RTOS기반 임베디드 S/W를 위한 API 정변환/역변환기의 개발

An API Translator for RTOS-Based Embedded Software Considering Forward/Reverse Transformation

박 병 률*, 맹 지 찬**, 이 종 범*, 유 민 수***, 안 현 식****, 정 구 민*****
Byeong-Ryul Park, Ji Chan Maeng, Jong-Bum Lee, Minsoo Ryu, Hyun-Sik Ahn, Gu-Min Jeong

Abstract - In this paper, we present a model-driven approach for RTOS-based embedded software development and an automated tool that produces RTOS-specific code or RTOS-independent code. We define generic RTOS APIs (Application Programming Interface) that are not bound to any specific RTOS but provide most of typical RTOS services. Generic RTOS APIs can be used to describe application's RTOS-related behavior. The proposed API translator translates task code between C-code for specific RTOS and intermediate code using generic API. Also, the result can be extended to other RTOS's modifying XML transformation rule.

Key Words : MDA, RTOS, API Translator, Reverse Transformation, Embedded Software Development, POSIX

1. 서론

최근 하드웨어 개발이 발전함에 따라 소프트웨어의 복잡도가 증가하면서 모델 주도형 접근(Model-Driven Approach) 방법을 이용한 소프트웨어 개발 방법이 등장하였다. 이로 인해 소프트웨어 개발자들은 어플리케이션을 하드웨어, 또는 소프트웨어에 종속되지 않도록 추상화 시키는 작업이 필요하게 된다 [1]. 또한 자동화된 툴을 이용해 모델을 코드로 변환시킬 수 있게 되었다 [2]. 더불어 이러한 모델 주도형 접근 방법은 생산성과 유지보수 측면에서 많은 이점들을 제공한다.

MDA(Model Driven Architecture)는 객체 관리 그룹(Object Management Group)에 의해 제안되어 모델 주도형 소프트웨어를 개발하는데 있어 실질적인 표준이 되고 있다. MDA에서는 PIM(Platform Independent Model)으로 설계된 모델을 하나, 또는 그 이상의 PSM(Platform Specific Model)으로 변환하여, 마지막으로 변환기에 의해서 최종 소스를 생성한다. 하지만, 현재의 MDA는 주로 EJB, .NET등 주로 미들웨어 플랫폼에 초점이 맞추어져 있기 때문에 임베디드 소프트웨어 개발에 대한 지원이 미약하다 [3].

본 논문에서는 RTOS기반 임베디드 소프트웨어 개발을 위한 모델 주도형 접근 방법과 자동화된 툴에 대하여 기술하였다. 모델링 단계에서 RTOS와 관련된 행동을 추상화 할 수 있도록 Generic API(Application Programming Interface)들을 정의하고, Generic API들을 실행 가능한 'C'코드로 변환시켜주는 API 변환기를 개발하였다. [4]에서 제안한 패턴, 심벌, 룰은 서로 연관성이 존재함에도 불구하고 각각 다른 형태로 분리되어 있었지만 본 논문에서는 이들을 XML 형식으로 통합하여 가독성을 높이고 중복성을 제거하였다. 또한 [5]에서 제안한 통합 XML 구조를 정변환과 역변환으로 각각 분리함으로써 변환기의 성능을 높였다.

현재 산업 현장에서는 모델 주도형 소프트웨어 개발을 위해 많은 CASE(Computer-Aided Software Engineering) 툴들이 사용되고 있다. 하지만 이러한 툴들은 대부분 플랫폼에 종속적인 면을 반영하기 위해 가상머신을 이용하기 때문에 실시간 오버헤드를 동반한다. 본 논문에서 제안하는 방법은 API 변환이 중심이 되기 때문에 오버헤드가 적으며 모델 주도형 접근 방법을 이용하지 않고 개발된 소프트웨어에 적용할 수 있도록 쉽게 확장이 가능하다.

본 논문은 다음과 같이 구성된다. 2장에서 Generic API를 이용한 모델링 방법에 대해 알아본다. 3장에서 API 변환기의 설계 및 구현에 대해서 기술하고, 4장에서 결론을 맺는다.

2. Generic API를 이용한 모델링

2.1 Generic API를 이용한 모델 주도형 접근

본 연구는 HOPES(Hopes Of Parallel Embedded Software) 프로젝트의 결과물 중 일부이다. HOPES의 목표는 병렬 임베디드 소프트웨어를 개발하기 위한 환경을 제공하는 소프트웨어를 개발하는 것이다 [6].

그림 1은 MDA 방법과 Generic API를 이용한 모델 주도형 접근 방법을 보여준다. 그림 1(B)에서 보는바와 같이 제안된 방법은 3단계로 구성된다. 첫 단계에서는 어플리케이션의 데이터 흐름 모델과 Generic API들을 이용하여 ABM (Application Behavior Model)을 생성 한다. 다음 단계에서는

저자 소개
* 國民大學 電子工學科 碩士科程
** 漢陽大學 電子通信컴퓨터工學科 博士課程
*** 漢陽大學 電子通信컴퓨터工學科 敎授 • 工博
**** 國民大學 電子工學科 敎授 • 工博
***** 國民大學 電子工學科 助敎授 • 交信著者

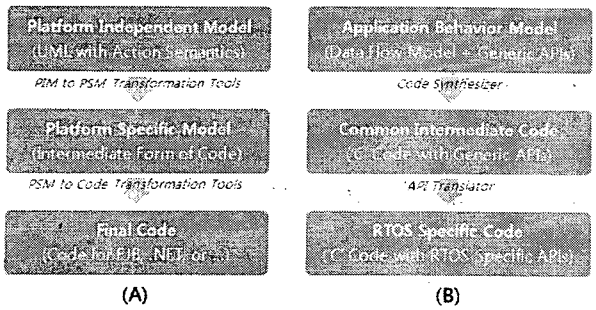

그림 1. (A)MDA 방법과 (B)제안된 방법.

코드 합성기를 통하여 Generic API들을 포함하는 중간 형태의 'C' 코드인 CIC(Common Intermediate Code)로 변환 된다 [7]. 마지막 단계에서 RTOS 타겟이 정해지고 API 변환기는 이 타겟에서 실행 가능한 코드로 변환을 수행한다.

2.2 Generic API의 정의

전형적인 RTOS는 태스크 관리, 메모리 관리, 파일 시스템 관리, I/O 관리 등의 서비스를 API형태로 제공한다. 우리는 이러한 API들을 추상화시켜 RTOS와 관련된 행동을 모델링을 할 수 있도록 Generic API를 정의하였다. 현재 사용되는 RTOS는 수백 개의 API를 제공하지만 본 논문에서는 IEEE에서 제정된 POSIX(Portable Operation System Interface for UNIX) 1003.1-2004 표준 [8]을 기반으로 하였다. POSIX는 UNIX에 기반을 두고 있는 일련의 표준 운영체제 인터페이스이다.

먼저 Generic API를 정의하기 위해 RTOS가 제공하는 서비스 그룹을 중심으로 POSIX API들을 분류하고 네트워크 분야, 실시간 분야를 추가하였다. 그리고 비슷한 기능을 가진 API들, RTOS와 관련이 없는 API들을 제거하여 좀 더 간결화 시켰다. 예를 들어 printf(), fprintf, sprintf() API는 출력 타겟이 서로 다르지만 모두 출력을 담당하는 함수이다. 이렇게 해서 모두 78개의 Generic API들이 정의되었으며 표 1에서 분류된 API의 일부를 보여주고 있다.

표 1. Generic RTOS APIs (총 78개)

분 류	Generic RTOS APIs
Task	FORK, THREAD_CREATE, ... (21 개)
Real-Time	MQ_SEND, MQ_RECEIVE, ... (15개)
Network	SOCK_ACCEPT, SOCK_SEND, ... (9개)
Memory	MALLOC, FREE, ... (4개)
File	OPEN, CLOSE, READ, ... (18개)
Device	SELECT, IOCTL, POLL (3개)
Others	CTIME, GETTIMER, RAND, ... (8개)

3. API 변환기의 설계 및 구현

Generic API를 이용하여 RTOS와 관련된 행동을 정의되고, Generic API는 API 변환을 통하여 타겟 RTOS에서 제공하는 완전한 API로 변환이 이루어진다. 이 과정은 API 변환기를 통하여 자동으로 수행되며 이번 장에서 어떻게 변환이 수행되는지 기술한다.

3.1 패턴과 심벌

사전 연구를 통하여 Generic API들은 몇 개의 API들로 이루어진 패턴을 형성하여 사용된다는 것을 발견하였다. 예를 들어, 파일을 열기 위해 OPEN() API를 사용했다면 파일을 닫기 위해 CLOSE() API가 반드시 사용된다는 것이다.

하나 또는 그 이상의 API들로 이루어지는 패턴은 각 API들의 부가 정보들과 함께 표현된다. 부가 정보들은 API의 반환 값의 존재 여부, 반환 값의 캐스팅 여부, 함수의 파라미터 개수 등이다. 이러한 부가 정보들은 변환 규칙에서 사용하기 쉽도록 심벌로 정의되며 순서대로 미리 정의된 @VAR 기호를 사용하여 마치 C언어의 배열처럼 사용되도록 하였다. 이렇게 부가 정보를 한꺼번에 표시함으로써 [5]에서 제안하는 XML 표현 방법보다 더 간결해졌기 때문에 손쉽게 확장이 가능하다.

```
...
<PATTERN name="OPEN0">
  <API name="OPEN" ret="0" cast="0" param="3">
    <RULE dir="INCLUDE" in="1">fcntl.h</RULE>
    <RULE dir="REPLACE">@VAR[0]=open(@VAR[1], @VAR[2])</RULE>
  </API>
  <API name="CLOSE" ret="0" cast="0" param="1">
    <RULE dir="INCLUDE" in="1">unistd.h</RULE>
    <RULE dir="REPLACE">close(@VAR[0]</RULE>
  </API>
</PATTERN>
...
```

(A) Multi-패턴

```
...
<PATTERN name="READ0">
  <API name="READ" ret="0" cast="0" param="3">
    <RULE dir="INCLUDE" in="1">unistd.h</RULE>
    <RULE dir="REPLACE">read(@VAR[0], @VAR[1], @VAR[2])</RULE>
  </API>
</PATTERN>
...
```

(B) Single-패턴

그림 2. OPEN0, READ0 패턴 정보 및 변환 규칙.

그림 2에서 보는 바와 같이 패턴을 이루지 않는 API들은 Single-패턴으로 분류하여 패턴을 이루는 다른 API들과 일관성을 유지하도록 하였다. API 변환기는 패턴 내의 API들을 검색하여 변환 대상 API 후보로 등록하게 되며 필요한 부가 정보들은 심벌 테이블에 등록하여 다음에 나오는 변환 규칙에서 사용한다.

3.2 변환 규칙

API 변환기는 각 패턴마자 정의된 변환 규칙을 참조하여 변환을 수행한다. 표 2에서 보여주는 바와 같이 변환 규칙을 표현하기 위하여 모두 7개의 명령어들이 사용된다. 사용자는 이 명령어들을 이용하여 구체적인 변환 규칙을 표현할 수 있다. 명령어와 함께 사용되는 부가 정보들은 각 명령어에 따라 다르다. 예를 들어, INCLUDE 명령어의 경우 외부에서 정의된 파일인지 내부 라이브러리인지를 판단하는 플래그가 필요하지만, DECLARE 명령어의 경우 전역 변수 선언인지 지역변수 선언인지를 판단하는 플래그가 필요하다.

그림 2에서 보는 바와 같이 개발자는 패턴 내의 API들과 심벌 테이블 정보를 이용하여 변환 규칙을 기술한다. API 변

표 2. 변환 규칙 명령어

명령어	역 할
INCLUDE	헤더파일을 포함한다.
DECLARE	변수를 선언한다.
INITIALIZE	초기화 코드를 삽입한다.
REPLACE	해당 코드를 지정된 코드로 대체한다.
INSERT	해당 코드를 삽입한다.
FINALIZE	종료화 코드를 삽입한다.
DELETE	해당 코드를 삭제한다.

환기는 패턴에서 검색된 변환 대상 API와 그와 관련된 정보들이 등록되어 있는 심벌 테이블을 이용하여 변환 구문을 생성한다. 더불어 헤더파일 추가, 변수 선언 및 초기화 코드를 삽입할 수 있다.

3.3 API 변환기 구현

API 변환기는 C언어로 작성되어 gcc로 컴파일 되어 구현되었다. 초기화, 입력 코드 해석, API 변환, 결과 코드 생성의 총 4단계 과정으로 동작한다. 초기화 단계에서는 패턴, 심벌, 변환 규칙이 기술된 통합 XML 파일을 파싱하여 API 변환 단계에서 참조할 수 있도록 준비한다. 입력 코드 해석 단계

```
...
if ((OPEN(infile_175, "./DIVX/friends.avi", O_RDONLY)) < 0)
  return;
SEEK(infile_175, 0L, SEEK_SET);
my_avi = AVI_open_input(infile_175, 1);
if (my_avi == NULL)
  return;
MALLOC(vidbuf, AVI_video_max(my_avi), char*);
...
```

⇩ *Transform*

```
...
if ((infile_175 = open("./DIVX/friends.avi", O_RDONLY)) < 0)
  return;
lseek(infile_175, 0L, SEEK_SET);
my_avi = AVI_open_input(infile_175, 1);
if (my_avi == NULL)
  return;
vidbuf = (char*)malloc(AVI_video_max(my_avi));
...
```

그림 3. API 변환기를 통한 코드 변환.

에서는 하나 이상의 입력 CIC 파일에서 API들을 파싱하여 변환이 필요한 Generic API를 분류할 수 있도록 준비한다. API 변환 단계에서는 로드 된 패턴, 심벌, 변환 규칙을 이용하여 각각 정의된 변환 규칙에 따라 변환 리스트를 생성한다. 마지막 결과 코드 생성 단계에서 최종 C 코드를 생성한다. 그림 2는 API 변환기에 의해 변환이 수행된 코드의 전·후를 보여주고 있다.

API 변환기를 구현하는데 있어 가장 중요한 목적은 변환기 외부에 정의된 통합 XML 변환 규칙을 수정함으로써 다른 RTOS로 확장이 가능하도록 하는 것이다. 심지어 코드에서 CIC로의 변환을 수행하는 역변환을 수행할 수 있다. 역변환을 이용하면 이종 RTOS간의 어플리케이션 변환을 자동으로 할 수 있다. 이를 위해 패턴, 심벌, 변환 규칙을 표현하기 위해 이미 데이터 표현의 효율성 면에서 검증된 XML을 이용하였다.

4. 결론

본 논문에서는 RTOS 기반 임베디드 소프트웨어 개발을 위한 모델 주도형 접근 방법과 정변환과 역변환을 수행할 수 있는 API 변환기에 대하여 기술하였다. RTOS가 제공하는 서비스들을 Generic API를 이용해 추상화 시켰으며, RTOS와 관련된 행동들을 표현하였다. 제안된 API 변환기는 XML 변환 규칙을 참조하여 Generic API를 포함하는 중간 형태의 CIC 태스크 코드로부터 타겟 RTOS API로의 변환(정변환), 또는 특정 RTOS API에서 CIC로의 변환(역변환)을 수행할 수 있다.

Acknowledgement

본 연구는 정보통신부의 IT Leading R&D Support Project의 지원으로 수행되었습니다.

참 고 문 헌

[1] Joao Paulo Almeida, Remco Dijkman, Marten van Sinderen and Luis Ferreira Pires, "On the Notion of Abstract Platform in MDA Development," *Proceeding of the 8th IEEE International Enterprise Distributed Object Computing Conference*, pp.253-263, 2004

[2] Jana Koehler, Rainer Hauser, Shubir Kapoor, Fred Y. Wu, and Santhosh Kumaran, "A Model-Driven Transformation Method," *Proceedings of the 7th IEEE International Enterprise Distributed Object Computing Conference*, (2003) 186-197

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

[4] Ji Chan Maeng, Jong-Hyuk Kim and Minsoo Ryu, "An RTOS API Translator for Model-driven Embedded Software Development," *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 363-367, 2006

[5] 박병률, 맹지찬, 이종범, 유민수, 안현식, 정구민, "임베디드 S/W 개발을 위한 RTOS API 변환기의 설계 및 구현," 정보 및 제어 학술대회 논문집, pp. 443-445 2006

[6] "HOPES", http://peace.snu.ac.kr/hopes/, 2005

[7] Dohyung Kim and Soonhoi Ha, "Static Analysis and Automatic Code Synthesis of Flexible FSM Model," *ASP-DAC 2005*, January 2005

[8] The Open Group, *The Open Group Base Specification Issue 6, IEEE Std 1003.1, 2004 Edition*, http://www.unix.org/, 2004

[9] Anneke Kleppe, Jos Warmer and Wim Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*, Addison Wesley, 2004