임베디드 S/W 개발을 위한 RTOS API 변환기의 설계 및 구현

Design and Implementation of an RTOS API Translator for Embedded Software Development

박 병 률*, 맹 지 찬**, 이 종 범*, 유 민 수***, 안 현 식****, 정 구 민*****
Byung-Ryuel 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 to RTOS(Real Time Operating System)-based embedded software development and an automated tool that can produce RTOS-specific code. we defined generic RTOS APIs(Application Programming Interface) that are not bound to any specific RTOS but can provide most of typical RTOS services. The generic RTOS APIs can be used as a means for describing application's RTOS-related behavior from design stage. Out tool, called Trans-PI, is able to produce specific 'C' code aimed at POSIX(Portable Operating System Interface for UNIX)-compliant RTOSs. And it is also configurable to target other RTOSs that do not conform to the POSIX standard.

Key Words : Model Driven Architecture(MDA), RTOS, API Translator, POSIX

1. 서론

하드웨어 개발이 진보됨에 따라 소프트웨어 개발 또한 더 복잡하고 거대한 시스템으로 바뀌게 되었고, 이로 인해 모델 주도형 접근 방법이 필요하게 되었다. 그리하여 개발자는 모델 주도형 접근 방법을 사용하게 되었고, 특정 하드웨어와 소프트웨어 환경에 종속되지 않는 추상 모델을 만드는데 더 중점을 두게 되었다 [1]. 또한 자동화된 변환 툴은 모델에서 코드로의 변환을 통한 최종 작업에 사용되어질 수 있다 [2].

모델 주도형 구조(Model Driven Architecture, MDA)는 소프트웨어 설계에 대한 기획을 보강하고, 소프트웨어 개발의 효율성을 증진시키기 위하여 객체 관리 그룹(Object Management Group, OMG)에 의해 제안되었다. MDA는 소프트웨어 개발의 효율성, 서로 다른 시스템간의 이식성, 각 컴포넌트들간의 상호 작용성, 유지 보수 및 문서화에 많은 이점을 가지고 있다 [2]. 이를 위하여 MDA는 플랫폼 독립적인 모델(Platform Independent Model, PIM)과 플랫폼 종속적인 모델(Platform Specific Model, PSM)의 두 가지 중요한 모델을 지원한다. PIM은 어떠한 특정 기술에도 종속되지 않는 추상화된 모델로 정의되며, 대부분 UML(Unified Modeling Language)에 의해서 구현된다. 반면에 PSM은 특정한 하나의 기술에 의해서 생성된 모델이다. 즉, PSM은 PIM과 특정 플랫폼에서 사용되거나 적용되고 있는 프로그램의 세부사항이 함께 결합되어 있는 중간 형태라고 볼 수 있다. MDA 개발 과정에서 PIM은 최종 코드를 생성하기 위하여 하나 또는 그 이상의 PSM으로 변환된다. 그러므로 MDA 접근 방법을 통하여 실행 가능한 코드를 생성하기 위해서는 그림 1(A)과 같은 2단계의 과정이 필요하다. 하지만 이러한 MDA는 불행하게도 EJB, 웹 서비스등과 같은 미들웨어 타겟 플랫폼이 주 대상이기 때문에 실시간 운영체제(Real Time Operating System, RTOS)기반 임베디드 소프트웨어 개발에 대한 지원이 미약하다 [3].

본 논문에서는 RTOS기반 임베디드 소프트웨어 개발과 관련된 모델 주도형 접근 방법과 자동 코드 변환 툴에 대하여 기술하였다. 제안된 접근 방법은 RTOS 플랫폼 환경이 강조된 MDA의 변형으로 보여 진다. 이를 위하여, 특정 RTOS에 속해있지 않지만 전형적인 RTOS의 서비스를 대부분 제공할 수 있는 Generic API(Application Programming Interface)들을 정의하였다. 이러한 Generic API들은 모델을 디자인하는 과정에서 RTOS와 관련된 행동을 정의하는데 사용되며, 이를 특정 RTOS의 API로의 변환을 수행하는 자동화된 툴을 개발하였다. TansPI라 불리는 이 툴은 POSIX(Portable Operating

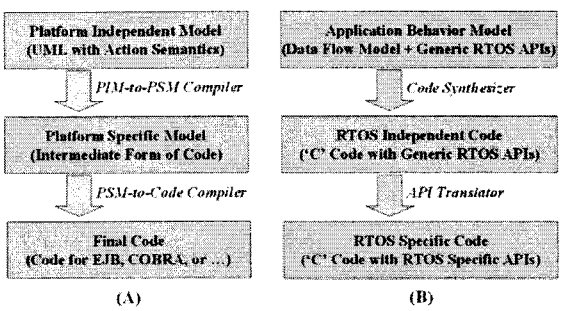

그림 1. (A) MDA와 (B) 제안된 방법.

저자 소개
* 國民大學 電子工學科 碩士課程
** 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 教授 • 工博
**** 國民大學 電子工學科 教授 • 工博
***** 國民大學 電子工學科 助教授 • 交信著者

System Interface for UNIX)를 따르는 RTOS들을 대상으로 'C' 코드를 만들어 낼 수 있으며, POSIX 표준을 따르지 않더라도 확장이 가능하다.

그림 1(B)에서와 같이 제안된 방법은 3단계로 구성된다. 첫 단계에서 개발자는 데이터 흐름 모델 [4][5]과 Generic RTOS API를 통하여 프로그램의 행동과 관련된 모델을 생성한다. 다음 단계에서는 코드 합성기가 HOPES [6][7]를 통하여 코드의 중간 형태인 RIC(RTOS Independent Code)를 생성하며, 마지막 단계에서 TransPI를 통하여 RSC(RTOS Specific Code)로 변환되어진다. 이러한 접근 방법은 변환이 중심이기 때문에 실시간 오버헤드가 적으며, 역변환을 이용하여 모델 주도형 접근 방법을 사용하지 않고 개발된 소프트웨어로의 확장이 가능하다.

본 논문은 다음과 같이 구성된다. 2장에서는 Generic API를 정의하여 RTOS API의 행동을 중심으로 모델링하는 방법을 알아보고, 3장에서 API변환기인 TransPI의 설계와 구현을 기술하고 4장에서 결론을 맺는다.

2. Generic RTOS API를 이용한 모델 주도형 접근

전형적인 RTOS는 태스크 관리, 메모리 관리, 파일 시스템 관리, I/O 관리등의 서비스를 제공한다. 이러한 RTOS 서비스들은 API형태로 제공되므로 프로그램에서 사용 가능하다. Generic API는 이러한 API들을 좀 더 추상화 시켜 RTOS와 관련된 행동을 모델링 할 수 있도록 정의된 API이다. 보통의 RTOS는 수백 개의 API를 제공하지만 본 논문에서는 POSIX 1003.1-2004 표준 [8]을 고려대상으로 하였다. POSIX는 IEEE에 의해 제정된 UNIX기반 소프트웨어를 정의하기 위한 표준 API들의 모음이다. 단순히 OS와 관련된 서비스에 한정되지 않고 프로그램의 전반적인 부분을 모두 포함하도록 설계되어졌다.

POSIX가 기반인 Generic RTOS API를 정의하기 위해서 먼저 POSIX API들 중에서 서로 관련 있는 API들끼리 그룹을 지었다. 전형적인 RTOS 서비스로 그룹을 분류하고 네트워크, 실시간 분야와 기타 분야를 추가하였다. 그리고 행동이 중복되는 API들과 RTOS와 관련 없는 API들을 제거하여 좀 더 간결화 시켰다. 예를 들어, printf(), fprintf()와 sprintf() API는 출력 타겟이 서로 다르지만 모두 출력을 담당하는 함수들이다. 이렇게 정의된 총 78개의 API들은 표 1과 같다. HOPES에서는 정의된 Generic API를 이용하여 프로그램의 행동과 관련된 모델을 설계하고, 코드 합성기를 통해 RIC형태의 중간 코드를 얻을 수 있다.

표 1. Generic RTOS APIs

Category	Generic RTOS APIs
Task	FORK, THREAD_CREATE, ... (21개)
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. TransPI 설계 및 구현

본 논문에서 제안하는 API 변환기 구현에서 가장 중요하게 요구되는 사항은 효율적인 확장성이다. TransPI는 POSIX 기반이 아닌 RTOS로의 확장을 위하여 외부에 정의된 변환 규칙을 사용한다. 개발자는 간편성, 개방성, 확장성이 뛰어난 XML(Extensible Markup Language)을 이용하여 정의된 외부 정보들을 수정하거나 새롭게 추가함으로써 원하는 시스템으로 손쉽게 변환할 수 있다.

3.1 패턴과 심볼

사전 연구를 통하여 Generic API들이 몇 개의 API들로 이루어진 패턴을 형성한다는 것을 발견하였다. 예를 들어, 파일을 열기 위하여 OPEN() 함수를 사용했다면 이후에 파일을 닫기 위하여 CLOSE() 함수가 사용 된다는 것이다. 이러한 API들은 함수 파라미터등의 서로 관련된 정보가 존재하게 되고 이러한 정보를 심볼로 정의한다. 후에 심볼 정보는 심볼 테이블로 구성되며 변환 규칙에서 변수처럼 사용된다.

패턴은 그림 2에서 보는 바와 같이 하나의 API가 단독으로 이루어진 Single-패턴과 여러 개의 API들이 패턴을 이루는 Multi-패턴으로 나누어진다. 패턴은 모든 변환이 완료된 후에도 여전히 유지되므로 역변환을 고려하기 위하여 변환 전의 Generic-패턴과 변환 후의 Specific-패턴이 함께 정의되어야 한다. 패턴이 변환되는 방향을 수정함으로써 역변환 구현이 가능하기 때문이다.

```
...
<pattern name="READ0">
  ...
  <generic count="1">
    <function>
      <name>READ</name>
      <parameter count="3">
        ...
      </paramenter>
    </function>
  </generic>
  <specific count="1">
  ...
  </specific>
  ...
</pattern>
...
```

(A) Single-Pattern

```
...
<pattern name="OPEN0">
  ...
  <function>
    <name>OPEN</name>
    <parameter count="3">
      ...
    </paramenter>
  </function>
  <function>
    <name>CLOSE</name>
    <parameter count="1">
    ...
    </parameter>
  </function>
  ...
</pattern>
...
```

(B) Multi-Pattern

그림 2. READ0, OPEN0 패턴 정보.

3.2 변환 규칙

TransPI는 각각의 패턴마다 정의된 변환 규칙을 이용하여 실제 변환을 수행한다. 변환 규칙에서는 총 7개의 명령어가 사용되고, 표 2는 각 명령어의 종류와 역할을 보여준다. 개발자는 ATL 명령어를 이용하여 좀 더 구체적인 변환 방법을 제시할 수 있다. 그림 3은 OPEN0 패턴의 변환 규칙을 보여주고 있다. 각 명령어에 따른 해당 동작 수행하기 위하여 심볼 정보가 이용된다. @VAR 기호는 패턴과 심볼 정보를 통해 만들어진 심볼 테이블을 의미한다. @GEN_FUN 기호는 패턴에서 정의된 Generic 패턴 API이며, @SPE_FUN 기호는

표 2. ATL 명령어

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

패턴에서 정의된 Specific-패턴 API이다.

```
...
<rule>
  <directive name="INCLUDE">
    <scope>INTERN</scope>
    <file>stdio.h</file>
  </directive>
  ...
  <directive name="REPLACE">
    <src>@GEN_FUN[0](@VAR[0], @VAR[1], @VAR[2])</src>
    <dst>@VAR[0] = @SPE_FUN[0](@VAR[1], @VAR[2])</dst>
  </directive>
  <directive name="REPLACE">
    <src>@GEN_FUN[1](@VAR[0])</src>
    <dst>@SPE_FUN[1](@VAR[0])</dst>
  </directive>
</rule>
...
```

그림 3. OPEN0 패턴의 변환 규칙.

3.3 TransPI 구현

TransPI는 C언어로 구현되었으며 코드 해석, 관련 정보 수집, 변환 수행 및 코드 생성의 총 4단계 과정으로 동작한다. 코드 해석 단계에서는 RIC를 해석하고, 관련 정보 수집 단계에서 패턴과 심볼 정보들을 통해 변환 대상 후보를 수집하고 심볼 테이블을 생성한다. 이렇게 수집된 정보를 통해 변환수행 단계에서는 실제 변환이 수행되고 마지막 코드 생성 단계에서 'C'형태의 RSC를 생성한다. 개발자는 패턴 정보에 있는 Generic-패턴을 Specific-패턴으로 변환함으로써 정변환을 수행하고, 반대로 역변환 수행도 가능하다. 또한 패턴 정보와 변환 규칙을 재정의 함으로써 다른 RTOS로의 확장이 가능하다. 그림 4는 TransPI를 이용한 변환 전과 후를 보여주고 있다. 파일의 입출력을 담당하는 Generic API들이 POSIX기반 Specific API들로 변환된 것을 확인할 수 있다.

변환 전 입력 RIC (task_1.RIC)

```
...
OPEN(outFile_212, "dev/audio", O_WRONLY);
if (audio_init(outFile_212) == -1)
{
    CLOSE(outFile_212);
    ...
}
...
```

⇩ ***Transformation***

변환 후 출력 RSC (task_1.C)

```
...
outFile_212 = open("dev/audio", O_WRONLY);
if (audio_init(outFile_212) == -1)
{
    close(outFile_212);
    ...
}
...
```

그림 4. OPEN0 패턴의 POSIX기반 RTOS API로의 변환.

4. 결론

본 논문에서는 RTOS 기반의 임베디드 소프트웨어 개발을 위한 모델 주도형 접근 방법과 자동화된 RTOS API 코드 변환 툴에 대하여 기술하였다. 우선 대부분의 RTOS에서 제공하는 서비스를 모두 포함할 수 있는 추상화된 Generic API를 정의하였다. 또한 TransPI를 통해 특정 RTOS 임베디드 시스템에서 수행 가능한 코드를 생성할 수 있었다.

본 논문에서 제안된 방법은 다음 2가지의 향후 연구 과제가 필요하다. 첫째, 코드에서 모델로의 변환을 위하여 역변환에 대한 좀 더 명확한 정의가 필요하다. 둘째, 다른 모델링 언어와의 상호 작용을 위하여 현재 널리 사용되는 MDA 프레임웍에 적용될 수 있도록 해야 할 것이다. 이러한 연구는 임베디드 소프트웨어를 개발하는데 많은 도움을 줄 수 있을 것이다.

Acknowledgement

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

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," *Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference,* pp.253-263, 2004

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

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

[4] David Harel. "Statecharts: A visual formalism for complex systems," *Science of Computer Programming,* 8(3):231-274, June 1987

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

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

[7] Soonhoi Ha, "Hardware/Software Co-design of Multimedia Embedded Systems: PeaCE Approach," white paper, 2004

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

[9] Ji Chan Maeng, Jong-Hyuk Kim and Minsoo Ryu, "An RTOS API Translator for Model-driven Embedded Software Development", *Proceedings of the 12th IEEE International Conference,* pp.363-367, 2006