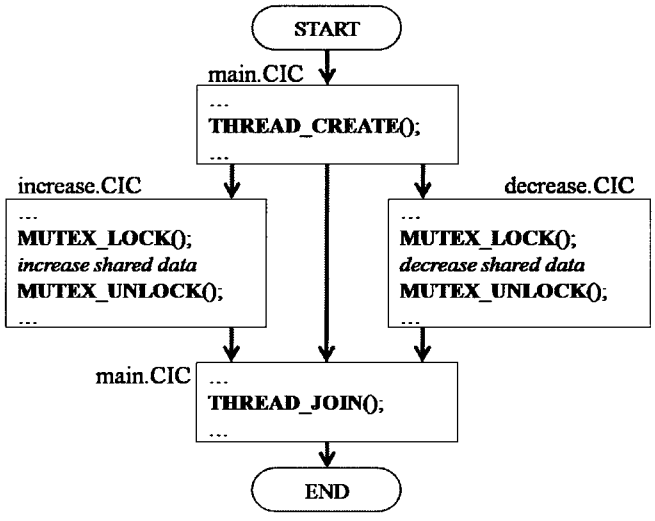

그림 168. Generic API들을 이용하여 RTOS와 관련된 행동을 기술하는 예.
Fig. 3. An example of specifying the RTOS-related behavior using generic APIs.

써 뮤텍스를 잠그거나 잠금을 풀 수 있는 것이다. 이러한 Generic API들은 API 변환기에 의해 타겟이 되는 플랫폼에서 제공하는 API들로 변환된다.

III. API 변환기의 설계 및 구현

Generic API들은 RTOS와 관련된 행동이 구체적으로 명시되고 나서, 타겟 RTOS에서 제공하는 API들로 변환된다. 이 과정은 API 변환기에 의해서 자동으로 수행되며 대상이 되는 응용프로그램의 태스크 코드는 다음과 같은 가정을 필요로 한다.

1. 모든 태스크 코드는 각 태스크 별로 분리되어 작성되어야 한다.
2. 각각의 태스크 코드는 초기화를 담당하는 _init() 함수, 태스크 수행을 담당하는 _go() 함수 그리고 종료화를 담당하는 _wrapup() 함수를 포함해야 한다.
3. 다른 태스크들을 생성하는 주 태스크는 각 태스크들의 _init() 함수, _go() 함수 그리고 _wrapup() 함수들을 순서대로 호출하여야 한다.

본 논문에서 제안하는 API 변환기는 각 태스크 코드들을 API 단위로 변환하기 때문에 때로는 초기화 코드나 종료화 코드를 삽입해야 하는 경우가 생긴다. 만약 이러한 코드들이 어떤 함수 내에서 지역적으로 요구된다면 우리는 함수 내부에 추가할 수 있다. 하지만 이와 반대로 태스크 전체에 전역적으로 사용하도록 요구된다면 우리는 이러한 코드들을 삽입할 수 있는 정형화된 함수가 필요하다. 만약 Generic API가 타겟 API로 변환될 때 초기화 코드가 필요하다면 _init() 함수에 삽입되고, 종료화 코드가 필요하다면 _wrapup() 함수에 삽입된다. 주 태스크에서는 각 태스크들에 정의되어 있는 _init() 함수, _go() 함수 그리고 wrapup() 함수를 차례로 호출함으로써 태스크가 정상적으로 수행될 수 있다.

이번 장에서는 API 변환기가 타겟 코드로의 변환을 구체적으로 어떻게 수행하는지 기술한다. 본 논문에서 제안하는 API 변환기를 개발하는 데 있어 가장 중요하게 요구되는 사항은 효율적인 확장성이다. API 변환기는 POSIX 기반이 아닌 다른 RTOS로 확장을 위해 이미 데이터 표현의 효율성 면에서 검증된 XML을 이용하여 기술된 변환 규칙을 사용한다. 개발자는 이 변환 규칙을 외부에서 손쉽게 수정하거나 추가하여 모델링된 태스크 코드를 원하는 타겟 플랫폼 코드로 변환할 수 있다.

3.1 API의 패턴과 심벌

선행 연구 [11]를 통하여 몇몇의 API들은 패턴을 형성하여 사용된다는 것을 발견하였다. 예를 들어, 메모리 동적 할당을 위하여 malloc() 함수를 사용했다면 사용한 메모리를

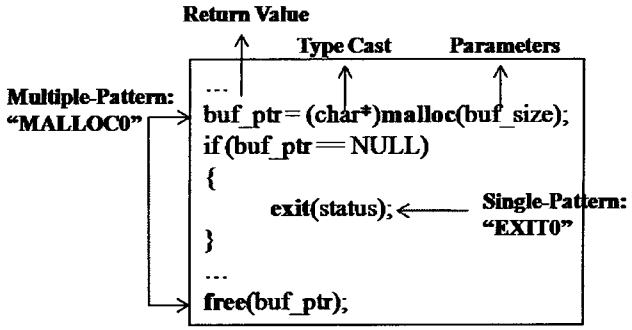

그림 169. 패턴과 API의 부가 정보들.
Fig. 4. Patterns and additional information of APIs.

해제하기 위하여 free() 함수가 사용된다는 것이다. 우리는 이러한 순차적인 함수 호출을 '패턴'이라고 정의한다. 이러한 패턴들은 오직 하나의 API로 구성된 single-패턴과 두 개 이상의 API들로 구성된 multiple-패턴으로 구분되어진다. 그림 4에서 보듯이 MALLOC0 패턴은 multiple-패턴으로서 malloc() 함수와 free() 함수로 구성되고, EXIT() 패턴은 single-패턴으로서 오직 exit() 함수로만 구성되는 것을 알 수 있다.

API는 일반적으로 부가정보들과 함께 호출된다. 부가정보들은 함수의 파라미터, 파라미터의 개수, 반환 값 또는 반환 값의 형 변환 등이 될 수 있다. 우리는 이렇게 API와 함께 제공되는 부가정보들을 '심벌'이라고 정의한다. 그림 4에서 malloc() 함수를 예로 들어보면, 반환 값은 변수 buf_ptr이고 반환 값의 형 변환 타입은 문자 포인터형이다. 파라미터는 buf_size 변수이고 파라미터의 개수는 1개이다. 또한 malloc() 함수와 패턴을 이루는 free() 함수와 buf_ptr 변수를 공통 심벌로서 서로 공유하고 있다.

우리는 이러한 패턴과 부가 정보들을 변환 규칙 파일에서 XML을 이용하여 표현하고 있다. 그림 5는 MALLOC0 패턴을 XML로 기술한 예를 보여주고 있다. 패턴과 심벌 정보들은 변환이 완료될 때까지 심벌 테이블에 등록하여 그 값을 유지한다. 이 정보들은 변환 규칙을 이용하여 변환 구문을 생성할 때 이용되어진다. 그래서 변환 규칙에서 심벌 정보를 표현하기 위하여 @VAR 기호가 사용되고, 인덱스를 표현하기 위하여 C언어에서의 배열과 유사하게 꺾쇠괄호 []가 사용된다. 즉, @VAR[0] 은 심벌 테이블에서 첫 번째 심벌 값을

```
<!--
      MALLOC(buf_ptr, buf_size, type_cast);
      FREE(buf_ptr);
-->
<PATTERN name="MALLOC0">
  <API name="MALLOC" ret="0" cast="0" param="3">
      <!-- transformation rule of MALLOC() -->
  </API>
  <API name="FREE" ret="0" cast="0" param="1">
      <!-- transformation rule of FREE() -->
  </API>
</PATTERN>
```

그림 170. XML을 이용한 MALLOC0 패턴의 표현.
Fig. 5. Description of pattern MALLOC0 using XML.

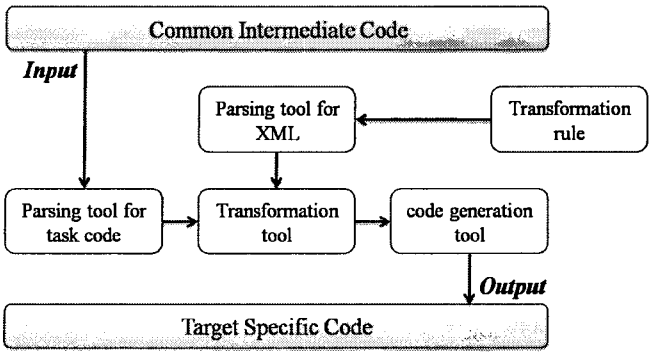

그림 172. API 변환기의 블록 다이어그램.
Fig. 7. Block diagram of API translator.

의미한다. 이렇게 변환 규칙의 심벌들이 심벌 테이블에 있는 값으로 대체되면 완전한 변환 구문이 만들어지게 된다.

3.2 변환 규칙

API 변환기는 모든 Generic API마다 정의된 변환 규칙을 이용하여 변환을 수행한다. 표 2는 변환 규칙을 표현하기 위해 사용되는 7개의 명령어들을 나타내고 있다. 개발자는 이 명령어들을 이용하여 변환 규칙을 구체적으로 표현할 수 있다. 명령어와 함께 사용되는 부가정보들은 각 명령어에 따라 다르다. 예를 들어, INCLUDE 명령어의 경우, 추가되어야 하는 파일이 외부 헤더 파일인지 내부 런타임 라이브러리인지 판단하는 플래그가 필요하지만, DECLARE 명령어의 경우, 선언되어야 하는 변수가 전역변수인지 지역변수인지 판단하는 플래그가 필요하다.

표 2. 변환 규칙 명령어.
Table 2. Directives of transformation rule.

명령어	의 미
INCLUDE	주어진 헤더 파일을 추가한다.
DELCARE	주어진 변수를 선언한다.
INITIALIZE	주어진 초기화 코드를 삽입한다.
REPLACE	주어진 구문으로 대체한다.
INSERT	주어진 구문을 삽입한다.
FINALIZE	주어진 종료화 코드를 삽입한다.
DELETE	주어진 구문을 삭제한다.

이러한 변환 규칙도 XML을 이용하여 기술되며, 그림 6은 패턴과 심벌을 이용하여 MALLOC0 패턴의 변환 규칙을 표현한 예를 보여주고 있다. API 변환기는 심벌들을 심벌 테이블에 등록되어 있는 완전한 값으로 대체하여 새로운 변환 구문을 생성한다. 이렇게 변환 규칙 명령어를 사용하면 헤더 파일 추가, 변수 선언, 특정 코드 삽입 또는 삭제가 가능하다.

```
...
<!--
        MALLOC(buf_ptr, buf_size, type_cast)
                ⇓
        #include <stdlib.h>
        buf_ptr = (type_cast)malloc(buf_size)
-->
<RULE dir="INCLUDE" in="1">stdlib.h</RULE>
<RULE dir="REPLACE">@VAR[0] = (@VAR[2])malloc(@VAR[1])</RULE>
...
<!--
        FREE(buf_ptr)
                ⇓
        #include <stdlib.h>
        free(buf_ptr)
-->
<RULE dir="INCLUDE" in="1">stdlib.h</RULE>
<RULE dir="REPLACE">free(@VAR[0])</RULE>
...
```

그림 171. MALLOC0 패턴의 변환 규칙.
Fig. 6. Transformation rule of pattern MALLOC0.

3.3 API 변환기의 구현

API 변환기는 C언어로 구현되었다. 그림 7에서 보듯이 중간 형태의 코드인 CIC를 입력으로 받아 타겟 코드로 출력을 내보내며 다음의 4가지 툴로 구성된다. 입력 파일 파서, XML로 기술된 변환 규칙 파일 파서, 패턴, 심벌 그리고 룰을 이용한 API 변환 툴, 마지막으로 변환 구문을 이용하여 출력 파일을 생성하는 툴이다.

API 변환기는 다음 4단계의 순서로 동작한다.

1. **초기화**: API 변환기는 필요한 초기화 코드를 수행하고 입력/출력 파일 리스트 획득 및 변환 규칙을 파싱하여 API 변환기가 사용할 수 있도록 자료구조를 정의한다.
2. **관련 정보 수집**: 입력 태스크 코드를 파싱하여 변환 후보가 되는 API를 분류하고 그들의 부가 정보들을 수집하여 변환 준비를 한다.
3. **코드 변환**: 패턴을 참조하여 변환이 되어야 하는 API들을 분류하고 심벌 테이블, 변환 규칙을 참조하여 변환 구문을 생성한다.
4. **출력 파일 생성**: 기존 코드와 앞 단계에서 생성된 변환 구문을 이용하여 최종 출력 코드를 생성하고 메모리 정리 및 종료화 코드를 호출한다.

API 변환기는 여러 개의 태스크 코드를 입력/출력으로 할 수 있으며 외부에서 정의된 파일 리스트를 통하여 관리된다. 개발자는 이 파일 리스트에 입력/출력 파일들을 추가하여 순서대로 변환시킬 수 있다.

IV. 실험 결과

API 변환기의 동작을 확인하기 위하여 간단한 쓰레드 동기화 예제를 이용하여 실험을 수행하였다. 예제 코드는 변환 규칙을 이용하여 서로 다른 두 RTOS로의 코드 변환을 수행

한다. RTOS는 POSIX 표준을 따르는 Linux와 VPOS 2.0을 사용하였다. VPOS 2.0은 HOPES 프로젝트에서 개발된 RTOS이다 [11]. 이 예제에서는 3개의 CIC 파일이 사용되었다. 다른 쓰레드들을 생성하는 쓰레드 코드인 task_main.CIC, 공유 데이터의 값을 증가시키는 쓰레드 코드인 task_increase.CIC, 그리고 공유 데이터의 값을 감소시키는 쓰레드 코드인 task_decrease.CIC이다. API 변환기는 이 CIC 파일들을 Linux용 변환 규칙을 이용하여 Linux에서 실행 가능한 C파일을 만들고 VPOS 2.0용 변환 규칙을 이용하여 VPOS2.0에서 실행 가능한 C파일을 생성한다.

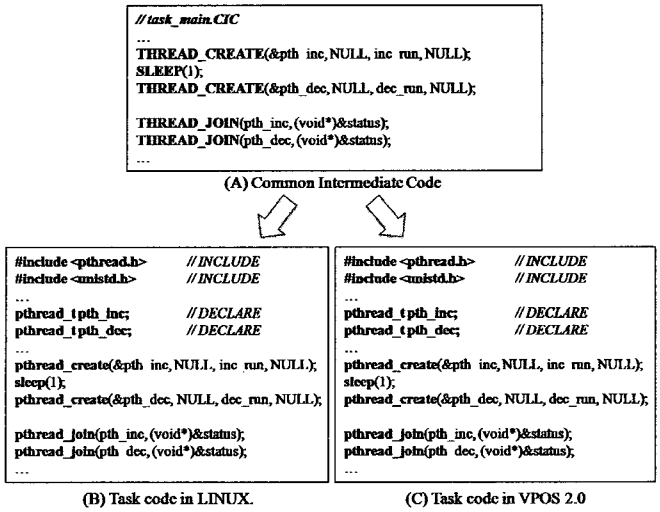

그림 173. API 변환기를 이용한 코드 변환.

Fig. 8. Code translation using an API translator.

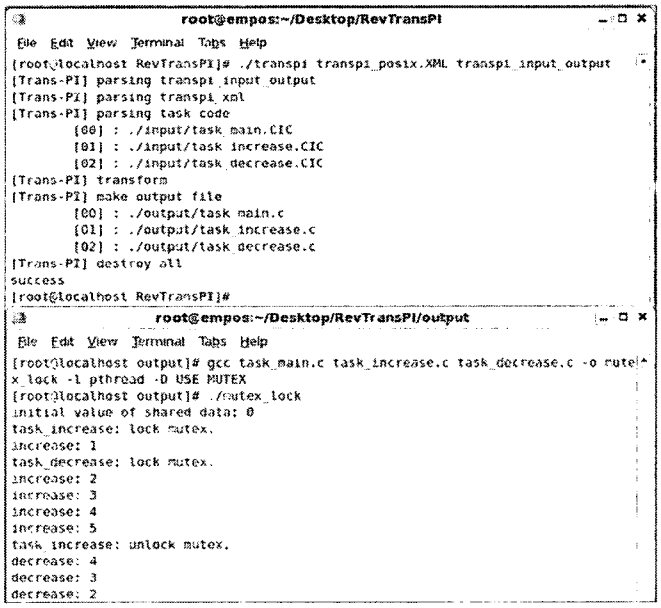

그림 174. 동기화 예제의 스크린샷.

Fig. 9. Snapshot of synchronization example.

변환 결과가 그림 8과 그림 9에 보이고 있다. 필요한 헤더 파일들이 추가 되었고, 쓰레드와 관련된 변수들이 선언되었으며, Generic API들이 Linux와 VPOS 2.0에서 제공하는 API들로 변환되었다. 그림 10에서 보는바와 같이 이러한 변환 과정은 API 변환기에 의해 자동으로 수행된다. 두 RTOS가 모두 POSIX 표준을 따르기 때문에 변환이 수행된 후의 쓰레드 관련 API가 동일하다. 그러나 우리는 API 변환기가 각 RTOS의 변환 규칙을 이용하여 Generic API들을 타겟 RTOS에서 제공하는 API로 분명하게 코드 변환이 이루어지는 것을 확인할 수 있었다. 코드 변환 완료 후, 개발자는 각 태스크 코드를 컴파일하여 실행 가능한 파일을 생성한다. 만약 POSIX 표준을 따르지 않는 다른 변환 규칙을 정의한다면 우리는 다른 RTOS로 확장이 가능한 것이다.

V. 결론

본 논문에서는 모델 주도형 접근 방법을 이용한 임베디드 소프트웨어 개발 방법과 자동화된 API 변환기에 대하여 기술하였다. RTOS가 제공하는 서비스들을 Generic API로 추상화 시켰으며 이를 이용하여 RTOS와 관련된 행동들을 중간 형태의 C-코드로 표현할 수 있었다. 이렇게 표현된 코드는 XML 변환 규칙을 참조하여 API 변환기에 의해 타겟 플랫폼에서 실제로 수행 가능한 코드로 변환된다. 제안된 방법은 MDA의 장점을 포함할 수 있도록 모든 변환 과정은 API 변환기에 의해 자동으로 수행되며 코드 변환인 중심이기 때문에 실시간 오버헤드를 줄일 수 있었다. 또한 변환 규칙 파일의 수정 및 추가를 통하여 다른 타겟 플랫폼으로 확장이 용이하도록 하였다.

제안된 API 변환기는 XML을 이용한 역변환 규칙이 좀 더 명확히 정의되어 기존 코드를 CIC 모델로 변환 가능한 역변환이 가능하도록 확장 되어야 한다. 정변환과 역변환 규칙은 서로 밀접한 관계를 형성하고 있다. 구체적인 변환 규칙을 제외한다면 정변환은 역변환의 반대 과정이라고 할 수 있다. 따라서 정변환 규칙을 자동으로 역변환 규칙으로 바꿔주는 툴을 사용할 수 있다면 좀 더 자동화된 개발 시스템 환경 구축이 가능할 것이다.

참고 문헌

[1] Balasubramanian K., Gokhale A., Karsai G., Sztipanovits J., and Neema S., "Developing Applications Using Model-Driven Design Environments," *Computer*, vol. 39, pp. 33-40, Feb. 2006.

[2] Koehler J., Hauser R., Kapoor S., Wu F. Y., and Kumaran S., "A Model-Driven Transformation Method," *Proc. of the 7th IEEE International Enterprise Distributed Object Computing Conf.*, pp. 186-197, 2003.

[3] Object Management Group Inc., MDA Guide v1.0.1, http://www.omg.org/, June 2003.

[4] Thomas O. Meservy and Kurt D. Fenstermacher, "Transforming Software Development: An MDA Road Map," *Computer*, vol. 38, pp. 52-58, 2005.
[5] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio, "A Model-driven Design Environment for Embedded Systems," *Design Automation Conf.*, pp. 915-918, July 2006.
[6] Anneke Kleppe, Jos Warmer, and Wim Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*, Addison Wesley, 2004.
[7] Stephen J. Mellor, Kendall Scott, Axel Uhl, and Dirk Weise, *MDA Distilled: Principles of Model-Driven Archicture*, Addison Wesley, 2004.
[8] Reational Rose-RT, http://www.ibm.com/, IBM.
[9] Rhapsody, http://www.ilogix.com/, I-Logix.
[10] TAU, http://www.telelogic.com/, Telelogic AB.
[11] HOPES, http://www.peace.snu.ac.kr/hopes/, 2005.
[12] Soonhoi Ha, "Hardware/Software Co-design of Multimedia Embedded Systems: PeaCE Approach," *white paper*, 2004.
[13] Dohyung Kim and Soonhoi Ha, "Static Analysis and Automatic Code Synthesis of flexible FSM Model," *Proc. of the 2005 Conf. on Asia South Pacific Design Automation*, pp. 161-165, Jan. 2005.
[14] The IEEE and The Open Group, The Open Group Base Specifications Issue 6 IEEE Std 1003.1 2004 Edition, http://www.unix.org/, 2004.
[15] Donald Lewine, *Posix Programmer's Guide: Writing Portable Unix Programs With the Posix. 1 Standard*, O'Reilly, 1991.
[16] Ji Chan Maeng, Jong-Hyuk Kim, and Minsoo Ryu, "An RTOS API Translator for Model-driven Embedded Software Development," *Proc. of the IEEE International Conf. on Embedded and Real-Time Computing Systems and Applications*, pp. 365-367, Aug. 2006.

박 병 률(Byeong-Ryul Park)

2006년 2월 국민대 전자공학부(학사)
2006년 3월~현재 국민대 전자공학과 석사 과정
※주관심분야 : 임베디드 시스템, 차량전자제어

맹 지 찬(Ji Chan Maeng)

2004년 2월 한양대 화학공학과(학사)
2006년 2월 한양대 정보통신학과(석사)
2006년 9월~현재 한양대 전자컴퓨터통신학과 박사과정
※주관심분야 : 실시간 시스템, 임베디드 S/W, RTOS

이 종 범(Jong-Bum Lee)

2006년 2월 국민대 전자공학부(학사)
2006년 3월~현재 국민대 전자공학과 석사 과정
※주관심분야 : 임베디드 시스템, 차량네트워크

유 민 수(Minsoo Ryu)

1995년 서울대 제어계측공학과(학사)
1997년 서울대 제어계측공학과(석사)
2002년 서울대 전기컴퓨터공학과(박사)
1998년 12월 ~ 2001년 10월 아이너스(주) 선임연구원
2001년 11월 ~ 2003년 2월 서울대 자동화연구소 연구원
2003년 3월 ~ 현재 한양대 정보통신학부 조교수
※주관심분야 : 실시간 시스템, 임베디드 S/W, RTOS

안 현 식(Hyun-Sik Ahn)

1982년 서울대 제어계측공학과(공학사)
1984년 서울대 제어계측공학과(공학석사)
1992년 서울대 제어계측공학과(공학박사)
현재 국민대 전자공학부 교수
※주관심분야 : 임베디드 시스템, 차량전자제어

정 구 민(Gu-Min Jeong)

1995년 서울대 제어계측공학과(공학사)
1997년 서울대 제어계측공학과(공학석사)
2001년 서울대 전기컴퓨터공학부 (공학박사)
현재 국민대 전자공학부 조교수
※주관심분야 : 임베디드 시스템, 차량전자제어