MDA에 기반한 실시간 운영체제 API 정변환/역변환기의 개발

論 文
56-12-31

A Forward/Reverse API Translator for Real-Time Operating System Based on a Model-Driven Approach

朴 秉 律* · 孟 知 燦** · 李 鐘 範* · 柳 珉 秀*** · 安 鉉 植§ · 鄭 求 珉†
(Byeong-Ryul Park · Ji Chan Maeng · Jong-Bum Lee · Minsoo Ryu · Hyun-Sik Ahn · Gu-Min Jeong)

Abstract - This paper presents 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 approach containing its advantages. 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 between a CIC using generic APIs and a C-code for specific RTOS. The proposed API translator converts them 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.

Key Words : API 변환기, MDA(Model Driven Architecture), 역변환, RTOS, 임베디드 소프트웨어 개발

1. 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]. 그러나, 현재의 MDA는 주요 대상이 EJB, Microsoft .NET 및 CORBA 등 타겟 플랫폼 위의 미들웨어이기 때문에 임베디드 소프트웨어 개발에 대한 지원이 미약하다. 대부분의 임베디드 시스템이 실시간성과 제한된 자원 등의 요구조건을 필요로 하기 때문에 임베디드 환경에서는 실시간 오버헤드를 고려해야만 하는 미들웨어 플랫폼 대신에 RTOS (Real-Time Operating System)가 요구된다.

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 이용하여 응용 프로그램의 모델을 만들고 자동으로 최종 코드를 생성한다. 하지만 이러한 툴들 역시 플랫폼과의 독립성을 이루기 위해 가상머신을 사용하기 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]. 정변환에 대한 기본 개념은 이전 연구 [14]에서 제안되었다. [14]에서 분리되었던 패턴, 심벌 및 룰을 본 논문에서는 XML을 이용해 통합하였다. 이를 바탕으로 역변환이 가능하도록 하였고 정변환과 역변환을 이용해 이종 RTOS간 응용프로그램 변환기를 개발할 수 있다.

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 연구 과제를 제시한다.

2. Generic API를 이용한 모델 기반 소프트웨어 개발

전형적인 RTOS들은 개발자들이 실시간 임베디드 시스템에서 응용프로그램을 개발할 수 있도록 여러 가지 서비스들

† 교신저자, 正會員 : 國民大學 電子工學科 助敎授
E-mail : gm1004@kookmin.ac.kr
* 正 會 員 : 國民大學 電子工學科 碩士課程
** 正 會 員 : 漢陽大學 電子通信컴퓨터工學科 博士課程
*** 正 會 員 : 漢陽大學 電子通信컴퓨터工學科 敎授 · 工博
§ 正 會 員 : 國民大學 電子工學科 敎授 · 工博
接受日字 : 2007年 7月 30日
最終完了 : 2007年 9月 21日

을 제공한다. 이러한 서비스들은 태스크 관리, 메모리 관리, 파일 시스템 관리 및 I/O 관리 등을 포함하는 API 형태로 제공된다. 우리는 이러한 API들을 특정 플랫폼에 종속적이지 않도록 추상화 시켜 'Generic API'로 정의하였다. POSIX 표준 1003.1-2004 [15]을 기반으로 서로 관련 있는 API들끼리 그룹 짓고, 서로 비슷한 API들과 RTOS와 직접적으로 관련 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 API들이기 때문에 하나의 PRINT() API로 정의될 수 있다. 이렇게 정의된 Generic API들을 RTOS가 제공하는 서비스 그룹을 중심으로 분류하고, 이에 네트워크 분야와 실시간 분야를 추가하여 좀 더 세분화 시켰다. 표 1은 Generic API들과 그 분류를 보여주고 있다.

표 1 Generic RTOS APIs (총 78개).
Table 1 Generic RTOS APIs (78 APIs in total).

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

Generic API를 이용한 모델 기반 접근 방법은 크게 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.

3. API 변환기의 설계 및 구현

Generic API들은 RTOS와 관련된 행동이 구체적으로 명시되고 나서, 타겟 RTOS에서 제공하는 API들과 상호 변환된다. 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.

제안된 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. 주 태스크에서는 각 태스크들의 _init() 함수, _go() 함수 그리고 wrapup() 함수를 차례로 호출함으로써 태스크가 정상적으로 수행될 수 있다.

이번 장에서는 API 변환기가 다른 RTOS로의 확장을 고려하면서 타겟 코드로의 변환을 구체적으로 어떻게 수행하는지 기술한다.

3.1. 패턴, 심벌 및 변환 규칙

선행 연구 [11]를 통하여 몇몇의 API들은 패턴을 형성하여 사용된다는 것을 발견하였다. 그리고 API는 일반적으로 함수의 파라미터, 파라미터의 개수, 반환 값 또는 반환 값의 형 변환 등의 부가정보들과 함께 호출된다. 우리는 API의 순차적인 호출을 '패턴'으로, API호출과 함께 제공되는 부가정보들을 '심벌'이라고 정의한다 [11].

```
<!--
        MALLOC(buf_ptr, buf_size, type_cast);
        FREE(buf_ptr);
                ⇓
        #include <stdlib.h>
        buf_ptr = (type_cast)malloc(buf_size)
        #include <stdlib.h>
        free(buf_ptr)
-->
<PATTERN name="MALLOC0">
  <API name="MALLOC" rct="0" cast="0" param="3">
    <RULE dir="INCLUDE" in="1">stdlib.h</RULE>
    <RULE dir="REPLACE">@VAR[0] = (@VAR[2])malloc(@VAR[1])</RULE>
  </API>
  <API name="FREE" rct="0" cast="0" param="1">
    <RULE dir="INCLUDE" in="1">stdlib.h</RULE>
    <RULE dir="REPLACE">free(@VAR[0])</RULE>
  </API>
</PATTERN>
```

그림 1 정변환에서 XML을 이용한 패턴, 심벌 및 변환 규칙의 표현.
Fig. 1 Description of pattern, symbol and transformation rule using XML in forward transformation.

우리는 이러한 패턴과 부가정보들을 그림 1에서 보는바와 같이 XML을 이용하여 변환 규칙 파일에서 표현한다. 패턴과 심벌 정보들은 변환이 완료될 때까지 심벌 테이블에 등록되고, 변환 구문을 생성할 때 사용된다. 이를 위해 변환 규칙에서는 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 의미한다.

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. 이러한 변환 규칙도 XML을 이용하여 기술되며, 그림 1은 MALLOC0 패턴의 변환 규칙을 표현하고 있다. API 변

환기는 심벌들을 심벌 테이블에 등록되어 있는 값으로 대체하여 실제 변환 구문을 생성한다. 역변환을 위해서는 정변환의 변환 규칙과 반대로 역변환의 변환 규칙도 정의되어야 한다. 이렇게 변환 규칙 명령어를 사용하면 헤더파일 추가, 변수 선언, 특정 코드 삽입 또는 삭제가 가능하다.

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

명령어	의 미
INCLUDE	주어진 헤더 파일을 추가한다.
DECLARE	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.

3.2. API 변환기의 구현

API 변환기는 C언어로 구현되었으며 정변환과 역변환을 모두 수행할 수 있다. 그림 2에서 보듯이 정변환에서는 중간 형태의 코드인 CIC를 입력으로 받아 타겟 코드로 출력을 내보내며, 역변환에서는 이와 반대로 수행된다. API 변환기는 다음의 4가지 툴로 구성된다. 입력 파일 파서, XML로 기술된 변환 규칙 파일 파서, 패턴, 심벌 그리고 변환 규칙을 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.

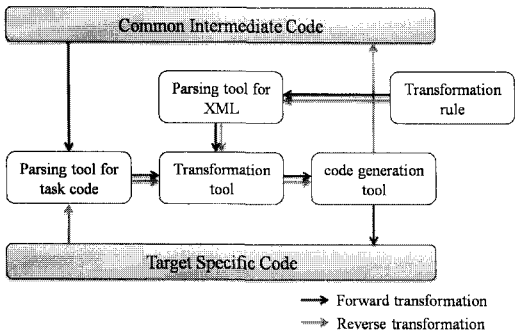

그림 2 API 변환기의 블록 다이어그램.
Fig. 2 Block diagram of API translator.

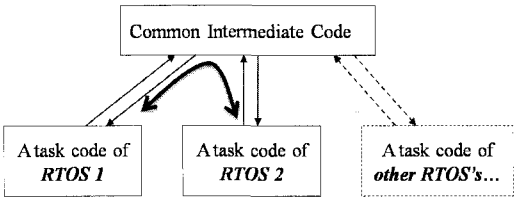

그림 3 제안된 API 변환기를 이용한 이종 RTOS간의 코드 변환.
Fig 3 Code transformation for different RTOS's using the proposed API translator.

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.

제안된 API 변환기를 이용하면, 특정 플랫폼에서 이미 개발된 응용 프로그램 코드를 다른 플랫폼의 태스크 코드로 변환시킬 수 있다. 그림 3에서 보는바와 같이, 응용 프로그램의 태스크 코드를 CIC로 변환하고, 다시 다른 플랫폼에서 수행할 수 있도록 태스크 코드를 변환한다. 이를 이용하면 이종 RTOS간의 응용 프로그램 변환기를 개발할 수 있다.

4. 실험 결과

API 변환기의 동작을 확인하기 위하여 간단한 쓰레드 동기화 예제를 이용하여 실험을 수행하였다. 예제 코드는 정변환/역변환 규칙을 이용하여 이종 RTOS간의 코드 변환을 수행한다. RTOS는 POSIX 표준을 따르는 Linux와 VPOS 2.0을 사용하였다. VPOS 2.0은 HOPES 프로젝트에서 개발된 RTOS이다 [11]. 이 예제에서는 앞서 정의한 3개의 쓰레드 관련 C 파일이 사용되었다. API 변환기는 Linux에서 작성된 C 파일들을 CIC 파일로 변환하고(역변환), 다시 반대로 VPOS 2.0에서 수행 가능한 C 파일로 변환한다(정변환).

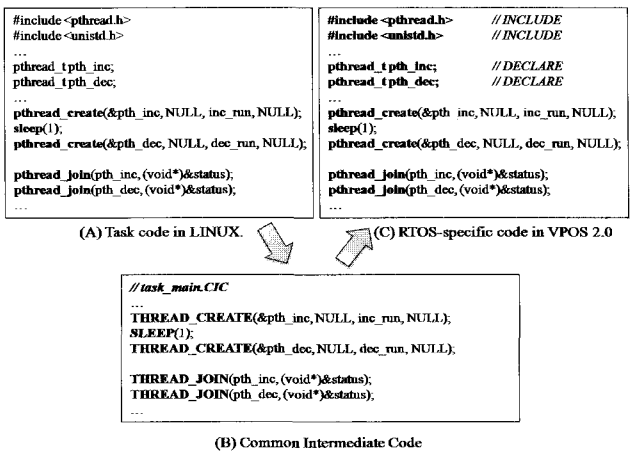

그림 4 API 변환기를 이용한 코드 변환.
Fig 4 Code translation using an API translator.

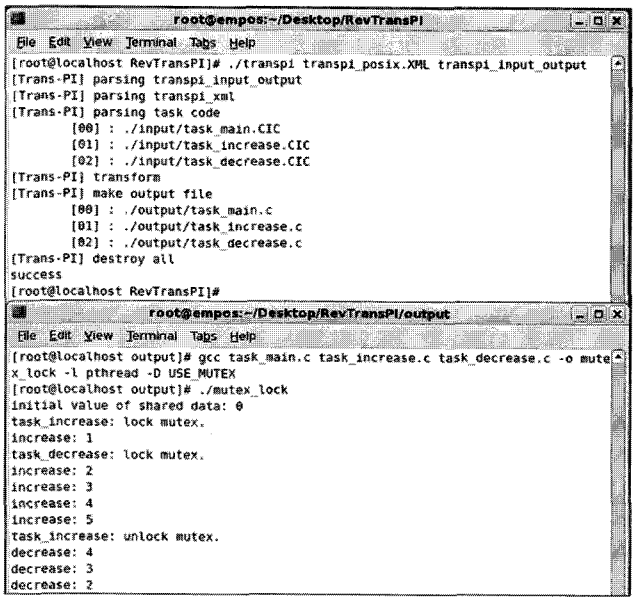

그림 5 동기화 예제의 스크린샷.

Fig 5 Snapshot of synchronization example.

변환 결과가 그림 4와 그림 5에서 보이고 있다. 필요한 헤더 파일들이 추가 되었고, 쓰레드와 관련된 변수들이 선언되었으며, Generic API들이 VPOS 2.0에서 제공하는 API들로 변환되었다. 그림 5에서 보는바와 같이 이 과정은 API 변환기에 의해 자동으로 수행된다. Linux 태스크 코드는 역변환을 이용하여 CIC로 변환되고, 이 CIC 코드는 다시 정변환을 이용하여 VPOS 2.0 태스크 코드로 변환된다. 두 RTOS가 모두 POSIX 표준을 따르기 때문에 변환되기 전과 후의 쓰레드 관련 API가 동일하다. 그러나 우리는 API 변환기가 정변환과 역변환을 이용하여 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.

5. 결 론

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 중간형태의 C-코드로 표현할 수 있었다. 이렇게 표현된 코드는 API 변환기에 의해 타겟 플랫폼에 맞는 코드로 변환(정변환)된다. 반대로 특정 플랫폼에서 작성된 코드를 어느 플랫폼에도 속해 있지 않은 중간 형태의 코드로 변환(역변환)할 수 있다.

코드 변환에 있어 정변환과 역변환은 서로 밀접한 관계를 형성하고 있다. 이는 정변환 규칙을 자동으로 역변환 규칙으로 바꿔주는 툴을 사용한다면 좀 더 자동화된 개발 시스템 환경을 구축할 수 있다는 것을 의미한다. 또한, 본 논문의 결과물은 이종 RTOS간의 응용프로그램 변환기를 개발하는데 사용되어질 수 있다. 디바이스 드라이버에 대한 코드변환이 좀 더 고려된다면 제안된 API 변환기는 플랫폼에 종속적인 코드는 모두 변환할 수 있다.

참 고 문 헌

[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] 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.

[15] The IEEE and The Open Group, The Open Group Base Specifications Issue 6 IEEE Std 1003.1 2004 Edition, http://www.unix.org/, 2004.