ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

이기종 분산 환경에서 동적 재구성이 가능한 통합 관리 및 모니터링 시스템☆

Dynamic Reconfigurable Integrated Management and Monitoring System for Heterogeneous Distributed Environments

민 법 기[1] 서 용 진[1] 김 현 수[1*] 국 승 학[2] 정 용 환[2] 김 점 수[2]
Bup-Ki Min Yongjin Seo Hyeon Soo Kim Seunghak Kuk Yonghwan Jung Chumsu Kim

요 약

본 논문에서는 이기종 분산 컴퓨팅 환경에서의 관리 대상이 되는 시스템이나 애플리케이션들의 정보 모델을 동적으로 재구성하는 통합 관리/모니터링 시스템을 개발한다. 다양한 플랫폼 위에서 동작하는 하위 시스템들이 추가, 제거, 수정되는 상황에서 분산 시스템을 효과적으로 관리하기 위해서는 분산 시스템의 형상과 통합 관리/모니터링 시스템의 정보가 일치해야 한다. 이에 시스템의 형상이 변화되었을 때, 시스템의 변경된 형상과 통합 관리/모니터링 시스템의 정보가 동기화되어야 하며, 동기화가 수행되는 동안 시스템은 정지 없이 모니터링 정보를 사용자에게 제공할 수 있는 가용성이 보장되어야 한다. 본 논문에서는 관리 대상인 하위 시스템에서 하드웨어나 소프트웨어의 형상의 변경이 있을 때 변경 내용을 통합 관리/모니터링 시스템에 통보하고, 통보된 정보를 바탕으로 관리 대상인 하드웨어 및 소프트웨어의 정보들을 동적으로 재구성 하는 방법을 제안한다. 이를 통해 어떤 이기종 분산 시스템도 통합 관리/모니터링 시스템에 반영하여 신뢰성 있는 통합 관리를 수행할 수 있을 것으로 기대된다.

☞ 주제어 : 이기종 분산 시스템, 통합 관리 및 모니터링 시스템, 정보 모델, 정보 모델 동적 재구성

ABSTRACT

In this paper, we develop an integrated management/monitoring system that supports to dynamically reconfigure information models for systems or applications managed by heterogeneous distributed systems. When the subsystems on diverse platforms are added, removed, or modified, the altered configurations should conform to the configuration information of the integrated management/monitoring system. Further, upon the system configurations being changed, the altered system configurations should be synchronized with the information on the integrated management/monitoring system. Moreover, availability should be assured during synchronization to the extent that users can access the monitoring information with no system halting. This paper focuses on notifying the integrated management/monitoring system of any changes in hardware/software configurations on any subsystems under its management, and on dynamically re-configuring the information about hardware and software being managed based on the information notified. Finally, we expect that this research will be contributory to carrying out reliable integrated management by reflecting the information on any heterogeneous distributed systems in the integrated management/monitoring system.

☞ keyword : Heterogeneous Distributed Systems, Integrated Management and Monitoring System, Information Model, Information Model Dynamic Reconfiguration

1. 서 론

분산 컴퓨팅 시스템은 원거리에 위치한 하위 시스템들을 하나의 시스템으로 통합한 것으로 확장성, 생산 성, 자원의 공유 등의 측면에서 많은 장점을 제공한다[1]. 이러한 장점으로 인해 공정 관리 시스템, 대형 플랜트, 함정 전투 관리 시스템 등 다양한 분야에 적용되고 있지만 서로 독립적인 공간에 위치해 있어 각 하위 시스템에서 발생하는 문제점을 파악하는데 어려움이 있다. 이러한 문제점을 해결하기 위해 등장한 것이 분산 컴퓨팅 시스템을 위한 통합 관리/모니터링 시스템이다[2]. 통합 관리/모니터링 시스템은 독립적인 위치에 존재하는 다양한 하위 시스템들의 상황을 중앙에서 파악하여 사용자에게 제

1 Dept. of Computer Sc. & Eng., Chungnam National University, Daejeon 305-764, Korea
2 Agency for Defense Development, Daejeon 305-152, Korea
* Corresponding author (hskim401@cnu.ac.kr)
☆ 본 연구는 국방과학연구소의 지원으로 수행되었습니다.(계약번호 UD090017KD)
[Received 11 September 2012, Reviewed 18 September 2012, Accepted 8 October 2012]

공한다. 그러나 분산 시스템은 언제라도 새로운 하위 시스템이 추가되거나 기존 시스템이 제거되는 등의 변화가 일어날 수 있다[3,4]. 변경된 시스템의 형상은 통합 관리/모니터링 시스템에도 즉각적으로 반영되어야만 사용자에게 신뢰성 높은 통합 시스템 정보를 제공할 수 있다. 따라서 분산 시스템과 통합 관리/모니터링 시스템 사이에서 분산 시스템의 형상 정보의 동기화가 가장 큰 이슈가 되고 있다[5-7]. 또한 동기화 과정에서 통합 관리/모니터링 시스템이 정지하게 된다면, 이 때 발생한 정보는 사용자에게 제공하지 못하게 된다. 이러한 문제는 치명적일 수 있다. 따라서 동기화뿐만 아니라 통합 관리/모니터링 시스템이 정지하지 않고 동작할 수 있는 가용성을 보장할 수 있어야 한다[5].

본 논문에서는 동적으로 형상이 변하는 분산 시스템과 그 분산 시스템을 관리하고 모니터링 하는 통합시스템 사이의 동기화 문제와 가용성 문제를 해결하기 위한 방안을 제시한다. 이를 위해 관리 대상인 하위 시스템에서 하드웨어나 소프트웨어의 형상의 변경이 있을 때 변경 내용을 통합 관리/모니터링 시스템에게 통보한다. 통합 관리/모니터링 시스템에서는 통보된 정보를 바탕으로 관리 대상인 하드웨어 및 소프트웨어의 정보들을 동적으로 재구성한다.

2. 관련연구

동적 재구성의 목적은 관리 대상 시스템이 현재의 구성에서 시스템을 중단하지 않고 새로운 구성으로 변경되는 것이다[3]. 따라서 시스템이 중단되지 않고 구성이 변경되기 위해서는, 동적 재구성을 위한 규칙이 필요하다.

[6]의 연구에서는 관리 대상 시스템에서 시스템의 형상이 변경되었을 때, 동적 재구성을 위한 작업을 수행하기 위해 노드 생성(node creation), 노드 삭제(node removal), 노드 연결 및 연결해제(node linking & unlinking)의 방법과 규칙을 제시한다. 이러한 작업들은 (표 1)과 같은 동적 재구성 규칙을 갖는다.

노드 삭제 규칙에서는 하나의 노드가 제거될 때, 제거된 노드 때문에 다른 노드들에 문제가 발생하지 않아야 한다. 따라서 제거가 예상되는 노드는 미리 격리 시킨다. 격리된 노드는 시스템에 영향을 줄 수 없기 때문이다.

노드 연결 및 연결해제 규칙에서 연결이나 연결해제에 대한 지시를 받은 노드는 수행중인 작업이 완료될 때까지 노드의 상태가 일관되어야만 문제가 발생하지 않는다. 노드가 실행상태일 경우 노드의 상태가 에러나 정지

(표 1) 동적 재구성 작업 상황별 규칙
(Table 1) Rule of Dynamic Reconfiguration

발생 동작	규칙
노드 삭제	제거 대상 노드는 다른 노드와 직접 연결이 되어있지 않아야 하기 때문에 실행이 중단되고 격리되어야만 한다.
노드 연결 및 연결해제	연결이나 연결해제 지시를 받은 노드는 정지 상태에 있어야 한다.
노드 생성	생성된 노드는 사용자에게 제어를 받기 전까지 정지되어야 한다.

등 다양한 상태로 변경될 수 있기 때문에 예상치 못한 문제가 발생할 수 없는 정지 상태로 미리 설정해야만 한다.

노드 생성 규칙에서는 새로 만들어진 노드는 초기에는 격리되어 있기 때문에, 응답을 받아야만 초기화 작업을 수행할 수 있다. 따라서 노드의 상태는 정지 상태로 존재하고, 실제 명령을 받았을 때, 노드의 상태가 변경되고 초기화 될 수 있다.

본 논문에서는 위에서 설명한 동적 재구성 규칙을 참조하여 하드웨어의 추가 및 제거, 애플리케이션의 추가, 삭제, 정보 변경 등의 발생 가능한 형상 변화를 정의하고, 이에 대응할 수 있는 방법을 제시한다.

이기종 분산 컴퓨팅 환경을 통합 관리하고 모니터링 하는 방안에 대해 다양한 연구가 진행되고 있다. 이 연구들의 공통점은 분산 시스템과 통합 관리/모니터링 시스템 간의 동기화를 해결하려는 것이다.

[5]의 연구에서는 AAOP(Adaptability Aspect-Oriented Programming)를 기반으로 동적 재구성 모니터링 시스템을 개발한다. 이 연구에서는 모니터링 시스템의 재시작 없이 동기화 문제를 해결하기 위하여 Monitoring Manager를 두고 있다. Monitoring Manager는 하위 시스템을 관리하는 Monitoring Agent들을 계속 추적한다. 새로운 Monitoring Agent가 발견되면 새로운 시스템이 추가된 것을 의미하고, 발견과 동시에 Monitoring Manager를 통해 모니터링 시스템에 등록한다. 이런 방식으로 관리 대상 시스템과의 동기화 문제를 해결한다.

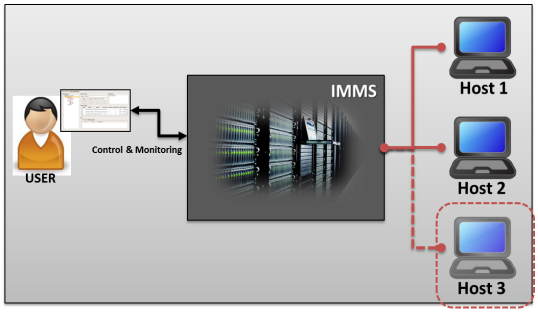
[7]의 연구에서는 그리드 컴퓨팅 환경에서 자원을 모니터링 하기 위한 시스템을 개발하였다. 이 연구에서는 관리 대상 시스템의 확장성, 유연성, 상호운용성에 대응하기 위해 모니터링 시스템에서도 이를 반영할 수 있는 방안을 제안하고 있다. 관리 대상 시스템의 변화에 대응하기 위해 DREAM(Dynamic REflective Asynchronous Middleware)을 사용한다. DREAM은 비동기식 분산 서비

스 플랫폼으로 비동기식 메시지를 통해 분산 서비스의 구성, 배치, 관리를 담당한다.

우리 연구에서는 통합 관리/모니터링 시스템이 관리 대상 시스템의 형상 정보를 유지하기 위해 정보 모델을 유지하고 있다. 하위 시스템에서 하드웨어나 소프트웨어의 형상이 변경되면 그 정보가 통합 관리/모니터링 시스템에 통보되고 그 정보를 바탕으로 정보 모델을 동적으로 재구성한다. 이렇게 함으로써, 실제 시스템과 정확히 일치하는 통합 관리/모니터링을 수행할 수 있다.

3. 정보 모델 기반 통합 관리/모니터링 시스템

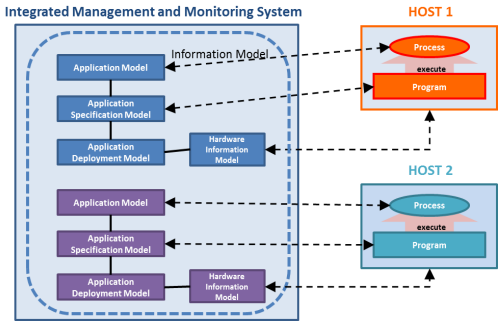
(그림 1)은 관리 대상이 되는 분산 하위 시스템들(Host 1, Host 2, Host 3)과 그것들을 관리하고 모니터링 하기 위한 통합 관리/모니터링 시스템(IMMS)과의 관계를 보여준다. 통합 관리/모니터링 시스템은 분산 하위 시스템들을 관리하고 모니터링 하기 위하여 내부적으로 하위 시스템들의 형상에 대한 정보를 정보 모델로써 유지하고 있다.



(그림 1) 분산 컴퓨팅 시스템과 통합 관리/모니터링 시스템과의 관계

(Figure 1) Relationship of Distributed Systems and Integrated Management/Monitoring System

정보 모델은 (그림 2)의 왼쪽 부분과 같은 형태를 갖는다. (그림 2)에서 오른쪽 부분은 관리 대상이 되는 분산 시스템의 하위 요소들을 보여주고 있다. Host는 컴퓨터를 의미하고, 프로그램은 컴퓨터에 탑재된 소프트웨어를 의미한다. 소프트웨어가 비실행 상태이면 프로그램으로 존재하고, 실행 시에는 프로세스의 형태로 존재하게 된다. 정보 모델은 이러한 요소들과 1:1 대응 관계를 갖는 클래스로 표현된다. 즉, Host에 대한 정보는 Hardware Information Model 클래스가 유지하고, 프로그램에 대한 정보는 Application Specification Model 클래스가, 프로세스에 대한 정보는 Application Model 클래스가 각각 유지한다. 추가로 Application Deployment Model 클래스는 어떤 프로그램이 어떤 Host에 탑재되어 있는지에 대한 정보를 유지한다. 실제로 정보 모델은 그림과는 달리 복잡한 구조이지만 여기서는 간략한 형태로 보여준다.



(그림 2) 애플리케이션과 하드웨어에 대한 정보 모델

(Figure 2) Information Models for Applications and Hardware

이 정보 모델들은 추상적이기 때문에 다양한 플랫폼에서 동작하는 애플리케이션들을 플랫폼에 독립적으로 하나의 관점에서 일관성 있게 표현하는 것이 가능하다. 정보 모델에서 유지하는 정보는 다양한 서비스를 통하여 사용자에게 다양한 측면에서 시스템 상황을 파악하고 제어할 수 있게 해준다.

4. 정보 모델 기반 동적 재구성

4.1 동적 재구성의 필요성

통합 관리/모니터링 시스템에서는 다음의 두 가지 문제점이 가장 큰 이슈가 되고 있다.

- 실제 관리 대상 분산 시스템과 통합 관리/모니터링 시스템 간의 동기화
- 동기화 과정에서 통합 관리/모니터링 시스템의 가용성 보장

통합 관리/모니터링 시스템은 관리 대상이 되는 분산 시스템의 시스템 상황을 실시간으로 관리하고 모니터링

한다. 따라서 관리 대상이 되는 분산 시스템의 형상과 일치하는 정보 모델을 유지하여야 한다. 또한 정보 모델의 동기화 중에도 동기화 대상이 아닌 다른 하위 시스템들은 관리되고 모니터링 되어야 한다.

관리 대상 시스템의 형상이 변경되면, 변경된 형상을 통합 관리/모니터링 시스템에 반영하는 방법은 두 가지가 있다. 첫째는 관리 대상 시스템 전체의 형상을 반영한 형상 파일을 작성하고 그것을 바탕으로 통합 관리/모니터링 시스템의 정보 모델을 재구축하는 것이다. 이 방법의 장점은 접근 방법이 단순하고, 구현이 쉽다는 것이다. 본 논문의 통합 관리/모니터링 시스템의 초기 버전 [2]에서는 이 방법을 구현하였다. 그러나 이 방법의 가장 큰 단점은 동기화 중에 모든 관리/모니터링 활동을 중지함으로써 가용성을 보장하지 못한다는 것이다. 두 번째 방법은 정보 모델의 전체가 아닌 일부분만을 재구축하는 방법이다. 즉 형상이 변경된 관리 대상 시스템에 대해서만 스펙을 재작성하고 그것을 바탕으로 부분적으로 정보 모델을 재구축하는 방법이다. 이 방법은 통합 관리/모니터링 시스템 전체에 걸쳐서 수행되는 작업이 아니기 때문에 통합 관리/모니터링 시스템을 중지하지 않고 부분적으로 정보 모델을 재구축할 수 있다. 이 방법의 가장 큰 장점은 형상이 변경되지 않은 시스템들에 대해서는 계속해서 관리/모니터링 작업을 수행할 수 있다는 것이다. 즉 가용성을 보장할 수 있다. 그러나 이 방법의 단점은 구현이 복잡하다는 것이다. 정보 모델을 동적으로 일부분만 재구축하기 위해서는 정보 모델에 내재되어 있는 다양한 의존성 정보를 정확하게 파악하고 수정하여야 하기 때문이다.

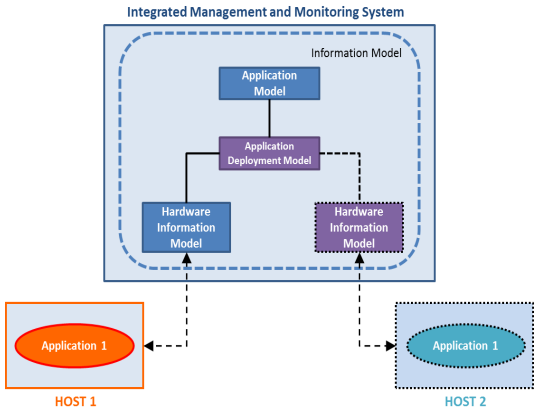
위의 두 가지 이슈를 해결하기 위해 본 논문에서는 정보 모델을 동적으로 재구성하는 방안을 채택한다.

4.2 관리 대상 시스템의 형상 변화

관리 대상 시스템의 형상의 변화는 다음과 같은 경우에 발생할 수 있다.

- 하드웨어의 추가
- 하드웨어의 제거
- 애플리케이션의 추가
- 애플리케이션의 삭제
- 애플리케이션의 정보 변경

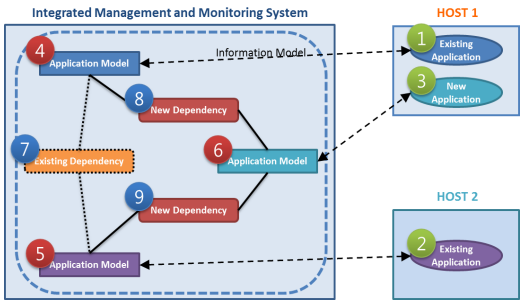
하드웨어 형상의 변화는 새로운 하위 시스템의 추가나 기존 하위 시스템의 제거로 인해 발생할 수 있는데, 이것은 하드웨어 내부에서 동작하고 있는 모든 애플리케이션과도 연관이 되기 때문에, 애플리케이션이 동작할 수 있는 하드웨어에 대한 배치 문제도 함께 고려되어야 한다.



(그림 3) 하드웨어의 추가에 따른 애플리케이션의 배치 변경
(Figure 3) Changes to Deployment of Application due to adding Hardware

(그림 3)은 하드웨어의 추가에 따른 배치정보가 수정되는 상황을 보여준다. 호스트 1에 존재하는 애플리케이션 1과 동일한 (통합 관리/모니터링 시스템의 정보 모델까지 공유하는) 애플리케이션 1을 가진 호스트 2가 새로 추가될 경우, 기존 애플리케이션의 정보 모델을 공유하고 Application Deployment Model 클래스가 수정되면 호스트 2에서도 애플리케이션 1이 동작할 수 있게 된다.

애플리케이션 형상의 변화는 동작하고 있는 시스템에 새로운 기능을 추가하거나 변경할 경우에 발생할 수 있다. 새로운 애플리케이션이 추가되거나 제거되면 기존 애플리케이션간의 의존성에 영향을 줄 수 있다. 예를 들어, 사용자에게 정보를 제공하는 사용자 인터페이스와 정보를 관리하는 데이터베이스로 구성된 물류 관리 시스템을 살펴보자. 이 시스템은 데이터베이스가 실행되어야만 사용자 인터페이스를 통해 사용자에게 정보를 제공할 수 있기 때문에 데이터베이스가 먼저 실행되고, 사용자 인터페이스가 나중에 실행되는 실행에 관련된 의존성이 존재한다. 이 시스템에 보안 요소를 강화하기 위하여 보안 관련 애플리케이션이 추가된다면 데이터베이스가 실행되고, 보안 애플리케이션이 실행된 후, 사용자 인터페이스가 실행되는 구조로 의존성이 변경된다. (그림 4)는



(그림 4) 애플리케이션 추가에 따른 의존성 변경
(Figure 4) Changes to Dependency due to adding Application

이러한 의존성 변경을 보여주는 그림이다. 애플리케이션이 새로 추가되면서 기존의 의존성은 없어지고 새로운 의존성이 추가됨을 의미한다.

본 논문에서는 하드웨어와 애플리케이션간의 배치문제, 애플리케이션과 애플리케이션 사이의 의존성과 관련된 문제를 고려하여 정보 모델의 동적 재구성을 수행한다.

4.3 동적 재구성 방법

본 논문에서는 정보 모델의 동적 재구성을 위해 호스트 단위의 하드웨어와 애플리케이션의 정보를 이용한다.

4.3.1 하드웨어 구성 정보

하드웨어에 대한 형상 정보는 호스트 전문가가 작성한다. 호스트 전문가는 하드웨어에 대한 정보를 정확히 파악하고 작성한다. 하드웨어 형상 정보는 (표 2)와 같다.

(표 2) 동적 재구성을 위한 하드웨어 정보
(Table 2) Hardware information for dynamic reconfiguration

항목	설명
호스트 이름	호스트의 이름. 유일하게 식별되는 이름을 가져야 함
운영체제	호스트의 운영체제 정보
경고 부하(Warning Load)	부하분산을 위한 호스트의 경고 수준 부하 설정 값
위험 부하(Danger Load)	부하분산을 위한 호스트의 위험 수준 부하 설정 값
CPU 성능 점수	호스트의 CPU 성능 점수
메모리 성능 점수	호스트의 메모리 성능 점수

(표 3) 동적 재구성을 위한 애플리케이션 정보
(Table 3) Application information for dynamic reconfiguration

항목	설명
애플리케이션 이름	애플리케이션의 이름. 유일하게 식별되는 이름을 가져야 함
명령(Command)	애플리케이션이 수행 가능한 명령을 기술
중복 그룹 (Redundancy Group)	애플리케이션은 결함허용을 위해 결함허용을 위한 그룹에 대한 정보를 기술
운영체제	애플리케이션이 동작 가능한 운영체제 정보
OS Maximum Version	애플리케이션이 동작 가능한 운영체제의 최대 버전 정보
OS Minimum Version	애플리케이션이 동작 가능한 운영체제의 최소 버전 정보
Startup Dependency	애플리케이션이 실행하는데 있어서 필요한 의존성 정보
Shutdown Dependency	애플리케이션이 종료하는데 있어서 필요한 의존성 정보

4.3.2 애플리케이션 구성 정보

애플리케이션에 대한 형상 정보는 애플리케이션 관리 전문가가 작성한다. 애플리케이션의 형상 정보는 (표 3)과 같다.

4.3.3 동적 재구성을 위한 절차

통합 관리/모니터링 시스템에서 동적 재구성을 위한 절차는 (표 1)에서 소개한 동적 재구성 규칙을 활용하여, 본 논문의 통합 관리/모니터링 시스템에 맞게 변형하였다. 이를 위해 생성, 제거, 연결 및 연결해제 등의 상황에 맞게 실제 관리 대상 시스템 및 애플리케이션의 실행이 정지되는 등의 임의의 제어 명령이 발생하게 된다. 계속해서 동적 재구성이 발생하는 각 상황의 절차를 기술한다.

4.3.3.1 하드웨어 추가

새로운 하드웨어의 추가는 하드웨어뿐만 아니라 새로운 애플리케이션도 추가됨을 의미한다. 따라서 하드웨어의 추가는 하드웨어 정보 모델의 추가, 애플리케이션의 정보 모델의 생성 및 추가, 애플리케이션이 동작 가능한 하드웨어에 대한 배치 정보의 추가 또는 수정 작업이 동반된다. 따라서 하드웨어 및 애플리케이션의 형상을 모두 고려한 동적 재구성이 수행되어야 한다. 하드웨어의

추가는 새로운 하위 시스템 자체가 추가되는 것이므로 애플리케이션들도 모두 새로 추가되며, 그 내부의 애플리케이션의 의존성도 독립적으로 존재하게 된다.

하드웨어 추가에 대한 수행 절차는 다음과 같다.

Algorithm - Add Hardware
In : hardware h, set of application A, set of dependency Dep
Out : void

```
1   function AddHardware(h, A, Dep) {
2     A_stop = ∅;
3       for each a_i∈A do {
4         D = ⋃_i (h,a_i);
5         for each (a_n, a_m)∈Dep do {
6           if (a_i = a_n) {
7             stopRunning(a_m);
8             A_stop = A_stop ∪ a_m;
9           }
10          if (a_i = a_m) {
11            stopRunning(a_n);
12            A_stop = A_stop ∪ a_n;
13          }
14        }
15      }
16      IM = IM ∪ (h, A, D, Dep);
17      startRunning(A_stop);
18      startRunning(h);
19    }
```

- $IM=(H,A,D,Dep)$ 정보 모델을 의미
- H는 하드웨어 정보 모델의 집합이다.
- A는 애플리케이션 정보 모델의 집합이다.
- $D \subseteq H \times A$는 배치 정보 모델의 집합이다. 만약 특정 애플리케이션 $a \in A$가 특정 하드웨어 $h \in H$에서 동작할 수 있다면, 하드웨어 h와 애플리케이션 a는 서로 배치 정보 $(h,a) \in D$를 갖는다.
- $Dep \subseteq A \times A$는 의존성 정보 모델의 집합이다. 만약 애플리케이션 $a_i \in A$가 실행 혹은 종료됨에 있어 애플리케이션 $a_j \in A$의 상태에 영향을 받는다면, 애플리케이션 a_i와 a_j는 서로 의존성 관계 $(a_i, a_j) \in Dep$를 갖는다.
- 7번째 줄과 11번째 줄의 stopRunning() 함수는 실제 관리 대상이 되는 하드웨어나 애플리케이션의 실행을 중단 시키는 함수이다.
- 17번째 줄과 18번째 줄의 startRunning() 함수는 중단되어 있던 실제 관리 대상이 되는 하드웨어나 애플리케이션을 실행시키는 함수이다.

4.3.3.2 하드웨어 제거

기존 하드웨어의 제거는 하드웨어의 추가와 마찬가지로 하드웨어 내부에 존재하는 애플리케이션들의 삭제도 포함한다. 따라서 하드웨어가 제거되면, 통합 관리/모니터링 시스템 내부의 하드웨어 정보 모델뿐만 아니라 하드웨어 내부의 애플리케이션 모델과 배치정보에 존재하는 하드웨어 정보들도 삭제되어야 한다. 하드웨어가 제거되었을 때는 하드웨어를 제거하기 전에 호스트에서 통합 관리/모니터링 시스템으로 하드웨어가 제거되었다는 메시지와 함께 하드웨어의 정보 및 하드웨어에서 동작하던 애플리케이션들의 정보도 전송해주어야 한다. 하드웨어 제거도 하드웨어 추가와 마찬가지로 하나의 하위 시스템 자체가 제거되게 된다.

하드웨어 제거에 대한 절차는 다음과 같다.

Algorithm - Remove Hardware
In : hardware h
Out : void

```
1   function RemoveHardware(h) {
2     stopRunning(h); A_stop = ∅;
3     D_rem = ∅; Dep_rem = ∅;
4     D = IM.getDeploymentSet();
5     for each (h_i,a_j)∈D do {
6       if (h = h_i) {
7         stopRunning(a_j);
8         A_stop = A_stop ∪ a_j;
9         D_rem = D_rem ∪ (h, a_j);
10      }
11    }
12    A_rem = A_stop;
13    for each (h_k,a_j)∈D do { //h_k ≠ h_i
14      for each a_n∈A_stop do {
15        if (a_n = a_j) {
16          A_rem = A_rem − {a_n};
17        }
18      }
19    }
20    A_stop = ∅;
21    Dep = IM.getDependencySet();
22    for each (a_i,a_j)∈Dep do {
23      for each a_n∈A_rem do {
24        if (a_n = a_i) {
```

```
25                  stopRunning(a_j);
26                  A_stop = A_stop ∪ a_j;
27                  Dep_rem = Dep_rem ∪ (a_n, a_j);
28                }
29                if (a_n = a_j) {
30                  stopRunning(a_i);
31                  A_stop = A_stop ∪ a_i;
32                  Dep_rem = Dep_rem ∪ (a_i, a_n);
33                }
34              }
35            }
36            IM = IM− (h, A_rem, D_rem, Dep_rem);
37            startRunning(A_stop − A_rem);
38          }
```

- 4번째 줄의 getDepolymentSet() 함수는 모든 배치 정보 모델을 가져오는 함수이다.
- 13~19번째 줄은 특정 애플리케이션에 대한 배치 정보가 두 개 이상 존재할 때, 어느 하나의 하드웨어만 삭제될 경우에는 삭제되지 않는 하드웨어에 대한 배치 정보를 유지하기 위한 코드이다.

4.3.3.3 애플리케이션 추가

애플리케이션의 추가는 하드웨어 내부의 애플리케이션 형상이 변경될 때 이루어지는 작업 중 하나이다. 애플리케이션이 추가되면서 다른 애플리케이션과의 의존성이 생기면서, 의존성 정보가 변경될 수 있기 때문에 영향받는 애플리케이션에 대해서도 정보가 수정되어야 한다. 따라서 추가된 애플리케이션과 시작 의존성 및 종료 의존성을 가진 애플리케이션들이 존재하는 하드웨어에 의존성이 변경되었음을 알려 동기화가 제대로 이루어질 수 있도록 한다.

예를 들어, 그림 4의 상황을 고려하자. 호스트 1과 호스트 2에는 애플리케이션 1번과 2번이 존재한다. 통합 관리/모니터링 시스템 내부의 정보 모델에는 호스트의 상황을 반영한 4번과 5번의 애플리케이션 정보 모델과 이들 사이에 의존성을 나타내는 정보 모델 7번이 존재한다. 만일 호스트 1에 3번 애플리케이션이 새롭게 추가되면서 기존의 1번 애플리케이션과 2번 애플리케이션 간에 존재하던 의존성 관계가 없어지고, 새로운 애플리케이션과 1번 애플리케이션 간에 의존 관계가 형성되고, 또한 2번 애플리케이션과도 의존 관계가 형성된다면, 이러한 의존성의 변화는 통합 관리/모니터링 시스템 내부의 정보 모델에도 반영되어야 한다. 3번 애플리케이션이 추가됨으로 인해 애플리케이션 정보 모델 6번이 생성되고 정보 모델 4번과 5번 사이에 존재하던 의존성은 정보 모델 4번과 6번 사이의 의존성 정보와 정보 모델 6번과 5번 사이의 의존성 정보로 변경되어야 한다. 이 과정에서 기존의 의존성 정보를 유지하던 7번 정보 모델과 연관된 4번과 5번 애플리케이션 정보 모델과 의존성 변경 중에 발생할 수 있는 에러를 방지하기 위해 실제 관리 대상인 1번 애플리케이션과 2번 애플리케이션에 잠시 실행을 중단할 것을 통보하고, 기존의 7번 의존성을 제거한다. 그리고 6번 애플리케이션 정보 모델과 갖게 되는 새로운 의존성 8번과 9번을 추가하여 의존성에 대한 정보를 수정하고, 1번과 2번 애플리케이션에 의존성이 변경되었음을 통보한다. 계속해서 정지되어 있던 1번, 2번 애플리케이션을 재실행시킨다. 이러한 과정이 새로운 애플리케이션 3번이 추가되는 시나리오이다.

위에서 설명한 애플리케이션 추가에 대한 절차는 다음과 같다.

Algorithm - Add Application
In : hardware h, application a,
set of added dependancy Dep_a
out : void

```
1   function AddApplication(h, a, Dep_a) {
2       A_stop = ∅;
3     for each (a_i, a_j)∈Dep do {
4       if (a = a_i) {
5         stopRunning(a_j);
6         A_stop = A_stop ∪ a_j;
7       }
8       if (a = a_j) {
9         stopRunning(a_i);
10        A_stop = A_stop ∪ a_i;
11      }
12    }
13    D_a = (h,a);
14    IM = IM ∪ (∅, {a}, D_a, Dep_a);
15    Notify(A_stop);
16    startRunning(A_stop);
17  }
```

- 14번째 줄의 $IM \cup (\varnothing, \{a\}, D_a, Dep_a)$에서 공집합은 하드웨어 정보를 뜻한다. 애플리케이션의 추가나 삭제, 정보 변경에서는 하드웨어의 정보가 따로 필요

하지 않기 때문이다.

- 15번째 줄의 Notify() 함수는 의존성 변경이 발생된 노드에 변경 가능성을 알려주는 함수이다.

4.3.3.4 애플리케이션 삭제

애플리케이션의 삭제도 애플리케이션의 추가와 마찬가지로 하드웨어 내부의 애플리케이션 형상이 변경될 때 이루어지는 작업이다. 애플리케이션이 삭제되면서 함께 의존성 있는 다른 애플리케이션에 영향을 줄 수 있기 때문에, 이것을 고려하여 애플리케이션을 삭제한다. 애플리케이션이 삭제될 때는 애플리케이션의 이름을 전송하여 해당 애플리케이션의 배치정보를 제거할 수 있도록 한다.

애플리케이션 삭제에 대한 절차는 다음과 같다.

Algorithm - Remove Application
In : hardware h, application a
Out : void

```
1  function RemoveApplication(h, a) {
2      stopRunning(a);
3      A_stop = ∅; Dep_rem = ∅;
4      Dep = IM.getDependencySet();
5      for each (a_i, a_j) ∈ Dep do {
6          if (a = a_i) {
7              stopRunning(a_j);
8              A_stop = A_stop ∪ a_j;
9              Dep_rem = Dep_rem ∪ (a, a_j);
10         }
11         if (a = a_j) {
12             stopRunning(a_i);
13             A_stop = A_stop ∪ a_i;
14             Dep_rem = Dep_rem ∪ (a_i, a);
15         }
16     }
17     D_a = (h,a);
18     for each (h_k, a_i) ∈ D do { //h_k ≠ h
19             if (a = a_i) {
20                 D_a = ∅; break;
21             }
22     }
23
24     IM = IM− (∅, {a}, D_a, Dep_rem);
25     Notify(A_stop);
26     startRunning(A_stop);
27 }
```

- 17~22번째 줄은 애플리케이션이 삭제될 때, 애플리케이션이 속한 배치정보를 삭제나 유지하기 위한 코드이다. 이는 삭제되는 애플리케이션과 동일한 애플리케이션이 다른 하드웨어에도 존재할 수 있어 배치정보를 공유할 수 있기 때문이다. 삭제되는 애플리케이션이 다른 하드웨어와 배치정보를 공유하지 않는다면 바로 삭제해도 문제가 발생하지 않지만, 다른 하드웨어와 배치정보를 공유하고 있다면 배치정보가 삭제되어서는 안 되기 때문이다.

4.3.3.5 애플리케이션 정보 변경

애플리케이션의 정보가 변경되면 애플리케이션 간의 의존성 정보가 변경될 수도 있다. 따라서 애플리케이션의 정보가 변경되었을 때에도 내부 의존성 정보의 변경이 발생되었는지를 확인하고, 이를 반영하여 정보 모델을 재구성 한다.

애플리케이션의 정보 변경에 대한 절차는 다음과 같다.

Algorithm - Change Application
In : application a, set of changing dependancy Dep_{ch}
Out : void

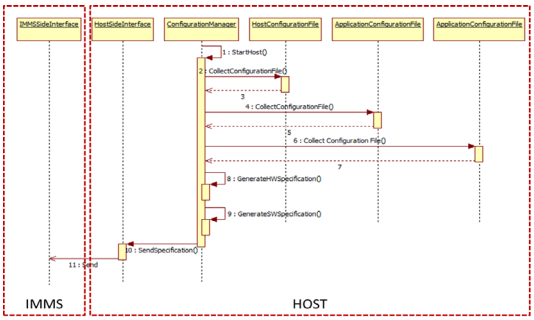
```
1  function ChangeApplication(a, Dep_ch) {
2      stopRunning(a);
3      A_stop = ∅; Dep_rem = ∅;
4      Dep = IM.getDependencySet();
5      for each (a_i, a_j) ∈ Dep do {
6          if (a = a_i) {
7              Dep_rem = Dep_rem ∪ (a, a_j);
8          }
9          if (a = a_j) {
10             Dep_rem = Dep_rem ∪ (a_i, a);
11         }
12     }
13     for each (a_i, a_j) ∈ Dep_ch do {
14         if (a = a_i) {
15             stopRunning(a_j);
16             A_stop = A_stop ∪ a_j;
17         }
18         if (a = a_j) {
19             stopRunning(a_i);
20             A_stop = A_stop ∪ a_i;
21         }
22     }
```

23	$IM = IM - (\varnothing, \{a\}, \varnothing, Dep_{rem})$;
24	$IM = IM \cup (\varnothing, \{a\}, \varnothing, Dep_{ch})$;
25	Notify(A_{stop});
26	startRunning(A_{stop});
27	}

4.3.4 정보 모델 기반 동적 재구성 시나리오

정보 모델의 동적 재구성을 위해 앞에서 기술한 하드웨어와 애플리케이션의 정보 및 추가/삭제 프로시저들을 이용한다. 이를 토대로 시스템의 변경된 형상을 반영할 수 있다.

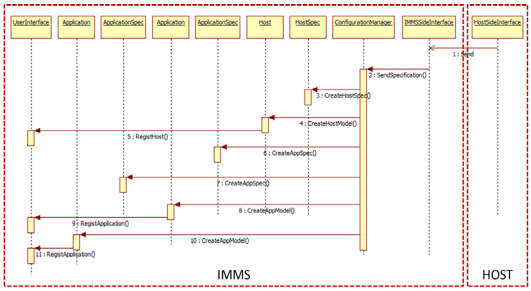
호스트 측에서의 동적 재구성을 위한 시나리오는 (그림 5)와 같다.



(그림 5) 호스트 측 정보 모델 동적 재구성 시나리오
(Figure 5) Scenario for Dynamic Reconfiguration on the Host side

① 호스트 관리자, 애플리케이션 관리자는 수정된 하드웨어 및 애플리케이션의 정보를 반영하여 형상 정보 파일을 수정한다.
② 수정된 정보를 반영하기 위해 호스트의 Configuration Manager를 호출한다.
③ Configuration Manager는 변경된 하드웨어와 애플리케이션의 형상 정보 파일을 수집한다.
④ 수집된 하드웨어와 애플리케이션의 형상 정보 파일을 바탕으로 XML 명세 정보를 생성한다.
⑤ 생성된 XML 명세 정보를 통합 관리/모니터링 시스템으로 전송한다.

통합 관리/모니터링 시스템 측에서의 동적 재구성을 위한 시나리오는 (그림 6)과 같다.



(그림 6) 통합 관리/모니터링 시스템 측 정보 모델 동적 재구성 시나리오
(Figure 6) Scenario for Dynamic Reconfiguration on the Integrated Management/Monitoring System side

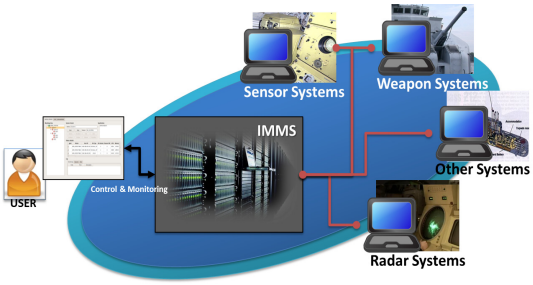
① 통합 관리/모니터링 시스템의 커뮤니케이션 인터페이스에서는 호스트 측에서 전송된 XML 명세 정보를 수신한다.
② 수신된 변경된 형상 정보 파일은 통합 관리/모니터링 시스템 측 Configuration Manager로 전달하여 형상 정보를 변경할 수 있도록 한다.
③ Configuration Manager에서는 변경된 형상 정보 파일을 바탕으로 하드웨어와 애플리케이션 명세를 생성하거나 수정한다.
④ 생성 또는 수정된 명세를 바탕으로 명세와 1:1로 대응되는 정보 모델을 생성한다.
⑤ 생성된 정보 모델을 사용자 인터페이스에 통보하여 사용자에게 정보를 제공할 수 있도록 한다.

5. 사례연구

본 논문에서 소개하는 통합 관리 및 모니터링 시스템에서 정보 모델의 동적 재구성을 위해 함정 전투 시스템에 적용시켜보았다.

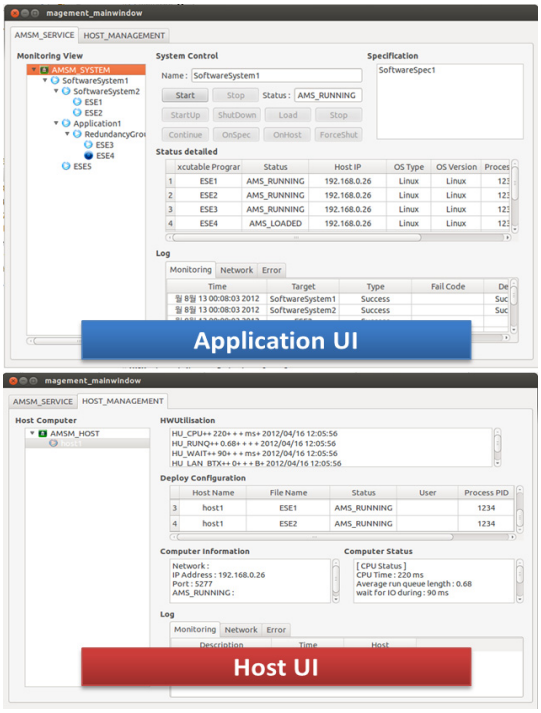
(그림 7)은 실제 함정 전투 시스템에 본 논문의 통합 관리/모니터링 시스템을 적용한 예이다. 함정 전투 시스템은 레이더, 센서, 무기체계, 소나 등 다양한 이기종의 하위 시스템으로 구성된다. 새로운 무기체계나 시스템이 개발되면 주기적으로 도입이 이루어지기 때문에 하위 시스템의 구성이 변경될 수 있다. 본 논문에서 제시하는 통합 관리/모니터링 시스템을 적용하여 확인 및 검증을 수행하기에 최적의 조건을 갖추고 있다.

(그림 8)은 함정 전투 시스템에서 애플리케이션의 형



(그림 7) 함정 전투 시스템에 적용된 통합 관리/모니터링 시스템

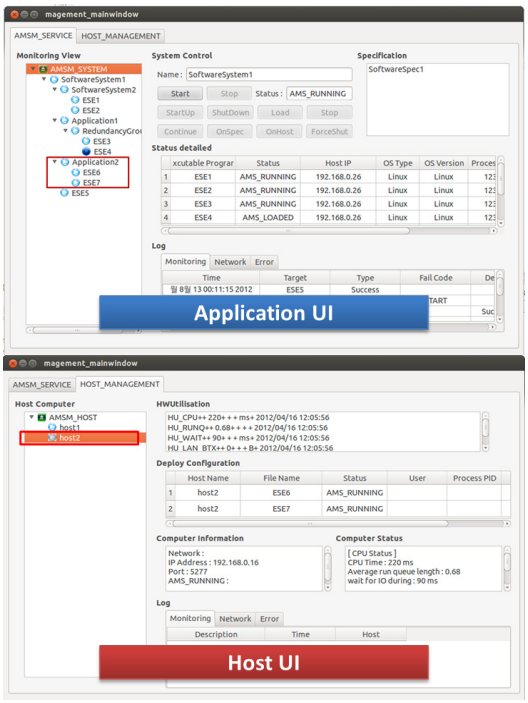
(Figure 7) Integrated Management/Monitoring System on the Naval Ship Combat System



(그림 8) 새로운 애플리케이션 및 하드웨어 추가 전 모니터링 정보

(Figure 8) Monitoring Situation before adding New Application

상 정보를 볼 수 있는 사용자 인터페이스의 스냅 샷이다. 사용자 인터페이스를 통하여 최초로 구성된 애플리케이션의 형상과 상태 정보를 알 수 있다.



(그림 9) 새로운 애플리케이션 및 하드웨어 추가 후 모니터링 정보

(Figure 9) Monitoring Situation after adding New Application

(그림 9)는 새로운 하드웨어와 애플리케이션이 추가되었을 때, 내부적으로 정보 모델을 재구성하고 그 결과가 사용자 인터페이스에 나타난 상황이다. 기존에 존재하지 않은 애플리케이션 정보가 다른 애플리케이션의 모니터링에 영향을 끼치지 않고 추가된 것을 확인할 수 있으며, 이들의 상태를 즉각적으로 모니터링한 결과를 UI를 통해 살펴 볼 수 있다.

6. 결 론

컴퓨터 시스템의 규모가 커지고 분산 컴퓨팅 환경으로 전환되어 감에 따라 전체 시스템을 관리하고 모니터링 할 수 있는 시스템에 대한 요구가 늘어가고 있다. 이로 인해 분산 시스템을 통합 관리하고 모니터링 할 수 있는 많은 연구가 진행되고 있으나, 실제 시스템의 구조와 통합 관리/모니터링 시스템 간의 동기화 문제를 해결하는데 많은 어려움을 겪고 있다.

실제로 본 논문의 이전 연구에서는 관리 대상 시스템의 형상이 변경되면 이를 반영하기 위해 형상 파일을 새로 작성하고, 작성된 내용을 통합 관리/모니터링 시스템에 적용하기 위해서 모든 정보 모델들을 재구축 하였다. 그렇게 하다 보니 동기화 동안에 관리 대상 시스템으로부터 수집되는 모니터링 정보를 사용자에게 제공하지 못한다는 치명적인 단점이 존재하였다. 이를 개선하기 위하여 본 연구에서는 통합 관리/모니터링 시스템이 중단 없이 서비스를 제공하면서 아울러 변경된 형상들에 대한 동기화가 가능하도록 정보 모델에 대한 동적 재구성 방법을 제시하였다. 이를 통해 변경된 시스템의 형상을 즉각적으로 반영할 수 있게 되었다. 본 논문의 통합 관리/모니터링 시스템을 통해서 다양한 분산 시스템을 통합 관리하고 모니터링 하는데 많은 도움이 될 것으로 확신한다.

참 고 문 헌(Reference)

[1] M.L. Massie, B.N. Chun, D.E. Culler, “The ganglia distributed monitoring system: design, implementation, and experience”, Parallel Computing, 30(7), pp.817-840, 2004

[2] B.K Min, S.H Kuk, Y.J Seo, D.G Mun, S.P Yun, H.S Kim, C.S Kim, “Standard-based Integrated Management System Implementation for Heterogeneous Distributed System in Naval Ship Combat Management System”, 11th Int‘l Conf. on Computer and Information Science, pp.535-540, 2012

[3] M. Wegdam “Dynamic Reconfiguration and Load Distribution in Component Middleware”, Ph.D. Thesis, University of Twente, the Netherlands, 2003

[4] J.P.A. Almeida, M. Sinderen, L.F. Pires, M. Wegdam, “Platform-independent Dynamic Reconfiguration of Distributed Applications”, 10th Int‘l Workshop on Future Trends in Distributed Computing Systems, pp.286-291, 2004

[5] A. Janik, K. Zielinski, “AAOP-based dynamically reconfigurable monitoring system”, Information and Software Technology, Vol. 52, Issue. 4, pp.380-396, 2010

[6] J. Kramer, J. Magee, “The evolving philosophers’ problem: dynamic change management”, IEEE Tran. on Software Engineering, Vol.16, No.11, pp.1293-1306, 1990

[7] V. Quema, R. Lachaize, E. Cecchet, “An asynchronous middleware for Grid resource monitoring”, Concurrency and Computation: Practice and Experience, Vol. 16, Issue. 5, pp.523-534, 2004

◐ 저 자 소 개 ◑

민 법 기

2009년 공주대학교 산업정보학과 졸업(학사)
2012년 충남대학교 대학원 컴퓨터공학과 졸업(석사)
2012년~현재 충남대학교 대학원 컴퓨터공학과 박사재학
관심분야 : 소프트웨어공학, 소프트웨어 테스팅, 소프트웨어 아키텍처, 스마트폰
E-mail : bkmin@cnu.ac.kr

서 용 진

2011년 충남대학교 컴퓨터공학과 졸업(학사)
2011년~현재 충남대학교 대학원 컴퓨터공학과 석사재학
관심분야 : 소프트웨어 테스팅, 소프트웨어 품질관리, 스마트폰, UX/UI
E-mail : yjseo082@cnu.ac.kr

◐ 저 자 소 개 ◑

김 현 수

1988년 서울대학교 계산통계학과 졸업(학사)
1991년 한국과학기술원 전산학과 졸업(공학석사)
1995년 한국과학기술원 전산학과 졸업(공학박사)
1995년~1995년 한국전자통신연구원 Post Doc.
1996년~2001년 금오공과대학 조교수
1999년~2000년 Colorado State University 방문 연구 교수
2007년~2008년 Purdue University 방문 연구 교수
2001년~현재 충남대학교 컴퓨터공학과 교수
관심분야 : 소프트웨어공학, 소프트웨어 테스팅, SOA
E-mail : hskim401@cnu.ac.kr

국 승 학

2004년 충남대학교 컴퓨터과학과 졸업(학사)
2006년 충남대학교 대학원 컴퓨터공학과 졸업(석사)
2012년 충남대학교 대학원 컴퓨터공학과 졸업(박사)
2012년~현재 국방과학연구소
관심분야 : 소프트웨어공학, 소프트웨어 테스팅, 소프트웨어 아키텍처, 소프트웨어 품질 etc.
E-mail : shkuk@add.re.kr

정 용 환

2000년 경북대학교 전자전기공학과 졸업(학사)
2006년 한국과학기술원 대학원 전기전자공학과 졸업(석사)
2006년~현재 국방과학연구소 선임연구원
관심분야 : 신호처리, 데이터베이스
E-mail : jungyh06@add.re.kr

김 점 수

1996년 인하대학교 전자계산공학과 졸업(학사)
1998년 광주과학기술원 대학원 정보통신공학과 졸업(석사)
1998년~현재 국방과학연구소 선임연구원
관심분야 : 분산시스템, 데이터베이스
E-mail : chskim@add.re.kr