제25회 한국정보처리학회 춘계학술발표대회 논문집 제13권 제1호 (2006. 5)

RTOS-기반 임베디드 소프트웨어를 위한 모델기반 개발방법

맹지찬, 김종혁, 유민수
e-mail : {jcmaeng, jhkim, msryu}@rtcc.hanyang.ac.kr

Model-Driven Development of RTOS-based Embedded Software

Ji Chan Maeng, Jong-Hyuk Kim, and Minsoo Ryu
Real-Time Computing and Communications Laboratory, Hanyang University

요 약

본 논문에서는 RTOS 기반 임베디드 소프트웨어 개발에 적합한 모델기반 방법론을 제안하고 이와 함께 개발된 자동코드생성 도구를 기술한다. 현재까지 알려진 대표적인 모델기반 방법론으로는 OMG (Object Management Group)의 MDA (Model-Driven Architecture)가 있으며, MDA 에서는 EJB, 웹서비스, .NET, 그리고 CORBA 와 같은 미들웨어 플랫폼을 대상으로 하는 응용 소프트웨어의 개발을 지원한다. 하지만, 통상적인 임베디드 시스템은 실시간성에 대한 요구조건은 물론 성능과 자원활용에 있어 많은 제약을 가짐에 따라 상당수의 임베디드 시스템은 미들웨어를 사용하지 않고 RTOS 상에서 직접 수행되도록 개발되고 있다. 이에 따라 본 연구에서는 MDA 방법론을 확장하여 플랫폼 의존적인 모델 (PSM, Platform Specific Model) 단계에서 추상화된 RTOS 행위를 표현할 수 있도록 추상 RTOS API (Generic RTOS API)를 정의하고, 아울러 추상화된 RTOS 행위를 자동으로 변환하여 C 코드를 생성해주는 도구인 TransPI 를 함께 제시한다.

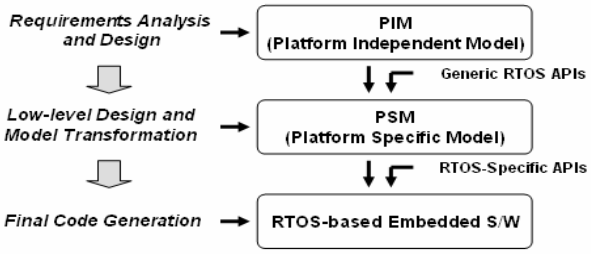
1. 서론

임베디드 소프트웨어의 복잡도가 증가함에 따라 이에 대한 해결책으로 모델기반 개발방법에 대한 관심이 점점 높아지고 있다. 모델기반 방식에서는 개발자가 높은 추상화 단계에서 업무를 수행할 수 있으므로 특정 하드웨어나 소프트웨어 플랫폼에 대한 고려 없이 단지 추상화된 어플리케이션 모델 작성에만 집중하게 된다 [4]. 모델기반 방법론 가운데 OMG (Object Management Group)의 모델기반 아키텍처 (MDA, Model-Driven Architecture)가 가장 대표적이다 [1]. MDA 의 주된 목표는 추상화된 시스템의 행위 및 기능을 상세 구현과 분리하는 것이다. 이를 위해 MDA 는 플랫폼 독립적인 모델 (PIM, Platform Independent Model)과 플랫폼 의존적인 모델 (PIM, Platform Specific Model) 개념을 제공한다. PIM 은 구현 기술과 같은 특정 플랫폼에 독립적이며 개발자가 한번 PIM 을 만들면, 자동으로 코드를 생성하기 위해 각 플랫폼에 대한 PSM 으로 변환된다.

아쉽게도, MDA 에서는 RTOS (Real-Time Operating System)기반 임베디드 소프트웨어 개발에 대해서는 특별히 기술하지는 않고 있다. MDA Guide [5]에 기술된 바에 따르면, MDA 는 EJB, 웹서비스, .NET, 그리고 CORBA 와 같은 미들웨어 플랫폼을 타겟으로 한다. 일반적인 임베디드 시스템에서는 실시간성에 대한 요구조건은 물론 성능과 자원활용에 있어 많은 제약을 가지며 이러한 요구사항이 RTOS 에서는 지켜지지만, 미들웨어를 사용하는 경우에는 실시간성도 보장할 수 없을 뿐만 아니라 그에 따른 추가 비용도 감수해야 한다 [2]. 이로 인해 상당수의 임베디드 어플리케이션들이 미들웨어 플랫폼이 없는 RTOS 위에서 직접 수행되고 있다.

본 논문에서는 RTOS 기반의 임베디드 소프트웨어 개발에 적합한 모델기반 개발방법과 이를 지원하기 위한 자동코드생성도구인 TransPI 를 기술한다. 그림 1 에서 보는 바와 같이, 본 논문에서 제시된 방법은 MDA 와 매우 유사하나 PSM 단계에서 추상화된 RTOS 행위를 표현할 수 있고 마지막 단계에서는 실제 특정 RTOS 용 구현물을 생성할 수 있다는 차이점이 있다. 본 연구의 구체적인 결과물은 다음과 같다. 첫째로, 추상화된 RTOS 행위를 표현하기 위해 대부분의 일반적인 RTOS 서비스들을 포함하면서도 특정 RTOS 에 한정되지 않는 추상화된 RTOS APIs

(Application Programming Interfaces)를 정의하고, 둘째로, 추상화된 RTOS 행위로부터 실제 RTOS 용 코드를 생성하는 도구인 TransPI 를 제시한다.



(그림 1) 제안된 모델기반 개발 과정

본 논문의 나머지 부분은 다음과 같이 구성되어 있다. 2 장에서는 추상화된 RTOS API 의 정의와 추상화된 RTOS 행위로 적용과정을 기술하며, 3 장은 모델에서 코드로 변환하는 도구의 구현에 대해 다룬다. 4 장에서는 실험 결과를 싣고, 마지막으로 5 장에서는 본 논문에 대한 간략한 요약과 함께 끝을 맺는다.

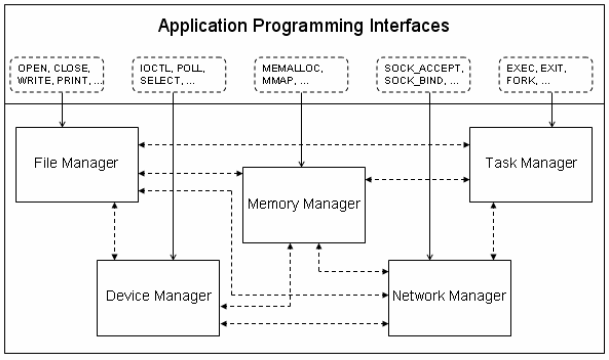
2. RTOS 행위 모델링을 위한 추상화된 RTOS APIs

일반적인 RTOS 는 태스크 관리, 메모리 관리, 파일 시스템 관리, 그리고 I/O 관리와 같은 다양한 종류의 서비스를 제공하며, RTOS 서비스들은 API 의 형태로 제공된다. 즉, 소켓 API 를 통해 네트워킹을 사용하며 메시지 큐 API 를 통해 프로세스간 통신을 사용하게 된다. 본 연구에서는 RTOS 행위를 표현하기 위한 추상화된 RTOS API 를 정의한다. 이는 RTOS 행위를 PSM 에 통합하기에도 용이하다는 이점이 있다. 하지만, 수많은 RTOS 가 저마다 각각 문법이나 의미가 다른 수백 개 이상의 API 를 제공하기 때문에, 본 연구에서는 POSIX 1003.1-2004 표준을 토대로 추상화된 RTOS API 를 정의한다.

2.1 POSIX 1003.1-2004 표준 개요

POSIX 는 IEEE 에서 UNIX 기반 소프트웨어를 위해 정의한 표준 API 집합이다. POSIX 의 최신 버전은 POSIX Standard 1003.1-2004 이며 Base Definitions volume, System Interfaces volume, Shell and Utilities volume, Rationale (Informative) volume 의 네 부분으로 구성된다.

본 논문에서, 우리는 1023 개로 구성된 실제 OS API 를 정의한 시스템 인터페이스 부분을 기반으로 하고 있다. POSIX 시스템 인터페이스는 8 개로 분류할 수 있으나 POSIX 표준은 OS 관련 서비스 이외에도 어플리케이션 프로그래밍에 대한 모든 부분들을 포함하도록 설계되어 여기에는 sin(), cos()같은 수학 함수처럼 직접 OS 의 지원을 필요로 하지 않는 수많은 API 가 포함되어 있다. 이는 POSIX API 집합이 표준 C 라이브러리 함수 (ISO/IEC 9899:1999)의 상위 집합이 되기 때문이며 수학 함수는 표준 C 라이브러리에서 (math.h) 지원된다. 또한, POSIX API 는 유사한 기능을 하는 API 를 포함한다. 예를 들어, printf(), fprintf(), sprintf()는 출력의 대상이 다르다는 점을 제외하고는 형식화된 데이터 출력이라는 동일한 기능을 갖는다.

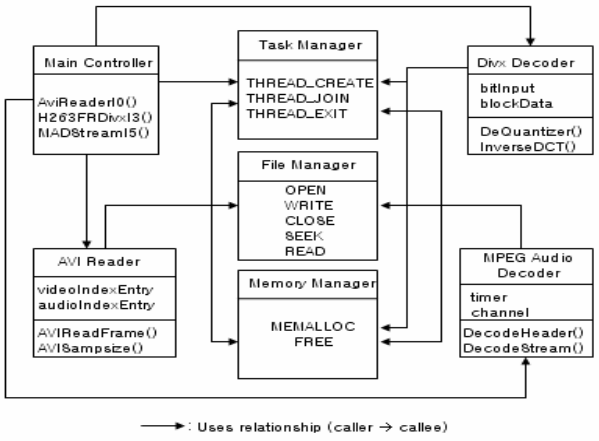


(그림 2) 일반 RTOS 구조

2.2 추상화된 RTOS API

추상화된 RTOS API 에 대한 정의는 그림 2 에서 보여지는 일반적이고 간단한 RTOS 의 구조에 기반하고 있다. 중심부에는 태스크 관리자, 메모리 관리자, 파일 관리자, 네트워크 관리자, 그리고 장치 관리자의 다섯 가지 컴포넌트가 있다.

위의 RTOS 구조에 기반하여, 본 연구에서는 각 RTOS 컴포넌트를 위한 추상화된 RTOS API 의 집합을 정의하고 있다. 이 과정에서 POSIX API 집합에서 OS 와 무관한 API 들을 제거하고, 유사한 API 들을 하나의 API 로 통합하여 추상화를 통해 API 를 정의하고 있다. 결과로 얻어진 추상화된 RTOS API 전체 목록은 표 1 에서 볼 수 있다.



(그림 3) PSM 에 통합된 RTOS 행위

추상화된 RTOS API 를 기반으로 하여 RTOS 행위는 PSM 요소들과 RTOS 컴포넌트들간의 관계를 지정해 주는 것만으로도 쉽게 PSM 에 통합될 수 있다. 그림 3 은 실험 연구에서도 사용된 H.263 디코더의 예를 보여준다. H.263 디코더는 크게 main controller, AVI Reader, Divx decoder, 그리고 MPEG audio decoder 의 네 가지 요소로 이루어져 있다. Main controller 는 다른 세 요소들을 실행하고 조율하고, 몇 가지 RTOS 서비스를 요구하며 AVI Reader, Divx decoder, 그리고 MPEG audio decoder 를 실행할 각각의 쓰레드를 생성한다. 이를 위해 여기에서는 THREAD_CREATE, THREAD_JOIN, 그

리고 THREAD_EXIT 까지 세 개의 추상화된 RTOS API 를 사용한다. 이와 유사하게, 다른 H.263 컴포넌트도 추상화된 RTOS API 을 사용한다 (그림 3).

3. TransPI 의 설계 및 구현

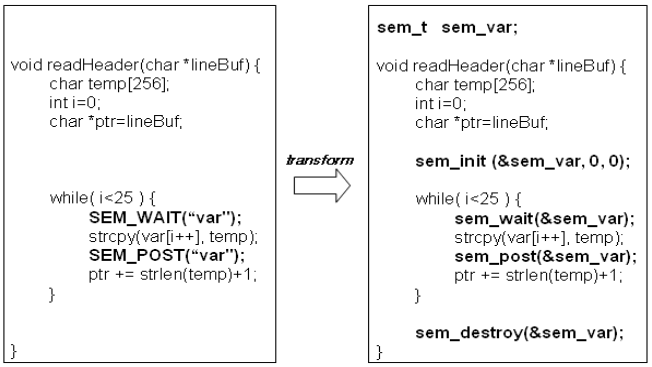
TransPI 개발의 중요한 목표는 POSIX 호환 RTOS 뿐만 아니라 대부분의 현존하는 RTOS 를 포괄하는 것이다. 이를 위해, 본 연구에서는 구성가능한 규칙기반 도구를 설계하고 구현하고 있다. TransPI 는 추상화된 RTOS API 를 변환하기 위한 규칙 집합을 필요로 한다. 각 규칙은 사용하기 쉬운 변환 언어를 사용할 수 있으며, 이를 통해 개발자가 비 POSIX API 를 지원하도록 규칙을 수정할 수 있다.

<표 1> 추상화된 RTOS API 의 전체 목록 (77 개)

RTOS Component	Generic RTOS APIs
Task Manager	EXEC, EXIT, FORK, GETPID, KILL, MQ_GETATTR, MQ_NOTIFY, MQ_RECEIVE, MQ_SEND, MQ_SETATTR, NICE, SCHEDGETPARAM, SCHEDSETPARAM, SEM_GETVAL, SEM_POST, SEM_WAIT, SHM_OPEN, SHM_UNLINK, SIGNAL, SLEEP, STAT, THREAD_BARR_WAIT, THREAD_COND_SIGNAL, THREAD_EXIT, THREAD_COND_WAIT, THREAD_CREATE, THREAD_DETACH, THREAD_JOIN, THREAD_KILL, THREAD_MUTEX_LOCK, THREAD_MUTEX_UNLOCK, THREAD_SPIN_LOCK, WAITPID, YIELD, THREAD_SPIN_UNLOCK, USLEEP,
Memory Manager	MEMALLOC, MMAP, MUNMAP, FREE
File Manager	CHDIR, CLOSE, EOF, FILENO, MKDIR, OPEN, PRINT, READ, REMOVE, RENAME, RMDIR, SCAN, SEEK, SETBUF, TELL, UNLINK, WRITE
Network Manager	SOCK_ACCEPT, SOCK_BIND, SOCK_SETOPT, SOCK_CONNECT, SOCK_GETOPT, SOCKET, SOCK_LISTEN, SOCK_RECV, SOCK_SEND,
Device Manager	IOCTL, POLL, SELECT
Others	CTIME, GETITIMER, SETITIMER, RAND, TIME, GETTIMEOFDAY, MKTIME, PERROR,

3.1 API 패턴

PSM 내에서 각각의 추상화된 RTOS API 를 찾고 특정 RTOS API 로 일대일 매핑할 수 있다. 그러나 이 방법은 몇몇 추상화된 RTOS API 가 반드시 쌍으로 사용되어야만 하므로 문제가 발생할 수 있다. 예를 들어, 락을 얻으려는 SEM_WAIT()는 락을 해제하는 SEM_POST()가 뒤따라야 한다. 또한, 이러한 API 들이 특정 RTOS 코드로 변환될 때에는, sem_open()과 sem_close() 같은 몇 개의 별도 루틴들이 최종 코드에 삽입되어야 한다. 본 연구에서는 하나 이상의 API 가 미리 정의된 의미를 충족하기 위해 코드 내에서 함께 사용되어야 할 때 이를 API 패턴으로 정의하고 있다. 각 API 패턴은 변환의 기본 단위가 되며 변환을 통해 추가적인 코드를 올바른 위치에 삽입함으로써 최종 코드를 생성하게 된다.



(그림 4) SEMAPHORE 패턴의 POSIX 호환 C 코드로의 변환

본 연구에서는 추상화된 RTOS API 로부터 42 개의 API 패턴을 추출하고 있다. 그림 4 는 SEM_WAIT 와 SEM_POST 로 구성된 SEMAPHORE 패턴의 POSIX 호환 C 코드로의 변환 예이다. 우선 SEM_WAIT()와 SEM_POST()를 sem_wait()와 sem_post()로 교체하고, 세마포어 변수 선언, sem_init(), 그리고 sem_destroy() 등의 추가적인 코드 삽입이 이루어진다. 추가적인 코드는 최종 코드에서 한번만 추가되어야 하며, 따라서 패턴 변환 단위로 이루어져야 한다.

```
RULE initialize_SEMAPHORE                                   (1)
    INCLUDE AT ["task1_sub1.c", HEADER]                     (2)
        "#include <semaphore.h>"                            (3)
    DECLARE AT ["task1_sub1.c", LOCAL(task1_sub1_go)]       (4)
        "sem_t @VAR0;",                                     (5)
END                                                         (6)
-------------------------------------------------------------
RULE open_SEMAPHORE                                         (7)
    INSERT AT ["task1_sub1.c", LOCAL(task1_sub1_go)]        (8)
        "sem_init(@VAR0, @VAR1, @VAR2);",                   (9)
END                                                         (10)
-------------------------------------------------------------
RULE transform_SEMAPHORE                                    (11)
    REPLACE "SEM_WAIT(arg1);" WITH                          (12)
        "sem_wait(@VAR0);"                                  (13)
    REPLACE "SEM_POST(arg1);" WITH                          (14)
        "sem_post(@VAR0);"                                  (15)
END                                                         (16)
-------------------------------------------------------------
RULE close_SEMAPHORE                                        (17)
    INSERT AT ["file_name", LOCAL(function_name)]           (18)
        "sem_close(@VAR0);"                                 (19)
END                                                         (20)
```

(그림 5) SEMAPHORE 패턴의 변환 규칙

3.2 변환 언어 및 규칙

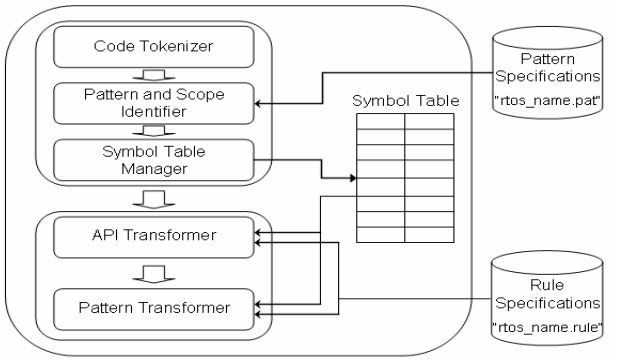
앞에서 언급했듯이, TransPI 의 목적은 모든 RTOS 를 타겟으로 코드생성을 지원하는 것이다. TransPI 는 변환 규칙을 지정할 수 있는 자신만의 변환 언어를 제공한다. 변환 언어에는 핵심 요소로 INCLUDE AT, DECLARE AT, INITIALIZE AT, INSERT AT, REPLACE WITH, FINALIZE AT, DELETE 의 7 개 지시자가 있다.

각 패턴은 initialize, open, transform, close, 그리고 finalize 의 다섯 가지 변환 규칙 중 하나 이상의 규칙을 가지며, 각 규칙은 위에서 설명한 지시자를 사용하여 작성한다. 그림 5 는 SEMAPHORE 패턴에 대한 규칙을 보여준다.

3.3 TransPI 구현

TransPI 의 변환 과정은 다음의 세 단계로 구성된다. 1 단계는 도구 초기화 단계로 추상화된 API 패턴 목록과 변환 규칙 집합의 두 설정 파일을 읽어 들인다.

2 단계는 모델 분석 단계로 PSM 을 입력으로 받아 API 패턴을 찾아낸다. 마지막 3 단계는 코드 생성 단계로 특정 RTOS 용 C 코드를 생성하기 위해 변환규칙을 기반으로 두 차례의 변환을 수행한다. 처음에는 개별 추상화된 RTOS API 를 타겟 RTOS 용 API 로 변환하고, 다음에는 API 패턴 별로 초기화 및 종료화 코드를 삽입한다(그림 6).



(그림 6) TransPI 의 구현 구조

4. 실험 검증

본 연구에서는 TransPI 를 검증하기 위해 두 가지 실험을 수행하여 실제 변환과정의 수행 여부를 보여주고 있다. 첫 번째 실험은, Open POSIX Test Suite 를 이용하여 TransPI 를 집중적으로 테스트하는 것이다. Open POSIX Test Suite 는 웹 상에서 자유롭게 구해서 사용할 수 있는 도구로 POSIX API 로 작성된 수많은 C 코드를 제공한다. TransPI 가 올바른 POSIX 호환 RTOS 용 코드를 생성해 내는가를 확인하기 위해, 원본 C 코드 내의 POSIX API 들을 일일이 수작업을 통해 추상화된 POSIX API 로 변환하고 이를 다시 POSIX 호환 C 코드로 변환하여 원본과 비교한다. 이 테스트를 통하여 많은 버그들을 수정하고 TransPI 를 안정화하는 것이 가능해진다.

두 번째 실험은, 리눅스 상에서 수행되는 H.263 디코더 개발에 TransPI 를 적용하는 것이다. 여기에서는 H.263 디코더를 개발하는 전체 과정은 설명하지 않는다. 먼저 준비 단계로, AviReaderI0.pjm, H263FRDivxI3.pjm, MADStreamI5.pjm 의 네 개 파일로 구성된 PJM 을 작성한다. 그리고 TransPI 를 사용하여 각 PJM 파일을 POSIX 호환 C 코드로 변환한다. 마지막으로 결과 파일들을 컴파일하여 생성된 H.263 디코더를 실행하여 수행 여부를 확인한다. 표 2 와 3 에는 PJM 파일에서 사용된 추상화된 RTOS API 와 C 코드 파일에서 사용된 POSIX API 를 각각 싣는다.

TransPI 는 기존의 코드를 다른 타겟 OS 로 이식하기 위한 도구로 단독으로 사용될 수 있다. 이를 위해 주어진 기존 코드에서 특정 OS 용 API 를 추상화된 RTOS API 로 변환하기 위한 역변환용 규칙을 일부 정할 수 있으며, 이를 통해 기존 코드로부터 RTOS 독립적인 모델을 끌어내어 다시 다른 RTOS 용 최종 코드로 변환할 수 있다. 이 역변환 기능을 활용하면 개발자가 추출한 모델을 수정하여 코드를 재생성함으로써 기존 코드의 수정 및 관리를 용이하게 할 수 있다.

Table 2. PJM 에 사용되는 추상화된 RTOS API

PJM files	Generic RTOS APIs
Proc0.pjm	THREAD_CREATE, THREAD_JOIN
AviReaderI0.pjm	SEEK, READ, MEMALLOC, FREE, GETTIMEOFDAY, OPEN, CLOSE, THREAD_EXIT
H263FRDivxI3.pjm	MALLOC, THREAD_EXIT
MADStreamI5.pjm	OPEN, CLOSE, WRITE, PERROR, FREE, THREAD_EXIT

Table 3. 최종 코드에 사용된 POSIX API

Generated C files	RTOS specific APIs
proc0.c	pthread_create, pthread_join
AviReaderI0.c	lseek, read, malloc, free, gettimeofday, open, close, pthread_exit
H263FRDivxI3.c	malloc, pthread_exit
MADStreamI5.c	open, close, write, perror, free, pthread_exit

5. 결론

본 논문에서는 RTOS 기반 임베디드 소프트웨어 개발을 위한 모델기반 개발방법과 이를 지원하는 도구를 제시하고 있다. 본 논문의 결과물은 (1) PSM 단계에서 RTOS 행위들을 표현하기 위해 사용할 수 있도록 정의된 추상화된 RTOS API 와 (2) PSM 으로부터 RTOS 용 코드를 생성할 수 있는 모델-코드 변환 도구이다.

참고문헌

[1] Thomas O. Meservy and Kurt D. Fenstermacher, “Transforming Software Development: An MDA Road Map,” IEEE Computer, 38(9): 52?58, September 2005.

[2] Pieter Van Gorp, Dirk Janssens, and Tracy Gardner, “Write Once, Deploy N: a Performance Oriented MDA Case Study,” Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference, pp. 123?134, 2004.

[3] 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, pp. 186?197, 2003.

[4] 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.

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

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

[7] Anneke Kleppe, Jos Warmer and Wim Bast, "MDA Explained: The Model Driven Architecture: Practice and Promise," Addison Wesley, 2003.

[8] Grady Booch, James Rumbaugh, and Ivar Jacobson. "The Unified Modeling Language User Guide," Addison Wesley, 1999