실시간 객체지향 프로그램의 실행시간을 감시하는 모니터의 설계 및 구현

민 병 준† · 최 재 영†† · 김 정 국††† · 김 문 회††††

요 약

본 논문에서는 실시간 객체 모델인 TMO(Time-triggered Message-triggered Object)에 기반을 둔 실시간 객체지향 프로그램의 시간 제약이 제대로 만족되는가를 시스템 수행 중에 감시하기 위한 모니터의 효과적인 구현 방안에 대하여 논한다. 구현 환경으로 윈도우 OS상의 TMO 프로그램 실행 환경인 WTMOS(Windows TMO System)를 이용하였고, 감시 주체가 되는 수행 시간 모니터의 성능 분석을 위하여 온도 제어를 위한 응용 시스템이 구축되었다. 모니터 대상과 모니터 조건을 TMO 프로그램내에 명시하는 방법과 정의된 모니터의 기능을 WTMOS 내부와 TMO 형태의 응용 객체로 분산시켜서 적은 비용으로 시스템을 구축하는 방법을 제시한다.

Design and Implementation of a Monitor Checking the Execution Time of Real-time Object-oriented Programs

Byoung-Joon Min† · Jae-Young Choi†† · Jung-Guk Kim††† · Moon-Hae Kim††††

ABSTRACT

This paper presents an efficient implementation of a run-time monitor which checks the satisfaction of the timing constraints of real-time object-oriented programs based on a real-time object model called TMO(Time-triggered Messaged-triggered Object). An execution environment of TMO programs on Windows OS, WTMOS(Windows TMO System) has been utilized for the implementation and an experimental application for thermostatic control has been developed to analyze the performance of the run-time monitor. We suggest a methodology for application programmers to specify the target methods with the monitoring conditions and an implementation technique which incurs the small cost by effectively distributing the functions of the monitor into the WTMOS and a TMO object.

1. 서 론

실제로 많은 컴퓨터 시스템 구현들이 설계 단계에서 충분히 엄격한 검증 과정을 거친 후에도 항상 올바르게 동작한다는 것을 보장하기는 어렵다. 그 이유는 구현의 복잡성으로 인하여 시험 과정에서 모든 발생 가능한 경우가 시험되지 못하기 때문이고, 또한 시스템 운용 중에 발생하는 예상치 못한 환경 변화 때문이다. 이로 인하여 시스템 운용 중에 지속적으로 동작 상태를 감시하는 수행시간 모니터(run-time monitor)의 필요성이 대두되었다[10].

일반적으로 수행시간 모니터는 프로그램 디버깅, 시스템의 성능 평가, 또는 시스템의 올바른 실행 상태 확인을 목적으로 사용된다. 이 중에서 시간 제약을 감시하는 수행시간 모니터는 수행 시간 중에 시스템의

※ 본 연구는 한국과학재단 특정기초연구(96-0101-07-01-3)지원과 인천대학교 MRC 지원에 의한 것임
† 종신회원 : 인천대학교 컴퓨터공학과 교수
†† 준 회 원 : 인천대학교 대학원 컴퓨터공학과
††† 정 회 원 : 한국외국어대학교 컴퓨터공학과 교수
†††† 정 회 원 : 건국대학교 컴퓨터공학과 교수
논문접수 : 2000년 11월 13일, 심사완료 : 2000년 12월 27일

시간 관련 정보를 수집하고 그것이 설계 시 기술된 시간 조건을 만족하는가를 확인한다. 위반 사항이 검출되면 모니터는 사용자에게 그 사실을 알리거나 사전에 준비되어 있는 복구 작업을 구동한다.

모니터 관련 연구는 크게 두 가지로 나누어 생각할 수 있다. 하나는 대상 시스템 내에 아무런 모니터 관련 장치를 두지 않고 모니터 하는 경우이다. 즉, 전형적인 입증과 시험 방법에 따라 대상 시스템을 블랙박스로 보고 입력 대비 출력으로 시험하는 것이다. 이는 형식 기술 언어를 이용하여 설계가 시스템 요구사항을 만족한다는 것을 입증하고 이로부터 가능한 시험 시퀀스를 만들어내서 모니터를 이용하여 구현의 오류를 제거하는 것이다. 그러나 이러한 모델은 대부분의 시스템이 동작하는 논리적 상태가 폭발적으로 늘어나는 문제를 안고 있으며 모델 확인이나 이론 증명 방법으로 설계의 정확도가 입증되었다고 하더라도 구현의 정확도를 확인하는데 제한된 시간 내에 모든 가능한 입력 시퀀스가 시험되었다고 보장하기 힘들다[7]. 첫번째 연구 방향의 한계를 극복하기 위한 두 번째 방법으로 대상 시스템 내에 모니터를 위한 개체를 두고 수행 시간 중에 상태를 확인하는 것이다. 이와 같은 모니터는 일반적으로 다음과 같은 목표를 추구한다.

- 최소 방해 작용 : 모니터의 대상이 되는 시스템 내에 모니터링 개체를 적절히 삽입하여 모니터로 인하여 발생되는 방해 작용을 최소화한다.
- 최소 검출지연 시간 : 위반 사항이 발생하면 가급적 빠른 시간 내에 검출이 완료되도록 한다.
- 최소 비용 : 최소한의 추가 비용(모니터 수행시간, 모니터 코드 삽입 노력)으로 모니터 시스템을 구축한다.

수행시간 모니터와 관련된 연구 결과로 다음과 같은 예를 들 수 있다. 참고문헌 [1]에서는 모니터를 위한 개체들을 계층구조로 분산시켜 최적의 구성을 찾기 위한 휴리스틱 알고리즘을 발표하였다. 메쉬와 하이퍼큐브 네트워크에서 제시된 휴리스틱 알고리즘을 모의 시험하여 모니터링 개체들을 분산시키는 것이 통신 비용 면에서 유리하다는 결론을 도출하였다. 모니터링 개체들을 구현하는 방법은 크게 두 가지로 구분할 수 있는데, 한가지 방법은 특수한 하드웨어를 사용하는 것이고 다른 방법은 운영체제 내에 또는 운영체제 상에 모니터링 개체를 삽입하는 것이다. 하드웨어를 이용하는 방법으로 참고문헌 [2]를 들 수 있다. [2]에서는 실행시간 중에 분산 시스템의 동작상태를 감시하고 성능을 측정하기 위한 모니터를 제시하고 있다. 여기서는 시험과 측정을 위한 특수한 프로세서를 두어 투명하고 매우 적은 비용으로 모니터하는 방법이 개발되었다. 모니터링 개체를 운영체제 내부나 응용 프로그램 내에 삽입하는 방법으로 다음과 같은 연구가 진행되었다. 참고문헌 [6]에서는 지능 이동 에이전트 기술을 적용하여 분산 프로그램을 모니터하기 위한 플랫폼 독립적인 방법을 제시하고 있다. 참고문헌 [11]에서는 Ada 프로그램 수행을 연속적으로 모니터하는 방법이 개발되었다. ANNA라는 형식적 기술 언어를 이용하여 Ada 프로그램에 사용자가 주석을 추가하도록 하고 있다. 참고문헌 [10]은 RTL(Real-Time Logic)에 기반을 둔 기술 언어로 작성된 시간 제약 위반을 가능한 빠른 시간에 감시하는 방법을 제시하고 있다. 또한, 참고문헌 [7]에서는 원시 코드에 자동으로 모니터를 위한 코드를 삽입할 수 있도록 해주는 모니터링 스크립트와 요구사항을 표현하는 프레임워크와 언어를 제시하고 있다. 실시간 멀티미디어 응용에 적용할 목적으로 참고문헌 [8]에서는 RT Mach에서 프로세서 사용량을 모니터하여 디스플레이하고 사용자와의 상호작용을 통해서 운영체제가 필요한 자원을 미리 확보할 수 있도록 제어하는 방법이 연구되었다. 참고문헌 [9]에서는 성능 측정을 위한 목적으로 Paradyn이라는 대규모 병렬시스템에서 병렬 프로그램내에 측정을 위한 코드 삽입을 동적으로 제어하여 시간과 공간 비용을 최소화하려는 노력이 있었다.

위에서 언급한 기존의 모니터 기법과 본 논문에서 제시하는 모니터 기법을 비교해보면, 본 논문에서 제시하는 기법은 모니터링 개체를 범용 운영체제 상의 미들웨어와 응용프로그램 내에 삽입하는 방법을 사용하고 있고, 특수한 하드웨어 지원이나 운영체제의 변경 없이 적은 프로그래머의 부담과 시간 및 공간 비용으로 프로그램의 시간 제약 위반을 검출할 수 있도록 고안된 것이다. 본 논문의 2장에서는 수행시간 모니터를 구현하기 위한 환경에 대하여 설명한다. 3장에서는 수행시간 모니터의 기능을 정의하고 실시간 객체 모델인 TMO(Time-triggered Message-triggered Object)를 기반으로 실시간성 관련 사건 중에서 모니터의 대상과 모니터 하는 조건을 기술하는 방법을 제시한다. 4장에서는 3장에서 정의된 수행시간 모니터의 기능을

TMO 실행 환경인 WTMOS(Windows TMO System)에 맵핑시켜서 적은 추가 비용으로 모니터 시스템을 구축하는 과정을 제시한다. 수행 시간 모니터의 성능 분석을 위하여 온도 제어를 위한 응용 시스템이 구축되었는데 실험 결과에 대하여 5장에서 논하고, 마지막으로 6장에서 결론을 맺는다.

2. TMO 모델 및 TMO 프로그램 실행 환경

이 장에서는 실시간 객체 모델인 TMO모델의 특성과 윈도우 OS 상의 TMO 프로그램 실행 환경인 WTMOS에 대하여 설명한다.

2.1 TMO 모델의 특성

최근 TMO(Time-triggered Message-triggered Object) 모델이 실시간 시스템의 효율적인 개발 방법으로 소개된 바 있다[3]. 일반적 객체 지향 프로그램의 객체가 객체 데이터 공간과 멤버 함수들로 구성되는데 반해 TMO 객체 모델은 객체의 특성을 유지하면서 그 구성 요소가 데이터 공간과 동적인 멤버 스레드(메소드)로 구성된다. TMO 객체 모델의 중요한 두 가지 특징은 다음과 같다.

(1) 두 가지로 분리된 메소드 그룹으로 구성된다. 첫번째 그룹은 실시간 클럭에 의해 주기적으로 수행되는 시간 구동 메소드(time-triggered method)들로 SpM(Spontaneous Method)이라고 한다. SpM은 시스템 설계 시에 그 작동이 결정되는 메소드이다. 작동 기간과 주기에 의해 수행되며 각 주기적 작동 시각에 대한 오차 허용 시간과 1회 작동 완료까지의 보장성 제한 시각이 주어진다. SpM의 이러한 시간 명세를 AAC(Autonomous Activation Condition)이라 하며 다음과 같은 표현 방식을 갖는다.

"for t = from 10 : 00am to 10 : 50am every 30min start-during (t, t + 5min) finish-by t+10min".

위의 표현은 {"start-during (10 : 00am, 10 : 05am) finish-by 10 : 10am", start-during(10 : 30am, 10 : 35am) finish-by 10 : 40am"}을 의미한다. 두 번째 그룹은 클라이언트로부터의 메시지, 즉 설계 당시에는 알 수 없고 운용 중에 발생하는 메시지에 의해 구동되는 메시지 구동 메소드(message-triggered method)로 SvM(Service Method)으로 불린다.

(2) SvM과 SpM의 구동이 공유되는 데이터 부분에 대해 충돌이 일어날 때에는 SpM에 우선권을 준다. 즉, SpM의 수행이 SvM에 영향을 받지 않도록 함으로써 설계 시 시간 보장을 용이하게 한 것이다. 이는 객체 데이터 공간을 세그먼트로 나누고 각 세그먼트에 대한 메소드들의 접근 리스트를 선언함으로써 가능한데 이러한 선언을 BCC(Basic Concurrency Constraint)라 한다.

이 모델을 이용하면 실시간 시스템을 TMO 객체들의 연결망으로 표현할 수 있다. 각 객체들은 호출에 의해서 상호 작용한다. 호출은 클라이언트 객체의 SpM이나 SvM이 서버 객체의 SvM에 메시지를 보냄으로써 이루어진다. 클라이언트 메소드(SpM's 또는 SvM's)는 다른 TMO의 SvM의 서비스를 요청하게 되는데, 동시성을 최대한 제공하기 위하여 비동기 호출이 가능하다. 또한, 설계자는 각 SvM과 SpM에 의한 출력에 마감시간을 정할 수 있다.

2.2 TMO 프로그램 실행 환경

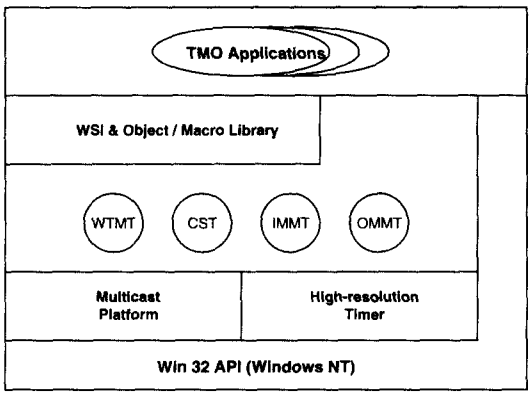
WTMOS[5]는 위와 같은 TMO의 주요 기능을 C++ 객체로 구현한 실시간 객체 네트워크의 수행 엔진으로서, Win32 API 위에서 개발된 미들웨어 플랫폼이다. WTMOS가 실시간 객체 TMO의 수행을 위하여 제공되는 기능은 다음과 같다.

- 최고 1msec 시간단위의 적시 실행
- SpM 및 SvM의 최악(historic worst-case)수행시간과 마감시간여유(deadline-laxity)에 의한 우선순위 제어
- 윈도우메시지 및 분산 IPC 메시지에의한 SvM 구동
- 분산 환경에서의 노드간 클럭 동기화

이러한 기능을 지원하는 TMO 수행환경인 WTMOS는 (그림 1)에 나타낸 바와 같이 계층 구조를 갖는다.

2.2.1 제1계층(Win 32 API)

TMO 객체 인스턴스의 SpM과 SvM은 윈도우 시스템의 스레드로 맵핑되어 그 수행이 관리된다. WTMOS는 SpM이나 SvM에 대해 직접 컨텍스트 변환을 수행하지 않으며, 메소드들의 시간 조건을 충족시키기 위



(그림 1) WTMOS(Windows TMO System) 계층 구조

해 맵핑된 스레드의 우선순위를 제어하는 간접적인 방법을 사용한다. TMO 응용 계층에서는 모든 Win32 API를 사용할 수 있어서 윈도우 관리, GUI등을 사용자에게 제공할 수 있다.

2.2.2 제2계층(Multicast Platform & High Resolution Timer)

1msec 정밀도의 선점(preemption)을 제공하는 Microsoft 'multimedia timer'의 내부 클럭 핸들러로 제3계층의 시스템 태스크들을 주기적으로 구동하여 그들로 하여금 TMO 내의 SpM, SvM의 적시 호출과 마감시간에 의한 우선순위 조정, 분산 노드간의 주기적 클럭 동기화, 분산 IPC 메시지 버퍼의 주기적 관리 등의 작업을 수행토록 한다. 또한, 멀티캐스트 통신 계층으로 Microsoft 'mail-slot' 서비스를 기반으로 TMO 응용 계층에 분산 노드간의 메시지 교환과 버퍼링을 제공한다.

2.2.3 제3계층(System Task Layer)

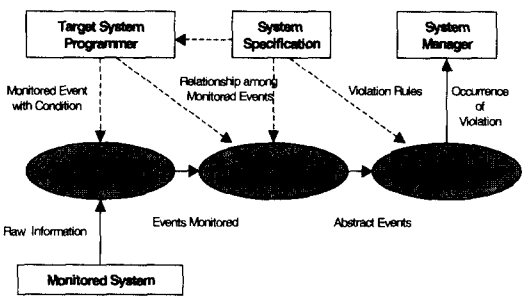
클럭 핸들러에 의해 주기적으로 구동되는 윈도우 시스템의 시간적으로 민감한(time-critical) 스레드 군으로 구성된다. CST(Clock Synchronization Thread)는 분산 환경하에서 각 노드의 클럭을 동기화한다. WTMT (Watchdog & TMO Management Thread)는 주기적으로 구동되어 구동 시간 조건에 의해 SpM을 구동 시킨다. 해당 SpM을 구동시킨 후 노드 내에서 수행중인 모든 SpM과 SvM의 잔여 마감시간, 구동 후 수행시간, 현재까지 최장 수행시간 등을 점검하여 우선순위를 제어한다. IMMT(Incoming Message Management Thread)와 OMMT(Outgoing Message Management Thread)는 메소드간 메시지 전달을 수행한다.

2.2.4 제4계층(WTMOS Service Interface & Object / Macro Library)

WSI(WTMOS Service Interface) 및 Object / Macro Library는 TMO 프로그래머에게 제공되는 TMO 관련 서비스 인터페이스들로 구성된다. 주요 인터페이스로 메소드 관리 인터페이스, 분산 IPC 인터페이스, 윈도우 관리 인터페이스 등이 있다.

3. 수행시간 모니터 구조 설계

수행시간 모니터란 시스템 수행 중에 시스템이 올바르게 동작하는지를 감시할 목적으로 필요한 정보를 입수하고 이 정보를 토대로 현 수행 상태가 시스템 요구 규격을 만족하는가를 판단하는 것을 말한다. (그림 2)에 나타낸 바와 같이 모니터는 원시정보수집, 추상사건추출, 위반사항 보고 기능들로 정의한다. 그림에서 실선 화살표는 수행시간 중에 정보의 흐름을 나타낸 것이고 점선 화살표는 시스템 설계와 구현 단계에서 요구되는 정보를 표시한 것이다.



(그림 2) 수행시간 모니터 기능

3.1 원시 정보 수집

이 기능은 프로그램이 실행될 때 시스템 내부에서 발생되는 정보를 수집하는 기능이다. 발생되는 모든 정보를 자동적으로 저장하는 경우, 짧은 시간동안에 엄청난 분량의 자료가 모이게 되어 메모리 공간과 통신 채널 자원을 소모하게 된다. 따라서, 모니터 대상이 되는 부분을 프로그래머가 프로그램에 명시할 수 있도록 한다. 실시간 시스템 설계자는 설계 시에 많은 사건 중에서 모니터에 의해 감지되어야 하는 사건과 모니터하는 조건을 명시적으로 나타낼 수 있어야 한다. 여기서 사건이란 크게 두 가지로 구분하여 생각할 수 있다. 하나는 특정 코드 블럭(메소드)의 실행이 시작되고 종료되는 것이고 다른 하나는 특정 변수 값이 변경

되는 것이다. 본 논문에서 전자의 경우에 중점을 두어 설명하기로 한다. 메소드 실행의 모니터 조건은 모니터할 시간 범위와 모니터하는 빈도를 말한다. 메소드 실행 사건을 모니터하기 위한 구문은 다음과 같다.

Monitor(MethodName, StartTime, StopTime, Frequency) ;

여기서, MethodName은 모니터 대상 메소드이고, StartTime과 StopTime은 모니터 시작과 종료 시간을 나타낸다. Frequency는 모니터하는 빈도를 나타낸다. 예를 들어 이 값이 n이라 하면 해당 메소드가 n번 실행될 때마다 한번씩 모니터된다.

3.2 추상 사건 추출

본 논문에서 추상 사건은 서로 상관 관계를 가지고 연속적으로 발생되는 일련의 사건을 의미한다. 실제 시스템의 사용자는 개별 사건의 발생 보다는 이와 같은 추상 사건에 더 관심을 가지게 된다. 추상 사건은 시간적 관련이 있는 사건들의 집합에서 한 시작사건으로부터 종료사건으로 연결되는 연속된 사건으로 표현된다. 예를 들어 사건 집합 {E1, E2, E3, E5}이 존재하고 각 사건이 E2, E1, E3, E5의 순서로 발생된다면 이것을 시작사건 E2, 종료사건 E5로 하는 하나의 추상 사건으로 만들 수 있다. 이와 같은 추상사건을 추출하기 위해서는 모니터된 사건들간의 관계 정보가 제공되어야 한다. 프로그래머는 단지 다음과 같은 구문을 원하는 코드 위치에 삽입시켜서 어떠한 순서와 시각에 추상 사건을 구성하는 개별 사건들이 발생하는가를 알 수 있도록 한다.

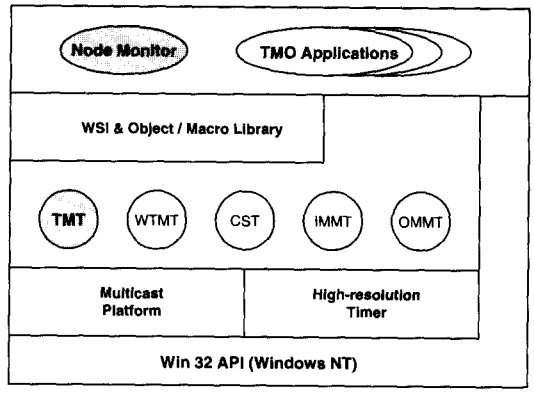
StartAbstractEvent(AbstractEventName) ;
EndAbstractEvent(AbstractEventName) ;

3.3 위반사항 보고

위반사항 보고 기능은 시스템 요구 규격에 정해진 각 추상 사건의 허용 한도를 확인하고 이것이 위반되었을 경우 시스템 사용자에게 보고한다. 보고 방법을 사전에 정의할 수 있도록 한다.

4. 모니터의 구현

앞서 정의한 모니터 기능이 (그림 3)에 도시한 바와 같이 WTMOS의 제3계층에 TMT(TMO Monitor Thread)



(그림 3) 모니터 기능의 맵핑

라 불리는 시스템 태스크와 응용 계층에 노드 모니터라는 TMO 객체로 구현되었다. 이와 같이 모니터를 위한 개체를 분산시킨 이유는 프로그램 수행 중 발생하는 내부 정보 수집의 용이함과 사용자 인터페이스 구현의 편리함을 추구함과 동시에 데이터 저장을 위한 메모리 공간과 통신 채널 자원을 효율적으로 이용하기 위해서이다.

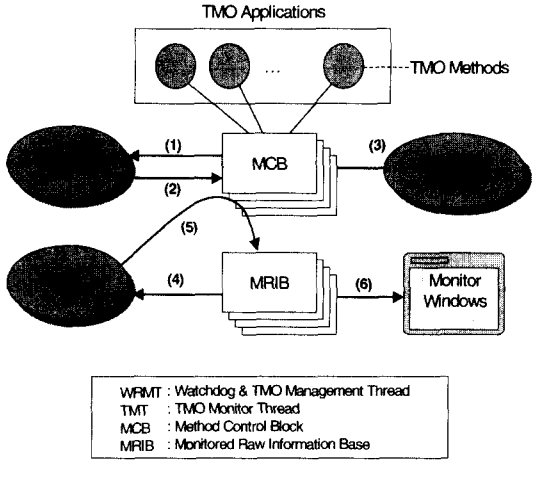
TMT는 프로그램 수행 중에 언제 어떤 메소드를 모니터해야 하는가를 확인하고, 해당 메소드를 제어하는데 필요한 정보에 접근하여 모니터 정보를 수집한 후, 그 중에서 시스템 사용자가 필요로 하는 데이터를 노드 모니터에 전달하는 역할을 한다. 이 논문에서 제시된 모니터가 시스템 사용자에게 제공하는 기능은 두 가지이다. 하나는 사용자에 의해 선택된 특정 메소드의 수행 상태를 프로그램 수행 중에 연속적으로 보여주는 것이고 또 다른 기능은 앞서 정의된 추상 사건, 즉 인과관계가 있는 여러 개별 사건들이 어떠한 시각에 어떠한 순서로 발생하는가를 스냅샷 형태로 묘사하는 것이다. 노드 모니터는 이러한 기능을 제공하는데 필요한 데이터를 TMT로부터 받아서 디스플레이 하는 역할을 한다.

제3계층 안에서 TMT는 각 메소드를 제어하는데 필요한 정보와 실시간 클럭에 접근할 수 있다. 짧은 시간 동안에 엄청나게 불어날 수 있는 모니터 원시 정보를 시스템 태스크 레벨에서 효과적으로 필터링하고 추상사건을 추출하는 일을 TMT가 수행한다. 동시에 시스템 사용자에게 모니터 결과를 디스플레이하고 사용자와의 인터페이스를 처리하는 일은 윈도우 API를 활용할 수 있는 응용 객체 형태의 노드 모니터가 담당한다.

TMO 객체 인스턴스의 SpM이나 SvM들은 윈도우 OS에 의해 스레드로 할당된다. 윈도우 OS는 우선순위에 의한 선점 스케줄링을 제공한다. 프로세스와 무관하게 가장 높은 우선순위의 실행 가능한 스레드가 퀀텀이라는 일정 시간 동안 CPU에 의해 처리되도록 되어 있다.

WTMOS에서 모니터 기능이 동작하는 과정은 다음과 같다. 윈도우 시스템의 고정밀 타이머에 의해 (그림 3)에 나타낸 5개의 시스템 태스크들이 CST, WRMT, TMT, IMMT, OMMT 순서로 10msec의 간격으로 주기적으로 구동된다.

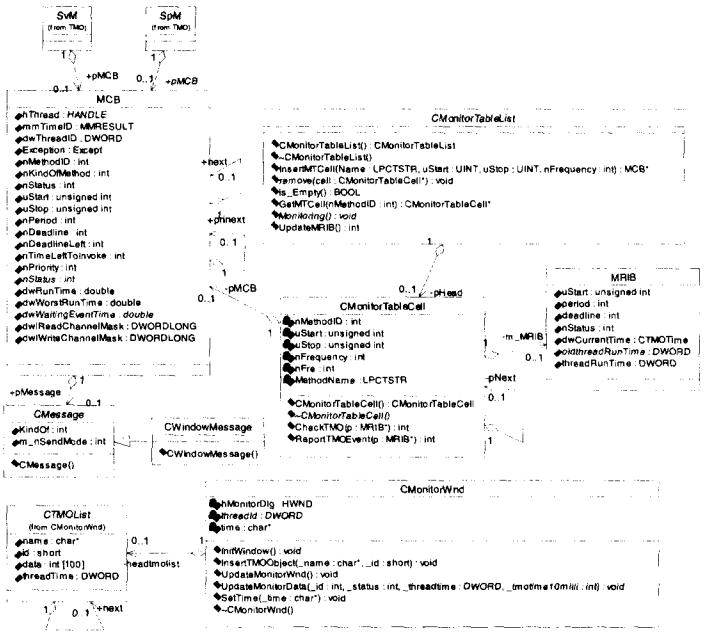
(그림 4)에 모니터 기능이 동작하는 과정을 도시하였다. 그림에서 MCB(Method Control Block)는 각 메소드 수행에 필요한 정보, 예를 들면, 메소드 상태, 마감시간, 최악 수행시간, 우선순위 등을 저장하기 위한 것이다. 그림의 (1)번 화살표는 WRMT가 구동 시간 조건을 검사하기 위하여 MCB를 참조하는 과정이다. WRMT는 (2)번에서 수행 중인 SpM, SvM의 잔여 마감시간, 구동 후 수행 시간 등에 따라 우선 순위를 산출하고 그 결과를 MCB에 저장한다. (3)번은 timeSetEvent 윈도우 API를 이용하여 각 메소드의 마감시간이 경과하면 예외처리 함수를 호출하도록 한 것이다. 원시 정보는 MCB를 참조하여 수집하는데 MCB의 정보가 아주 짧은 시간 간격으로 계속적으로 변하기 때문에 모니터 조건과 일치할 경우에만 정보를 수집하여 MRIB(Monitored Raw Information Base)에 저장한다.



(그림 4) 모니터 동작 시나리오

(4)번에서 TMT는 MRIB를 참조하여 모니터 조건을 검사한다. TMO가 초기화될 때 메소드 시작시간과 마감시간, 모니터 조건은 MCB로부터 MRIB에 전달된다. (5)번에서 TMT는 모니터 조건과 일치하는 TMO 메소드들의 MCB를 참조하여 모니터 정보를 수집하고, (6)번 화살표에 나타낸 것과 같이 모니터된 정보를 시스템 사용자가 볼 수 있도록 모니터 화면에 출력시킨다.

(그림 5)는 모니터 구현과 관련된 클래스들의 관계를 나타낸 것으로 Rational Rose 사의 UML(Unified Modeling Language)도구를 이용하여 도출한 클래스 다이어그램의 일부이다. 하나의 TMO는 여러 SpM과 SvM으로 구성될 수 있고 SpM(SpMName), SvM (SvMName)과 같은 매크로를 통해 선언된다. 각 메소드는 수행에 필요한 정보를 MCB에 저장하고 각 MCB와는 포인터로 연결된다. TMO 프로그램 구현 시 Monitor()라는 매크로를 통해 프로그래머가 모니터하기를 원하는 메소드와 모니터 조건을 명시해줌으로써 WTMOS 자체적으로 정보를 수집할 수 있도록 하였다. Monitor() 매크로를 선언해 주면 CMonitorTableCell이라는 클래스 내에 모니터할 메소드의 이름과 WTMOS 내에서 사용되는 메소드ID, 모니터 조건이 주어진다. CMonitorTableList는 모니터해야 할 전체 메소드의 목록을 갖고 있다. CMonitorTableList에서 WTMOS에서의 모니터 관련 정보를 관리하며 모니터링에 필요한 기능을 수행한다. CMonitorTableList는 모니터하고자 하는 TMO 메소드의 수만큼 MonitorTableCell을 갖는다. 각 Cell에 모니터할 메소드들의 이름과 모니터 조건 정보를 InsertMTCell() 함수를 이용하여 입력한다. 개발자가 응용 프로그램 개발 시에 TMO 메소드를 등록하는 Register 함수에 사용하는 인자중 period, deadline, start_time 값을 MRIB에 각각 period, deadline, uStart이라는 이름으로 저장한다. 이값을 이용하여 TMO 메소드가 시간 제약을 지키는지 검사한다. MCB를 이용하여 WRMT에서 기본적으로 시간 제약 검사를 수행하고 마감시간이 초과했을 경우 예외처리 함수가 호출되도록 한다. TMT에서는 WRMT와는 독립적으로 시간 제약 검사를 수행하여 신뢰성을 높였다. SvM은 deadline값과 메소드가 ACTIVE 상태가 된 시간을 uStart 변수에 입력하여 시간 제약을 검사한다. 또한 TMO 메소드들이 실제로 구동되는 시간을 검사하기 위해 GetThread-Times 윈도우 API를 이용하여 threadRunTime 변수에 각 메소드들의 CPU 사용 시간을 저장하도록 하였다. 이는 MCB에서 ACTIVE 상태이어도 윈도우 OS에 우선 순위



(그림 5) 모니터 구현 클래스 다이어그램

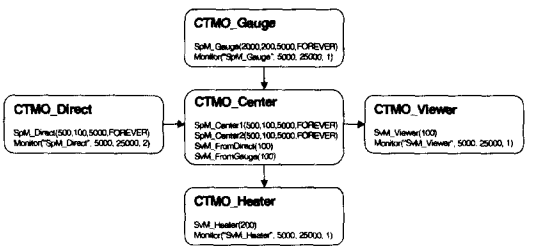
가 더 높은 스레드로 인해 실제로는 WAIT 상태인 경우가 발생할수도 있기 때문이다. 실제 모니터링은 WRMT에서 TMOMethod의 정보를 확인 변경하고 우선순위 스케줄링을 한 후 TMT에서 수행한다. TMT는 CMonitorTableList의 Monitoring() 함수를 호출하여 모니터링에 필요한 모든 작업을 일임한다. Monitoring() 함수는 UpdateMRIB() 함수를 통해 CMonitorTableList에서 갖고 있는 Monitor-TableCell의 모니터 조건을 검사하여 만족하면 모니터링 정보를 수집하여 MRIB를 갱신한다. 변경된 정보를 사용자에게 보고하기 위해 CMonitorWnd의 CTMOList에 각 메소드들의 수행상태를 UpdateMonitorData() 함수를 이용하여 보관하고 일정한 시간 간격으로 화면에 출력하기 위해 UpdateMonitorWnd() 함수를 호출한다.

5. 구현 결과 및 실험

수행 시간 모니터의 성능 분석을 위하여 온도 제어를 위한 응용 시스템이 구축되었는데, 이 장에서는 그 실험 결과에 대하여 논한다.

5.1 응용 시스템

구현된 모니터의 성능을 평가하기 위하여 실시간 TMO 응용 프로그램을 개발하여 성능을 실험하였다. 이는 온도제어를 위한 실시간 시스템응용으로 (그림 6)에 나타낸 것과 같은 개략 구조로 되어 있다.



(그림 6) 응용 시스템의 개략 구조

그림에서 둥근 네모상자 안에 표기한 것은 객체 클래스 이름과 주요 메소드 이름, 그리고 모니터 조건을 나타내는 구문이다. CTMO_Direct는 시스템 사용자가 적정하게 설정해 놓은 온도 값을 입력하여 CTMO_Center에 전달한다. CTMO_Gauge는 실제 온도 측정값을 읽어 들인다. CTMO_Center가 온도 제어를 위한 주 처리를 담당한다. CTMO_Direct와 CTMO_Gauge로부터 정보를 받아서 CTMO_Viewer에게 현재 온도와 설정된 온도값을 전달하고 연산 결과로 나온 제어 명령을 CTMO_Heater

에게 전달한다. 이를 구현한 코드의 일부를 (그림 7)에 나타내었다. 굵은 글자로 나타낸 부분이 3장에서 설명된 모니터를 위한 코드로 응용 시스템 개발자가 추가로 작성해 넣은 부분이다.

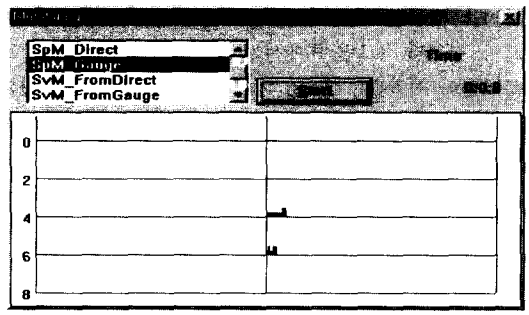
```
SpM_Body(CTMO_Gauge, SpM_Gauge)
void CTMO_Gauge : : SpM_Gauge() {
  WSI.SpM_Register(2000, 200, 5000, FOREVER) ;
  Monitor("SpM_Gauge", 5000, 25000, 1) ;
  WSI.AllocChannel(6, sizeof(CMsgGaugeToCenter)) ;
  while (WSI.SpM_WaitInvocation() == TRUE) {
    CurTemp = ReadTemp() ;
  }
  ...
}
long CTMO_Gauge : : ReadTemp() {
  StartAbstractEvent(Event2) ;
  ...
}
SvM_Body(CTMO_Heater, SvM_Heater)
void CTMO_Heater : : SvM_Heater() {
  WSI.SvM_Register(200) ;
  Monitor("SvM_Heater", 5000, 25000, 1) ;
  WSI.AllocChannel(9, sizeof(CMsgCenterToHeater)) ;
  while(WSI.SvM_WaitInvocation(&m_CenterToHeater,
        9)== TRUE) {
    switch(m_CenterToHeater.KindOf) {
      ...
    }
  }
  WSI.DeallocChannel(9) ;
  WSI.SvM_Exit() ;
}
void CTMO_Heater : : ReceiveCenter(long setTemp) {
  ...
  EndAbstractEvent(Event2) ;
}
```

(그림 7) 응용 시스템 구현 코드 일부

(그림 6)의 각 메소드 명세를 보면 SpM_Gauge는 주기 2000msec, 마감시간 200msec이고 SpM_Direct, SpM_Center1, SpM_Center2는 주기 500msec, 마감시간 100msec이다. SvM_FromDirect, SvM_FromGauge, SvM_Viewer는 마감시간 100msec이고 SvM_Heater는 마감시간이 200msec이다.

이 논문에서 제시된 모니터가 시스템 사용자에게 제공하는 기능은 두 가지이다. 하나는 사용자에 의해 선택된 특정 메소드의 수행 상태를 프로그램 수행 중에 연속적으로 보여주는 것이고 또 다른 기능은 인과 관계가 있는 여러 개별 사건들의 집합인 추상사건 요소들이 어떠한 시각에 어떠한 순서로 발생하는가를 스냅샷 형태로 보여주는 것이다.

(그림 8)은 위의 기능 중 첫번째 기능을 나타낸 것이다. (그림 6)의 응용 프로그램을 실행시키면 (그림 7)에

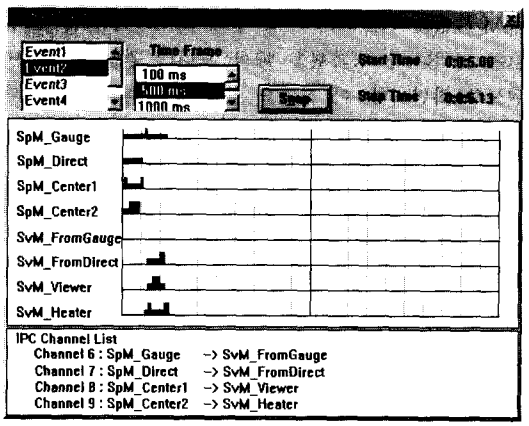


(그림 8) 메소드 모니터 결과 화면

명시된 ***Monitor("SpM_Gauge", 5000, 25000, 1)*** 에 의해 프로그램 시작한 후 5초부터 25초까지 매번 SpM_Gauge 메소드의 수행 상태가 모니터된다. (그림 8)은 약 8초가 경과했을 때의 모니터 결과를 나타낸 윈도우 화면이다. 이 화면을 보면 여러 메소드 중에서 SpM_Gauge가 선택되었고 현재 타임은 8초가 경과한 것을 알 수 있다. 모니터된 정보는 1초 단위로 화면에 나타나며 한 화면에 10초 동안의 메소드 상태를 연속적으로 볼 수 있도록 하였다. 윈도우에 표현된 그래프는 막대 그래프로서 10ms 단위이다. 높이가 낮은 짧은 막대는 메소드가 ACTIVE 상태임을 나타내고, 높이가 높은 긴 막대는 실제 CPU를 점유하고 있는 상태를 나타낸 것이다. 비록 화면에는 10ms 단위로 표시되지만 WTMOS의 타이머는 timeSetEvent라는 Windows API를 사용하여 1ms 단위로 구동된다. 따라서 시간 조건을 위반할 경우, 내부적으로는 1ms 단위로 처리된다.

모니터가 시스템 사용자에게 제공하는 두 번째 기능으로 추상 사건을 구성하는 개별 사건들이 어떠한 시각에 어떠한 순서로 발생하는가를 스냅샷 형태로 나타낸 결과가 (그림 9)이다. 그림에서 추상사건Event2가 선택되었는데 이 결과는 (그림 7)에 나타낸 ***StartAbstractEvent(Event2)***와 ***EndAbstractEvent(Event2)*** 에 의해 모니터된 것이다. 여기서도 마찬가지로 높이가 낮은 짧은 막대는 메소드가 ACTIVE 상태임을 나타내고, 높이가 높은 긴 막대는 실제 CPU를 점유하고 있는 상태를 나타낸 것이다. 짧은 막대는 중복되는 경우가 있지만 긴 막대는 중복되는 경우가 없다. 이는 여러 개의 메소드가 ACTIVE 상태가 될 수 있지만 특정 시간에 동작하는 메소드는 하나 뿐이기 때문이다. 보다 세밀한 시간을 디스플레이할 수 있도록 시간 축

을 1초 단위 뿐 아니라 500ms, 100ms로 선택할 수 있도록 하였다. (그림 9)에서는 500ms가 선택되었고 프로그램이 시작되고 Snap 버튼이 눌려진 후 추상사건의 시작과 종료 시간을 알 수 있다. 이 화면을 통하여 이 추상 사건을 구성하는 각 메소드는 시간 명세에 맞게 구동되는 것을 확인할 수 있다. 화면의 아래부분에 IPC Channel List는 각 SpM이 구동을 끝낸 후 어떤 SvM으로 메시지를 전달하는지를 보여준다. 같은 Channel을 사용하는 SpM, SvM 메소드는 하나의 쌍을 이루고 있다. SvM은 SpM 메소드의 실행 결과로서 구동된다. SpM이 실행을 끝내고 SvM으로 메시지를 전달하면 SvM이 구동되는 것을 알 수 있다.



(그림 9) 추상사건 모니터 결과 화면

이상에서 설명된 모니터의 기능은 프로그램이 시간 제약을 위반하지 않고 정상적으로 작동하는 경우만을 나타낸 것이다. 시간 제약 위반 시, 빠른 시간 내에 시스템을 복구할 수 있는 조치를 취할 수 있도록 지원하는 것은 내부적으로 처리된다. 즉, TMT는 발생 사실을 기록하고 해당 예외처리 루틴을 호출한다. 복구를 위한 예외처리 방법은 이 논문에서는 논하지 않는다.

5.2 모니터 성능 평가

이 절에서는 지금까지 설명된 수행시간 모니터의 성능 분석 결과를 검출 지연 시간 측면과 모니터를 구현하고 실행시키는데 수반되는 추가 비용 측면에 대하여 논한다.

5.2.1 검출 지연 시간

본 논문에서 다룬 수행시간 모니터의 목적은 실시간 객체의 각 메소드들이 설계 시에 주어진 시간 제약을 충족하는가를 확인하고, 어떤 이유에 의해서 시간 제약이 위반되었을 경우, 가급적 빠른 시간 내에 시스템을 복구할 수 있는 조치를 취할 수 있도록 지원하는 것이다. 따라서 시간 제약이 위반된 시점으로부터 복구 조치가 시작되는 시점까지의 시간을 검출 지연 시간이라고 할 때, 이 시간을 줄이는 것이 매우 중요하다. 시간 제약을 위반하는 상황을 만들기 위해서 다음의 두 가지 방법을 이용하여 인위적으로 결함이 발생하도록 한 후에 모니터의 동작을 조사하였다.

- 주처리 작업량을 설계 시 필요량보다 과다하게 증가
- CPU 사용량이 많이 요구되는 다른 윈도우 응용을 실행시켜서 TMO 프로그램이 사용할 수 있는 CPU 자원을 감소

위 두 가지 경우에 대하여 측정결과를 각각 <표 1>과 <표 2>에 나타내었다.

<표 1> 작업량 증가로 인한 결함에 대한 검출지연시간

작업량 증가 비율	100%	150%	200%
검출지연시간(msec)	0.10~0.22	0.03~0.29	0.03~0.27

<표 2> CPU 자원 감소로 인한 결함에 대한 검출지연시간

CPU 자원 감소량	30%	50%	70%
검출지연시간(msec)	0.20~0.25	0.52~0.70	0.30~0.47

비교적 작업량이 많은 SpM_Center1과 SpM_Center2에 대하여 결함을 발생시킨 후 각 메소드의 마감시간으로부터 TMT가 발생시킨 예외처리 스레드가 시작되는 시간차이를 20회 이상 측정하여 평균한 값이다. 실제로 TMO 프로그램이 실행되는 환경이 1msec 단위의 정밀도를 가지고 있는 점을 감안하면 매우 짧은 검출 지연 시간임을 알 수 있다.

5.2.2 모니터의 추가 비용

모니터에 의해 발생되는 추가 비용은 다음의 세 가지 유형으로 나눌 수 있다.

- 프로그래머의 부담
- 시간 비용
- 공간 비용

첫번째, 프로그래머의 부담은 (그림 7)의 예에 나타낸 바와 같이 매우 적다. 모니터하고자 하는 대상과 모니터 조건을 명시한 매크로를 선언하고 추상 사건의 시작과 끝을 나타내는 매크로를 응용 프로그램 내에 삽입하기만 하면 된다. 두번째 시간 비용은 TMT 스레드가 소비하는 시간과 모니터 결과를 화면에 나타내기 위한 시간으로 이루어진다. 이 중에서 TMT가 소비하는 시간의 거의 무시할 정도이다. 화면 디스플레이를 위한 시간은 이보다 훨씬 크지만 시스템 부하에 따라 디스플레이 시간 간격을 조절하여 시간 비용을 감소시킬 수 있다. 마지막으로 공간 비용은 모니터가 차지하는 메모리 소모량을 의미하는데, 실제로 실험을 위해 개발된 응용 프로그램의 경우, 모니터에 필요한 공간, 즉, MRIB의 크기는 원래 필요한 MCB 크기의 약 30%~40% 이다. 따라서, 본 논문에서 구현된 수행시간 모니터의 비용은 크지 않다고 할 수 있다.

6. 결 론

시스템 수행시간 중에 실시간성을 감시하는 수행시간 모니터를 구현하기 위해 모니터의 대상과 모니터하는 조건을 명시하는 방법과 이를 효과적으로 구현하는 기법을 제시하였다. 구현 환경으로 실시간 객체 모델인 TMO와 윈도우OS 상의 TMO 프로그램 실행 환경인 WTMOS를 이용하였다. 본 논문에서 정의된 수행시간 모니터의 기능을 WTMOS와 TMO 응용 객체에 맵핑시켜서 매우 적은 추가 비용으로 모니터 시스템을 구축하였다. 제시된 감시 기법은 모니터링에 수반되는 시간 비용과 공간 비용을 줄일 수 있도록 고안되었고 응용 프로그램내에 간단한 구문을 삽입하여 감시되어야 할 메소드와 감시 조건을 명시할 수 있도록 하여 프로그래머의 부담을 줄이는데 중점을 둔 것이다.

구현된 모니터의 성능을 평가하기 위하여 실시간 TMO 응용 프로그램을 개발하여 성능을 실험하였다. 실험 결과, 검출 지연 시간과 모니터를 구현하고 실행시키는데 수반되는 추가 비용이 모두 양호한 것으로 나타났다.

참 고 문 헌

[1] Jiannong Cao, et. al., On Heuristics for Optimal Configuration of Hierarchical Distributed Monitoring Systems, The Journal of Systems and Software 43, 1998.

[2] Dieter Haban and Dieter Wybranietz, A Hybrid Monitor for Behavior and Performance Analysis of Distributed Systems, IEEE Trans. on Software Engineering, Vol.16, No.2, 1990, 2.

[3] K. Kim, et. al., "A Timeliness-Guaranteed Kernal Model : DREAM Kernal and Implementation Techniques," RTCSA, 1995, 10.

[4] K. Kim and C. Subbaraman, Fault-Tolerant Real-Time Objects, Communications of the ACM, vol.40, no.1, 1997, 1.

[5] J-G Kim, J P Hong, B-J Min, M H Kim, "Modeling of Multimedia Services using the TMO Model," Journal of Computer Systems Science and Engineering, 1998, 5.

[6] Franz Kurfess and Dhaval Shah, Monitoring Distributed Processes with Intelligent Agents, Proc. of IEEE Conference and Workshop on Engineering of Computer-Based Systems, 1999, 3.

[7] I. Lee, et al., "A Monitoring and Checking Framework for Run-time Correctness Assurance," Proc. Korea-US Technical Conf. on Strategic Technologies, 1998, 10.

[8] Clifford Mercer and Ragunathan Rajkumar, An Interactive Interface and RT-Mach Support for Monitoring and Controlling Resource Management, Proc. Of the Real-Time Technology and Applications, 1995, 5.

[9] B. P. Miller and M. D.Callaghan, The PARADYN Parallel Measurement Tools, IEEE Computer, vol 28(11) 1995, 11.

[10] A. K. Mok and G. Liu, "Efficient Run-Time Monitoring of Timing Constraints," Proc. Real-Time Technology and Applications, 1997, 6.

[11] S. Sankar and M. Mandal, "Concurrent Runtime Monitoring of Formally Specified Programs," IEEE Computer, 1993, 3.

[12] B.A. Schroeder, "On-line Monitoring : A Tutorial," IEEE Computer, 1995, 6.

민 병 준

e-mail : bjmin@lion.inchon.ac.kr

1983년 연세대학교 전자공학과 (학사)

1985년 연세대학교 전자공학과 (석사)

1991년 미국 캘리포니아 주립대학교(UC Irvine) 전기 및 컴퓨터공학과(박사)

1984년~1986년 삼성전자 연구원

1992년~1994년 한국통신 선임연구원

1995년~현재 인천대학교 컴퓨터공학과 조교수

관심분야 : 병렬처리, 분산시스템, 통신망관리

최 재 영

e-mail : vctr@isis.inchon.ac.kr

1999년 인천대학교 전자계산학과 (학사)

1999년~현재 인천대학교 컴퓨터공학과(석사과정)

관심분야 : 분산처리, 시스템 소프트웨어

김 정 국

e-mail : jgkim@maincc.hufs.ac.kr

1977년 서울대학교 계산통계학과 (학사)

1979년 한국과학기술원 전산학 (석사)

1986년 한국과학기술원 전산학 (박사)

1983년~현재 한국외국어대학교 컴퓨터공학과 교수

1994년~1995년 미국 캘리포니아주립대학교 (UC Irvine) 교환교수

관심분야 : 운영체제, 소프트웨어공학

김 문 회

e-mail : mhkim@kkucc.konkuk.ac.kr

1979년 서울대학교 전기공학과 (학사)

1981년 서울대학교 전기공학과 (석사)

1985년 미국 남플로리다대학교 전산학과(석사)

1991년 미국 캘리포니아 주립대학교(UC Berkeley) 전산학과(박사)

1991년~현재 건국대학교 컴퓨터공학과 교수

2000년~현재 미국 캘리포니아주립대학교(UC Irvine) 교환교수

관심분야 : 소프트웨어공학, 실시간분산처리시스템