제21회 한국정보처리학회 춘계학술발표대회 논문집 제11권 제1호 (2004. 5)

내장형 미들웨어 환경에서 동적 재구성이 가능한 실시간 스케쥴러의 설계

서영준, 송영재
경희대학교 컴퓨터공학과
e-mail: yjseo@khu.ac.kr

A Design of Dynamic Reconfigurable Real-Time Scheduler in Embedded Middleware Environments

Young-Jun Seo, Young-Jae Song
Dept of Computer Engineering, Kyung-Hee University

요 약

최근 디지털 가전과 모바일 컴퓨팅이 화두로 떠오르면서 내장형 기술이 전성기를 맞이하고 있으며, 다양한 내장형 시스템들간의 상호 운용성, 플랫폼 독립성과 이식성을 지원할 수 있는 환경을 제공하는 내장형 미들웨어의 중요성이 부각되고 있다. 미들웨어는 신뢰성, 안전성, 보안성, 실시간성 등과 같은 기능에 직접 포함되지 않는 비기능적 요구를 응용 분야에 따라 요구할 수 있으며, 그 때마다 다양한 비기능적 요구에 맞는 미들웨어를 각기 따로 개발하는 것은 지극히 어려운 일이다. 따라서, 비기능적 요구에 적합하게 미들웨어를 동적으로 재구성하는 컴포넌트 개발 방법이 주목 받고 있으며, 이를 통해 재구성된 내장형 미들웨어에서는 비기능적 요구인 실시간성이 요구되므로, deadline 만족 여부를 확인하는 스케쥴링 분석이 중요하다. 많은 최적의 스케쥴링 분석 알고리즘이 존재하나 그들 중 어느 것도 동적 재구성이 될 때 태스크 집합상의 가정(assumption)이 변경되는 내장형 미들웨어를 만족 시킬 수 없다. 가정에 적합한 스케쥴링 알고리즘으로 대체하기 위한 제안들은 대부분 정적 조립 환경에서 새로운 버전에 의해 기존 클래스를 교체하는 코드 수정에 기반하나, 동적 재구성을 통해 행위를 변경하는 내장형 미들웨어에서는 응용하기 어렵다. 따라서, 본 논문에서는 동적 재구성 환경에서 수행되는 내장형 미들웨어를 위해, 적합한 스케쥴링 알고리즘으로 대체 할 수 있는 스케쥴러를 런타임 컴포넌트 개조 기술 중 하나인 Type-safe delegation이 적용된 strategy 패턴을 기반으로 설계하였으며, 이를 통해 내장형 미들웨어 스케쥴러의 유연성과 확장성을 증가하는 방안을 제시하였다.

1. 서론

[1]최근 디지털 가전과 모바일 컴퓨팅이 화두로 떠오르면서 내장형(embedded) 기술이 전성기를 맞이하고 있다. 내장형 기술은 모바일 시스템에서부터 가전제품, 산업용 기기까지 그 적용분야가 다양하며, 그에 따른 운영체제 또한 셀 수 없이 많다. 따라서, 분산되고 편재되어 있는 환경에서 다양한 내장형 시스템들간의 상호 운용성, 플랫폼 독립성과 이식성을 지원할 수 있는 환경을 제공하는 내장형 미들웨어의 중요성이 부각되고 있다[1].

미들웨어는 신뢰성, 안전성, 보안성, 실시간성 등과 같은 기능에 직접 포함되지 않는 비기능적 요구를 응용 분야에 따라 요구할 수 있으며, 그 때마다 다양한 비기능적 요구에 맞는 미들웨어를 각기 따로 개발하는 것은 지극히 어려운 일이다. 이를 위해서는 미들웨어의 핵심부품들은 빠르게 구축되어야 하고 제품의 품질 개선이 용이해야 하며 부품의 재사용이 가능하여 개발비용을 절감해야 되고, 소프트웨어의 증가하는 복잡도를 감소시킬 수 있는 소프트웨어 개발 기술이 요구되고 있다. 이러한 변화와 아울러 재구성 집합이 실행 시간에 알려지며, 기대하지 않은 변화를 다루는 동적 재구성과 같은 컴포넌트기반 개발 방법이 주목받고 있다[2].

내장형 시스템에 적합하게 재구성된 내장형 미들웨어에서는 비기능적 요구인 실시간성이 요구되므로, 모든 태스크가 그들의 deadline을 만족하는지 여부를 확인하는 스케쥴링 분석이 중요하다. 많은 최적의 스케쥴링 분석 알고리즘이 존재하나 그들 중 어느 것도 동적 재구성이 될 때 태스크 집합상의 가정(assumption)이 변경되는 내장형 미들웨어를 만족 시킬 수 없다. 가정에 적합한 스케쥴링 알고리즘으로 대체하기 위한 제안들은 대부분 정적 조립 환경에서 새로운 버전에 의해 기존 클래스를 교체하는 코드 수정에 기반하나[3], 동적

1) 본 연구는 한국과학재단 목적기초연구(과제번호 : R01-2001-000-00357-0) 지원으로 수행되었음.

재구성을 통해 행위를 변경하는 내장형 미들웨어에서는 응용하기 어렵다.

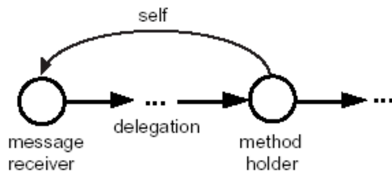
따라서, 본 논문에서는 동적 재구성 환경에서 수행되는 내장형 미들웨어를 위해, 적합한 스케쥴링 알고리즘으로 대체할 수 있는 유연하고 확장할 수 있는 스케쥴러를 제안한다. 제안한 스케쥴러는 런타임 컴포넌트 개조 기술 중 하나인 Type-safe delegation이 적용된 strategy 패턴을 기반으로 설계하였으며, 이를 통해 본 논문에서는 유연성, 연속성이 필요한 환경에서 내장형 미들웨어를 스케쥴링 하는 스케쥴러를 설계하는 방안을 제시하였다.

2. 관련 연구

본 장에서는 런타임에 컴포넌트의 인터페이스와 행위를 변경함으로서 새로운 컴포넌트로 컴포넌트를 개조하는 기법과 대표적인 스케쥴링 알고리즘에 대해서 살펴보도록 하겠다.

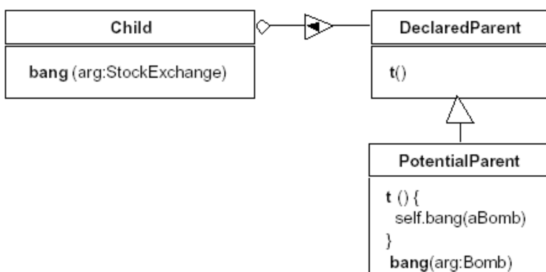
2.1 런타임 컴포넌트 개조

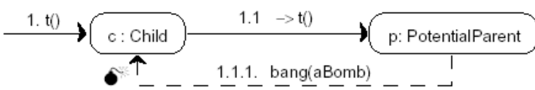
Type-Safe Delegation[4]에서 delegation은 (그림 1)과 같이 child(message receiver)에 매칭하는 메소드를 가지지 않은 메시지는 자동으로 parent(method holder)에 포워딩 되며, 적합한 메소드가 parent에 발견될 때, self 매개변수에 바인딩 된 후에 실행 된다. self 값이 child에 바운드 되므로, child의 수정된 행위를 인식하게 되며, 컴포넌트가 직접적으로 수정되거나 수행중인 시스템으로부터 unload 되지 못할 때, wrapper(child)를 추가함으로서 행위를 변경할 수 있음을 의미한다.



(그림 1) delegation의 정의

(그림 2)는 child와 parent의 bang() 메소드가 서로 다른 인자 타입을 가지면 인스턴스 c로 보내지는 self.bang(aBomb) 메시지가 불안정해지는 Typing 문제를 나타내며, bang()을 포함하는 공통으로 선언된 supertype(DeclaredParent)의 수정을 통해 호환성을 가짐으로서 해결할 수 있다.





(그림 2) Typing 문제가 발생한 delegation

2.2 스케쥴링 알고리즘

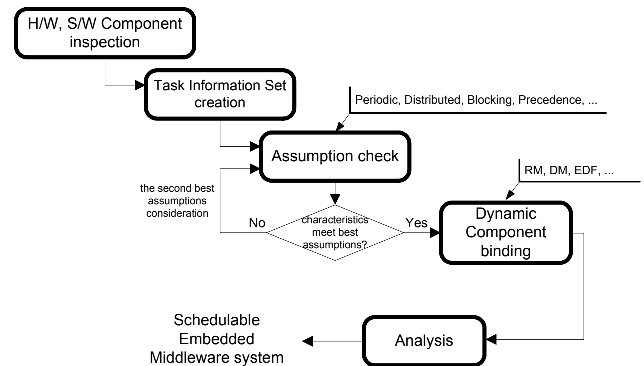
실시간 스케쥴링 기법은 크게 세 가지 기법, 즉 우선순위 유도형, 시간 유도형, 공유 유도형 스케쥴링 기법으로 분류될 수 있다[5]. 우선순위 유도형 스케쥴링 기법은 각 태스크에게 하나의 우선순위를 부여하고, 특정 순간에 가장 높은 우선순위를 가진 태스크를 우선적으로 선택하여 실행한다. 이 기법은 태스크의 우선순위 부여 방법에 따라 정적 우선순위와 동적 우선순위 스케쥴링 알고리즘으로 분류할 수 있다. 가장 잘 알려진 정적 우선순위 스케쥴링 알고리즘은 발생 간격이 가장 짧은 태스크에 가장 높은 우선순위를 할당하는 RM(Rate Monotonic)과 데드라인이 짧은 태스크에 더 높은 우선순위가 주어지는 DM(Deadline Monotonic) 알고리즘이 있다. 또한 대표적인 동적 우선순위 알고리즘은 매 순간 대기하고 있는 태스크들의 데드라인을 비교해서 그 가운데 가장 데드라인에 임박한 태스크에게 가장 높은 우선순위를 배정하는 EDF(Earliest Deadline First) 알고리즘이 있으며, 각 태스크에게 미리 우선 순위를 정하지 않고, 프로세서가 가용한 시점에서 데드라인에 임박한 순서에 따라 경쟁을 통해 프로세서를 확보한다. 그 외에 어떤 태스크를 언제 실행할 것인지를 사전에 오프-라인으로 스케쥴링 하여 이를 테이블 형태로 저장하고, 시스템 실행 시간에는 단지 주어진 테이블에 따라 프로세서를 할당하는 시간 유도형 스케쥴링 기법과 엄격한 실시간 성능 보장을 요하지 않을 때 적합한 공유 유도형 스케쥴링 기법이 있다. 그러나, 본 논문에서는 소수의 태스크 속성만으로 스케쥴 가능성을 결정할 수 있고 구현하기도 쉬운 우선순위 유도형 알고리즘만을 대상으로 한다.

3. 동적 개조가 가능한 내장형 미들웨어 스케쥴러

본 논문에서 제안한 스케쥴러는 동적, 선택적, 프록시 기반의 컴포넌트 개조 방법을 사용하며, 동적 재구성시에 알려지는 가정에 적합한 스케쥴링 알고리즘으로 대체하기 용이하다.

3.1 동작 프로세스

제안한 스케쥴러의 동작 프로세스는 (그림 3)과 같이 H/W, S/W 컴포넌트 검사, 태스크 정보 집합 생성, 가정 검사, 동적 컴포넌트 바인딩, 분석의 여러 단계로 나뉘어 진다.



(그림 3) 동작 프로세스

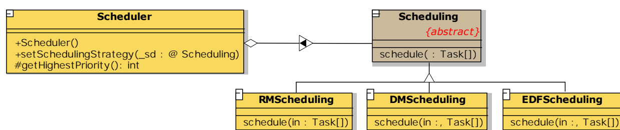
H/W, S/W 컴포넌트 검사 단계에서는 동적 재구성이 이루어진 내장형 미들웨어의 소프트웨어와 하드웨어 컴포넌트들을 조사하며, 태스크 정보 집합 생성 단계에서는 특징들과 태

스크 집합의 타이밍 정보를 생성한다. 가정 리스트에는 태스크가 주기적인지(periodic), 다중 프로세서 상에 분산되어 있는지(distributed), 낮은 우선순위 태스크가 높은 우선순위 태스크를 막을 수 있는지(blocking), 우선순위 제약이 있는지(precedence) 여부에 대한 가정들[3]이 있으며, 가정 검사 단계에서 특징들과 일치하는 가정들이 있는 분석을 선택한다. 예를 들면, 단일 프로세서상에서 독립적인 주기적 태스크를 갖는 시스템을 위해서는 Rate Monotonic 스케쥴링 알고리즘이 시스템에 적용될 것이다. 그러나, 만약 적합한 가정들이 있는 분석을 찾지 못하면 가정 검사 단계에서는 차선책을 선택한다. 동적 컴포넌트 바인딩 단계에서는 선택된 스케쥴링 알고리즘을 스케쥴러에 동적으로 대체한다. 동적으로 대체된 알고리즘을 사용하여 내장형 미들웨어에 적절한 분석과 스케쥴링 정책이 적용되는 것을 가능케 한다.

3.2 Type-safe delegation에 기반한 런타임 컴포넌트 개조

앞서 3.1절에 설명한 동작 프로세스 중 동적 컴포넌트 바인딩 단계에서는 구체적으로 런타임 컴포넌트 개조 기술중 하나인 Type-safe delegation이 적용된 strategy 패턴을 사용하여 적합한 스케쥴링 알고리즘으로 대체한다. strategy 패턴[6]을 이용하면 클라이언트와 독립적인 다양한 스케쥴링 알고리즘으로 변형할 수 있다.

제안한 스케쥴러의 구조는 (그림 4)와 같이 클라이언트로부터의 요청을 Scheduling 클래스에 보내는 Scheduler 클래스와 제공하는 모든 스케쥴링 알고리즘에 대한 공통의 오퍼레이션들을 인터페이스로 정의한 Scheduling 클래스, 그리고 각각 서로 다른 스케쥴링 알고리즘을 구현한 RMScheduling, DMScheduling, EDFScheduling 클래스와 같은 서브클래스들로 구성된다.



(그림 4) 스케쥴러의 구조

Scheduler 클래스는 미들웨어가 내장형 시스템의 비기능적 요구에 맞게 재구성될 때 비실시간 스케쥴링 알고리즘인 FIFOScheduling 대신해서 Scheduling 클래스에 적합한 실시간 스케쥴링 알고리즘으로 대체하는 책임을 위임하며, 객체 sd에 다른 Scheduling 서브타입의 객체를 할당함으로서 간단하게 서로 다른 집합을 동적으로 스위치 할 수 있다.

동일 메소드 schedule(@Task[] in)은 Scheduling 클래스의 모든 서브클래스들에서 구현 되며, Scheduler 클래스에서는 schedule() 메소드의 명시적인 정의는 불필요하며, 암시적으로 Scheduling 클래스에 위임된다.

(리스트 1)은 서로 다른 스케쥴링 분석 기법 사이를 동적으로 스위치 하고 사용할 수 있는 Scheduler 클래스의 일부 소스 코드를 나타내며, delegation 연산을 지원하는 Java의 확장 언어인 LAVA를 사용하여 기술하였다. LAVA에서 객체는 delegation 속성에 의해 참조되는 다른 객체에 위임할 수 있으며, delegation 속성은 인스턴스 변수 선언에 mandatory delegatee 키워드를 추가함으로서 객체의 클래스에 선언되어야 한다. delegation 속성이 선언된 타입에 따르는 값을 참조할 수 있기 때문에 delegation 속성에 대한 할당은 런타임에 객체의 행위를 변경하기 위해 사용될 수 있다. EDF 알고리즘의 경우 동적 우선순위를 가지므로 최우선순위 태스크를 찾기 위해 getHighestPriority() 메소드를 호출 한다. schedule() 메소드 내에서는 EDFScheduling 클래스와 같이 복잡한 알고리즘은 receiver 메시지를 통하여 필요한 추가적인 입력을 얻을 수 있으며, 반면에 단순한 알고리즘은 매개변수로서 전달되는 입력을 사용한다. 만약, delegation을 사용하지 않는다면 매개변수들을 포함하기 위해 메소드 시그너처를 확장해야 한다.

```
public class Scheduler {
    protected mandatory delegatee @Scheduling sd;

    public Scheduler() {
        sd = new FIFOScheduling(); }

    public setSchedulingStrategy(@Scheduling _sd) {
        sd = _sd; }

    protected int getHighestPriority() { ... }

    ...
}

public abstract delegatee class Scheduling {
    abstract public int schedule(@Task[]);
}

public class RMScheduling extends Scheduling {
    public void schedule(@Task[] in) {
        ...
    }
}

public class EDFScheduling extends Scheduling {
    public void schedule(@Task[] in) {
        int taskid = receiver.getHighestPriority();
        holder.scheduleEDF(in, taskid);
    }

    public void scheduleEDF(@Task[] in, int taskid) {
        ...
    }
}
```

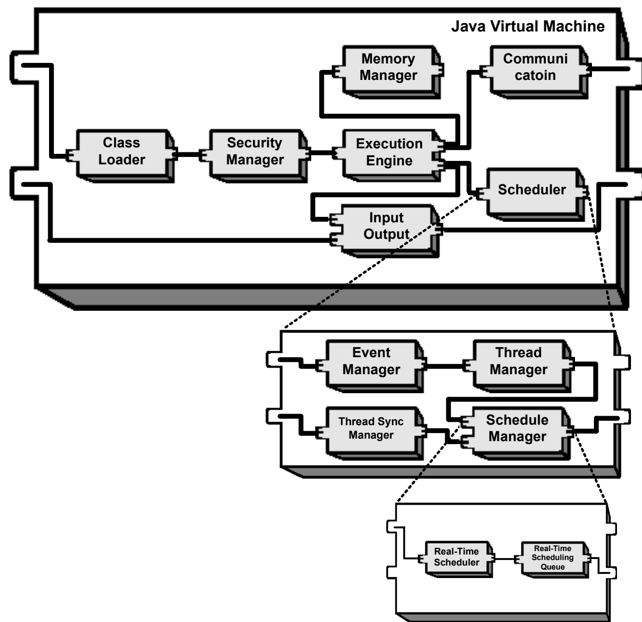
(리스트 1) LAVA로 기술한 스케쥴러의 소스 코드

3.3 실시간 자바 가상머신 스케쥴러

마지막으로 본 논문에서는 연구 과제의 일환으로 개발중인 컴포넌트 지향 실시간 자바 가상머신의 서브 시스템 중 동적 재구성이 가능한 스케쥴러에 대해서 살펴보도록 하겠다.

실시간 자바 가상머신 스케쥴러는 크게 스케쥴 관리자, 쓰레드 관리자, 쓰레드 동기화 관리자, 이벤트 관리자로 나눌 수 있다. 스케쥴 관리자는 생성된 실시간 쓰레드가 스케쥴 될 수 있는지를 결정하고 또한 스케쥴링 될 쓰레드들을 선택해서 수행시키는 기능을 하며, 쓰레드 관리자는 실시간 쓰레드의 생성 및 관리를 담당하며, 동기화 관리자는 실시간 쓰레드간의 공동작업을 위한 동기화 관련 함수 및 라이브러리를 제공 한다. 이벤트 관리자는 실시간 시스템에서 사용되는 센서의 신호 입력과 같은 요구 사항 처리와 자바 가상머신 시스

템 내부의 비동기적인 요구를 처리한다. 실시간 자바 가상머신 코어 중 실시간 스케쥴러와 관련된 컴포넌트들의 아키텍처는 (그림 5)와 같다.



(그림 5) 실시간 스케쥴러 및 관련 컴포넌트 아키텍처

스케쥴 관리자(Schedule Manager) 컴포넌트는 실시간 자바 애플리케이션에서 요구하는 실시간 쓰레드를 스케쥴링 하기 위해서 실시간 스케쥴링에 필요한 함수 및 라이브러리를 포함한 스케쥴 관리자와 가상머신 내부에서 스케쥴링 알고리즘들을 동적으로 재구성하여 실시간 쓰레드를 직접적으로 스케쥴링 할 수 있는 실시간 스케쥴러, 실시간 쓰레드의 수행순서와 수행가능여부를 판단하는 실시간 스케쥴링 큐로 구성된다.

쓰레드 관리자(Thread Manager) 컴포넌트는 자바 가상머신에서 동작하는 모든 쓰레드를 생성하고 관리하기 위해서 쓰레드 관리자와 실시간 쓰레드 생성자를 포함한다. 쓰레드 관리자는 운영체제와 독립적인 비 실시간 쓰레드 및 실시간 쓰레드 관리를 위한 라이브러리와 함수를 포함하는 컴포넌트이다. 실시간 쓰레드 생성자는 실시간 자바 애플리케이션의 실시간 자바 쓰레드를 생성한다. 기반 운영체제에 민감한 쓰레드 생성 및 몇몇 관리에 관한 함수를 포함하는 컴포넌트이다.

쓰레드 동기화 관리자(Thread Sync Manager) 컴포넌트는 실시간 자바 가상머신 코어 개발에 있어서 발생하는 동기화 관련 문제인 우선순위 역전 현상(Priority Inversion)을 막기 위해서 쓰레드의 동기화를 지원하기 위한 함수들에 우선순위 상속(Priority Inheritance)기법을 적용한다. 실시간 자바 가상머신에서 사용하는 세마포어(semaphore)의 경우 실시간 쓰레드가 다른 쓰레드가 소유하고 있는 세마포어를 얻기 위해서 기다릴 때 세마포어를 소유하고 있는 쓰레드 보다 대기하고 있는 쓰레드의 우선순위가 높을 때 우선순위 역전 현상이 나타날 수 있다.

이벤트 관리자(Event Manager) 컴포넌트는 이벤트 생성과 이벤트를 전달하기 위한 시그널 관련 함수와 이벤트가 발생하였을 때, 이벤트를 처리하는 이벤트 헨들러를 포함한다. 이벤트는 센서에서 받아들이는 시그널 처리와 같은 비동기적인 실시간 애플리케이션의 요구사항을 지원한다. 이벤트 핸들러는 이벤트가 발생하였을 때, 해당 이벤트를 처리하기 위한 로직을 포함한 실시간 스케쥴 될 수 있는 객체이다. 특정 시점에 이벤트가 발생하면 비동기적으로 해당 이벤트를 처리할 수 있는 이벤트 핸들러가 동작하는데 자바 가상머신의 예외 처리기와 다른 점은 이벤트 핸들러가 해당 쓰레드에서 수행되는 것이 아니라 독립된 컴포넌트로서 동작한다는 점이다.

4. 결론

본 논문에서는 내장형 시스템의 비기능적 요구에 맞게 동적 재구성 되는 내장형 미들웨어를 위해 런타임에 스케쥴링 알고리즘의 교체가 수행되는 스케쥴러를 제안하였다. 본 논문에서 제안한 스케쥴러는 적합한 분석 기법을 선택하기 위해 태스크의 특징들을 수집한 뒤 일치하는 가정들이 있는 스케쥴링 알고리즘으로 동적으로 대체한다. 그러나, 스케쥴링 알고리즘의 재구성을 구현할 때는 몇 가지 문제점이 고려되어야 한다. 비실시간 스케쥴링 알고리즘인 FIFOScheduling으로 스케쥴 된 채 수행중인 태스크는 동적 대체되는 다른 실시간 스케쥴링 알고리즘과의 비호환성으로 deadline 기술과 같은 문제가 발생할 수 있다. 따라서, 두 스케쥴링 알고리즘의 공동 사용이 요구될 수 있으나, 본 논문에서는 이에 대한 해결 방안까지는 제시하지 않았다.

현재 본 논문에서 제안한 스케쥴러의 동작 프로세스를 근거로 성능, 유연성, 확장성 요소들간의 이해득실을 고려한 내장형 미들웨어 스케쥴러를 구체적으로 설계, 구현하고 있다.

참고문헌

[1] Edward A. Lee, "What's Ahead for Embedded Software?", *IEEE Computer*, pp. 18-26, September 2000.

[2] Truyen, E., Jorgensen, B.N., Joosen, W., "Customization of Component-based ORB Through Dynamic Reconfiguration", *Technology of Object-Oriented Languages 33*, June 5-8, 2000.

[3] John A. Stankov, Ruiqing Z, Ram Poornaling, Chenyang Lu, Zhendong Yu, Marty Humphr, "VEST: An Aspect-Based Real-Time Embedded System Composition Tool", *IEEE Real-Time Systems Symposium*, 2002.

[4] Kniesel, G., "Type-Safe Delegation for Run-Time Component Adaptation", *Proceedings of the 13th European Conference on Object-Oriented Programming(ECOOP '99)*, R.Guerraoui(Ed.), Lisbon, Portugal, June 14-18, 1999.

[5] 심재홍, 송재신, 최경희, 박승규, 정기현, "다양한 실시간 스케쥴링 알고리즘들을 지원하기 위한 재구성 가능한 스케쥴러 모델", *정보과학회 논문지*, Vol. 29, No. 4, pp. 201-212, 2002.

[6] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.