제22회 한국정보처리학회 추계학술발표대회 논문집 제11권 제2호 (2004. 11)

실시간 운영체제의 정형적 모델기반 개발 방법론*

김진현, 이수영, 심재환, 양진석, 최진영
고려대학교 컴퓨터학과
e-mail : {jhkim, sylee, jhsim, jsyang, choi}@formal.korea.ac.kr

Development of Real-Time Operating System based on Formal Model

Jin Hyun Kim, Su-Young Lee, Jae-Hwan Sim, Jin Seok Yang and Jin Young Choi
Department of Computer Science, Korea University

요 약

실시간 운영체제는 임베디드 환경에서 매우 중요한 소프트웨어 컴포넌트이다. 임베디드 실시간 운영체제는 고안전성 시스템이나 유비쿼터스 시스템 내에서 강하고(Robust), 신뢰성(Reliable)있고, 안전한(Safety) 특성을 지녀야 한다. 근래에는 이러한 실시간 시스템을 UML 과 같은 모델 기반의 방법으로 설계하고 코드 생성을 통해 구현까지 이끄는 모델 유도 공학(Model Driven Engineering: MDE)[1]을 통해서 개발하고 있다. 이러한 MDE 의 모델은 대부분 UML 기반의 언어를 사용하여 아키텍처를 구성하고 설계하여 코드를 생성하여 구현 코드를 생성해 낸다. 본 논문에서는 이러한 MDE 의 비정형적 언어를 대신하여, 정형명세 언어를 사용하여 실시간 시스템의 모델을 설계하고 이를 검증하여 실시간 운영체제를 개발하는 방법론을 기술한다. 이렇게 함으로써 보다 엄격한 언어를 통해 시스템의 설계의 모호함이나 모순을 없애고, 수학적 기반의 검증을 통해 보다 안전하고 신뢰성 있는 시스템을 구현하자 한다.

1. 서론

임베디드 시스템은 개인용 컴퓨터가 널리 보급 된 이래 컴퓨터 공학의 새로운 개념으로 자리잡고 있다. 항공 제어 시스템, 원자력 발전 제어 시스템과 같은 제어 시스템으로부터 시작하여 가전제품에 이르기까지 산업 전반에 걸쳐 아날로그 시스템의 단순한 기능을 임베디드 시스템을 통해 다양화 되었다. 또한 원자력 발전소와 같이 수 십 년 이상의 생명 주기를 갖는 시스템은 부품의 고갈로 말미암아 생명 주기를 단축시켜야만 했으나 임베디드 시스템의 출현으로 말미암아 기능의 확장 및 유지 보수도 가능하게 되었다. 근래 들어 유비쿼터스라는 말이 보급되면서 임베디드 환경의 중요성은 날로 증가되고 있다.

이러한 임베디드 시스템은 하드웨어와 상호 작용하는 반응형 시스템(Reactive systems)으로, 실시간적 특징을 지니고 있으며, 대부분 실시간 운영체제를 지니고 여러 기능과 작업을 나누어 담당하게 된다.

이러한 임베디드 시스템의 소프트웨어의 특징은 소프트웨어의 생명 주기가 거의 시스템의 생명 주기와 같을 정도로 길다. 기존의 개인용 컴퓨터는 사용자의 편의에 따라 다양한 애플리케이션을 설치하고 지웠지만, 이러한 임베디드 시스템은 대부분 한번 설치하면 기능이 바뀌거나 버그가 발견되기 전에는 바꾸지 않게 된다. 따라서 이러한 시스템은 대부분 신뢰성(Reliable)있고. 안전(Safety)하고 어떠한 환경 하에서도 강해(Robust)하는 특징을 지고 있다.

이러한 특성은 임베디드 소프트웨어의 요구 단계부터 설계 구현에 이르기까지 엄격한 설계 과정과 검증 과정을 거칠 것이 요구된다.

근래 들어 이러한 임베디드 소프트웨어를 개발하기 위해 모델 유도 공학(Model Driven Engineering)이 연구

* 본 논문은 KNICS(Korea Nuclear Instrumentation & Control Systems) 사업단의 지원하에 이루어짐.

되고 있다.

모델 유도 공학은 시스템의 요구 분석부터 시작하여 설계 구현에 이르기까지 모델에 기반하여 시스템을 분석하고 설계하며, 가상 실행하여 시스템을 확인한 다음, 궁극적으로 코드를 생성하여 시스템을 개발하는 방법론을 가리킨다. 이러한 모델 유도 공학은 주로 UML2.0 의 다양한 언어를 통해 구현되고 제안되고 있는데, 특히 UML 은 시스템의 다양한 특성을 다양한 각도에서 제시할 수 있기 때문에 유용하다.

하지만 UML 은 그 의미론이 수학적으로 기술되지는 않고 있다. 즉 정형적 의미론을 가진 언어를 통해 시스템을 설계하고 있지는 않다는 것이다.

이러한 언어에서는 설계 언어를 보다 자유롭게 사용하려는 UML 개발자들의 의도를 살펴 볼 수 있다. 즉 수학적인 의미론을 묶어 놓는다면, 설계는 명료해지겠지만 설계하는데 여러 제약사항으로 인해 설계의 어려움을 간과할 수 없기 때문에, 단지 문법만을 제시해 놓고 나름대로의 의미론으로 정의하여 사용하게 하고 있다.

본 논문에서는 MDE 의 개발 방법론을 기반으로 실시간 시스템을 설계하되 보다 엄격한 정형 의미론을 지니고 시스템을 설계하고, 정형적 모델을 수학적 검증 시스템에 놓고 검증함으로 단지 설계의 장점에서의 시스템의 개발의 편리함을 떠나, 설계의 검증 차원에서 설계를 이용함으로 신뢰성과 안전성을 보장하는 차원에서의 실시간 시스템 및 실시간 운영체제를 개발하는 방법론을 제안하고자 한다.

본 논문은 다음과 같이 구성되어 있다. 2 장에서는 임베디드 시스템을 위한 실시간 시스템의 특징을 고려할 것이며, 3 장에서는 본 논문에서 제시하는 정형적 모델 기반의 임베디드 실시간 시스템의 개발 방법론을 기술한다. 4 장에서는 이러한 방법론에 근거한 모델과 확인 검증 결과를 기술한다. 5 장에서는 본 논문의 결론을 기술한다.

2. 임베디드 시스템을 위한 실시간 운영체제

실시간 운영체제는 임베디드 시스템의 다양한 기능을 쉽게 관리하게 해주는 자원 관리자 역할을 하는 핵심 소프트웨어이다. 임베디드 시스템의 실시간 운영체제는 다음과 같은 특징을 지닌다. [2]

임베디드 시스템을 위한 실시간 운영체제는 기본적으로 외부 환경에서 들어오는 신호에 대해 시간적 제약을 가지고 처리를 반응을 보여야 하는 반응형 시스템의 특징을 지니고 있다. 즉 특정 자극(Stimuli)에 대해 시간적 제약을 가지고 반응(reaction)을 보여야 하는 것이다. 운영체제는 이렇게 적절한 시간 내에 자극을 처리할 수 있는 태스크를 운용하여 반응을 출력하도록 태스크를 스케줄링하게 된다. 만약 스케줄링이 어긋나거나 시간적 제약 내에 태스크를 완료하지 못하면 시스템 전체에 큰 영향을 초래하게 된다. 중요한 것은 실제 일을 처리하는 것은 실시간 운영체제가 운용하게 될 태스크이며 이러한 태스크의 집합이 모여 하나의 작업(Job)을 이루어 마치게 된다. 여기에 관련된 것이 운영체제가 관리하는 자원이 된다. 자원이라함은 시스템의 논리적인 자원과 물리적인 자원으로 나눌 수 있는데, 논리적인 자원이라 함은 특정 작업을 수행하는 유닛이라 할 수 있는 태스크(task) 혹은 프로세스, 쓰레드, 메모리를 추상화 시킨 세그먼트와 페이지, 태스크의 동기화와 통신을 위한 세마포어, 메시지 큐, 메일 박스 등이 있다. 물리적인 자원은 실제 작업을 수행하는데 사용하는 자원으로 CPU, 메모리, 디스크, 레지스터, 버스 등이 있으며, 이들 대부분은 운영체제에 의해 관리되고 보호된다. 따라서 실시간 운영체제는 시간적 제약 내에, 특정 입력에 대해 다양한 자원을 이용하여 반응을 나타내도록 자원을 관리해주는 소프트웨어라 할 수 있다.

따라서 시스템의 안전성과 신뢰성, 그리고 견고성을 만족시키는 임베디드 시스템은 실시간 운영체제가 얼마나 자원을 효과적으로 관리하느냐에 달려 있다고 해도 과언이 아니다.

하지만 중요한 사실로, 임베디드 실시간 운영체제는 일반 운영체제와 달리 단순한 기능만을 제공하도록 설계된다는 것이다.

따라서 일반적인 임베디드 시스템을 위한 실시간 운영체제는 다음과 같은 기능을 지닌다.

- ❑ 스케줄링
  - ♦ 컨텍스트 스위치
  - ♦ 스케줄링
- ❑ 태스크간의 동기화 및 통신기능
  - ♦ 세마포어
  - ♦ 메시지 메일박스
  - ♦ 메시지 큐
- ❑ 인터럽트 핸들러
  - ♦ 타임 틱 핸들러
  - ♦ 사용자 정의 인터럽트 핸들러
- ❑ 디바이스 드라이버

위와 같은 기능도 필요 없는 것은 컴파일 시에 컴파일하지 않고 제외시킴으로 실시간 운영체제 자체를 간소화 시킨다. 따라서 기존 범용 운영체제에서 제공하는 다양한 기능들-이를 테면 데드락 방지 기능, 우선순위전도 방지를 위한 기능 등-은 기대할 수 없다.

또한 대부분 태스크들은 컴파일 시에 그 수와 특성이 정해진다. 그 즉시 정해 지지 않는다 하더라도 일반 범용 운영체제가 운용하는 태스크 보다는 훨씬 단순하고 결정적이다.

따라서 이러한 문제를 피해가기 위해서는 태스크의 설계 자체가 큰 문제로 떠오른다. 즉 데드락을 방지하기 위해서 공유 자원을 공유하는 태스크들은 자원에 대한 우선 순위와 사용 순서를 결정하여 데드락이 처음부터 발생하지 않도록 설계해야 하며, 또한 스케줄링 역시 스케줄링 분석을 통해 실시간 시스템의 스케줄링 가능성을 만족시켜야 한다.

즉 시스템의 결정성은 실제 실시간 운영체제에 달려 있다기 보다는 실시간 운영체제의 최소 기능을 이용해 태스크가 얼마나 조화롭게 운용되는가에 달려 있다 할 수 있다.

따라서 이러한 임베디드의 실시간 운영체제의 확인 검증은 실시간 운영체제의 자체의 확인 검증 외에도 태스크를 운용하는 임베디드 시스템의 설계에 대한 검증이 선행 되어야 한다.

이러한 연구를 위해 모델 유도 공학에서는 다음과 같은 모델을 제시한다.

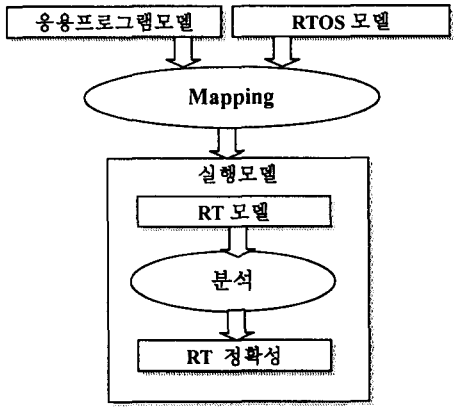

그림 1 MDA 를 이용한 실시간 시스템 개발

그림 1 에서 볼 수 있는 것처럼 모델 유도 공학의 MDA(Model Driven Architecture)를 이용해서, 임베디드 시스템을 설계하게 되는데, 응용프로그램과 실시간 운영체제를 함께 모델링하고 이를 합쳐 실행 모델을 만들어 분석 한 후, 실시간적 특성을 분석하여 시스템을 개발하게 된다. 본 논문에서는 이러한 모델을 정형적 모델로 대체함으로 보다 명료한 설계에서 정형적 분석을 유도하여 보다 완전한 시스템을 구현하고 자 한다.

3. 정형 모델을 기반한 실시간 시스템의 개발 방법론

본 논문에서 제안하는 정형적 모델을 기반한 실시간 시스템의 개발 방법론은 다음과 같다.

그림 2 에서는 본 논문에서 제시하는 정형적 모델을 기반한 모델 기반의 개발 방법론을 제시한다. 임베디드 시스템의 요구 사항이 주어지면 요구 사항을 유즈 케이스 다이아그램(Use-Case Diagram)과 메시지 시퀀스 챠트(Message-Sequence Charts : MSC)을 이용하여 요구 분석을 하여, 요구 분석에 대한 요구 모델을 만들게 된다. 이 때, 두 가지 모델이 생성되게 된다. 즉 RTOS 모델과 태스크(애플리케이션) 모델이 생성되게 된다. 이 태스크 모델의 특징은 RTOS 의 요구 사항을 고려하여 태스크가 안전성이나 정확성을 고려하여 각각의 태스크나 태스크 간의 수행 모델을 시나리오 별로 혹은 유즈케이스 별로 모델링 한 것이다. 이러한 분석을 통해 자연어의 요구 사항의 모순을 찾거나, 실시간 운영체제의 제약된 기능을 고려한 태스크의 수행 모델이 만들어지게 된다.

그 다음 단계로, MSC 모델을 모두 수용할 수 있는 실행 모델이 만들어지게 된다. Statecharts[3]는 전형적인 실행 중심 모델 언어로서 정형적인 의미를 지닌다. 본 논문의 중심인 정형적 모델은 바로 이 Statecharts 를 가리키는 것이다. 이 Statecharts 는 MSC 가 요구하는 모든 시나리오만을 수용하는 행위적인 설계를 하게 된고 이를 통해 실제 실행 모델을 시뮬레이션 해 보게 된다. 그에 더하여, 이러한 실행 모델은 모델 체킹[4]을 통해 검증하게 된다. 검증 특성으로는 실시간 시스템의 핵심인 스케줄링 가능성과 데드락을 실험하게 된다.

Activity charts 는 프로그램의 구조(Architecture)를 표현하기 위해 그려진다. 다시 말해 Activity charts 는 임베디드 소프트웨어의 컴포넌트 즉 인터페이스와 내부적인 행위를 표현하기 위해 만들어지는 것을 각각의 입출력 및 의존 관계를 기술하게 된다.

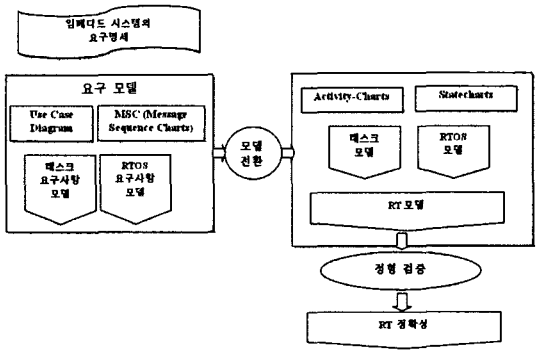

그림 2 RTOS 의 정형적 모델 기반의 RTOS 개발론

4. 정형모델에 기반한 실시간 운영체제 개발

본 절에서는 3 절에서 언급한 방법론 대로 설계 되고 검증된 결과를 기술한다.

4.1 실시간 운영체제의 요구사항

실시간 운영체제는 다음과 같은 기능을 갖는다.

- **스케줄링 :** 스케줄링 알고리즘은 **우선 순위 기반**의 스케줄링 알고리즘을 갖는다. 스케줄링 시점은 스케줄링 요청이 들어온 시점이며, 스케줄링 수행 도중에는 재 스케줄링 요청을 받아들이지 않는다. 스케줄링시 태스크 간의 우선순위전도를 방지하기 위한 알고리즘은 존재하지 않는다.

- **태스크 간의 통신 :** 태스크 간의 통신은 세마포어와 메시지 큐를 지닌다. **세마포어나 메시지 큐는 태스크가 사용할 수 없을 때에는 모두 태스크를 블록 상태로 만든다.** 태스크 간의 자원 공유에 대한 데드락을 막기 위한 메커니즘은 존재하지 않는다.

- **인터럽트 핸들러**: 외부로부터 들어오는 디바이스 신호를 입력으로 하는 부 정기적인(sporadic) 인터럽트와 정기적으로 스케줄링을 일으키는 **타임 틱(Time Tick)**을 처리하는 핸들러가 존재한다.

- **태스크 관리자** : 태스크를 생성하고 삭제하는 관리자이다. 태스크를 생성한 후, 실시간 운영체제가 멀티태스킹을 실행 중이라면 스케줄링을 요청하고, 멀티태스킹을 실행 중이 아니라면 스케줄링을 요청하지 않는다. 태스크 삭제 시에도 같은 스케줄링 요청이 존재한다.

- **컨텍스트 스위치**: 실행 중(running)상태의 태스크를 중지시키고, 그 태스크의 컨텍스트를 해당 태스크의 스택에 저장하고, 스케줄링 된 태스크를 실행하기 위해 태스크 컨텍스트를 레지스터에 저장한다.

4.2 실시간 운영체제의 정형적 모델

본 절에서는 실시간 운영체제의 정형적 모델을 기술한다.

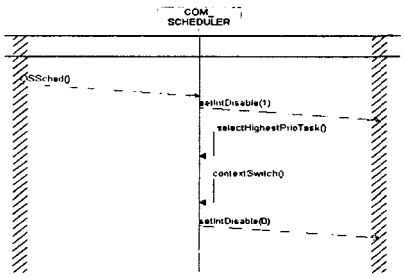

그림 3 스케줄러의 MSC

그림 3 에서는 스케줄러의 요구 분석된 MSC 모델이다. 이를 구현 RTOS 의 실행 모델이 그림 4 이다. 이 그림에서는 태스크 4 개를 스케줄링 하는 스케줄러의 정형적 모델을 볼 수 있다.

본 연구에서는 이 외에도 태스크의 모델을 설계하였으며 함께 시뮬레이션과 모델 체킹을 수행하였다. 이러한 확인 검증을 통해 모델에서 간과할 수 있는 인터럽트와 관련된 오류를 검증할 수 있었으며, 요구명세와 설계 명세를 수정하는데 도움을 주었다.

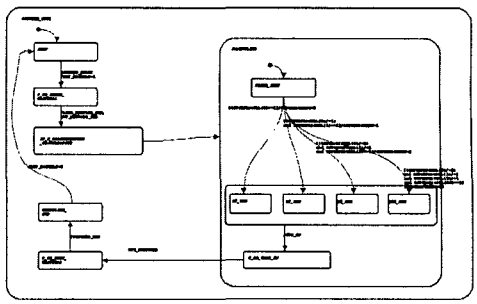

그림 4 스케줄러의 정형적 모델

5. 결론

본 논문에서는 신뢰성와 안정성이 요구되는 실시간 운영체제를 개발하기 위해 정형적 모델 기반으로 시스템을 모델링하고 확인 검증하고 개발하는 방법론을 제시한다. 기존의 MDE 가 추구하는 설계 방법론을 충분히 이용하면서, 정형 검증이라는 장점을 살릴 수 있는 것이 본 방법론의 특징이라 할 수 있다.

향후 연구 과제로서는 이러한 개발 방법론으로 실제 시스템을 개발하여 운용하는 것과 모델간의 변환을 보다 정형적으로 할 수 있는 관계를 정의하는 것이 필요하다고 생각된다.

참고문헌

[1] Bruce Powel Douglass, Real-Time UML 3rd Edition, Addison Wesley, 2004.

[2] Jin Cooling, "Software Engineering for Real-Time Systems", Addison Wesley, 2003-09-08

[3] D.Harel "Statecharts: A Visual Formalism for Complex Systems", Science of Computer Programming, Vol. 8, pp. 231-274, 1987.

[4] Edmund M. Clake, Jr, Orna Grumberg, Doron A. Peled, " Model Checking" , MIT Proess 1999.