JINI 기반 원격 응용 모니터링 시스템
(A Remote Applications Monitoring System using JINI)

임 성 훈 † 송 무 찬 † 김 정 선 ††
(Seonghun Im) (Moochan Song) (Jungsun Kim)

요 약 원격 모니터링 시스템이란 네트웍에 분산된 호스트 혹은 원격 어플리케이션의 상태를 실시간으로 모니터링 하는 시스템이다. 네트웍에 분산된 호스트들은 다양한 플랫폼을 갖추고 있는 것이 현실이다. 하지만, 기존의 모니터링 시스템들은 플랫폼에 의존적인 성격을 띠고 있다. 네트웍을 통해서 모니터링을 하기 때문에 잠재적인 네트웍의 문제발생에 대한 적절한 대응이 필요하지만, 기존의 모니터링 시스템들은 네트웍의 문제에 대해서 적절하게 대응을 못하고 있다. 그리고 호스트의 상태를 변화시키는 요인은 호스트의 시스템 자원보다는 어플리케이션의 상태변화가 주 요인이 되고 있지만, 기존의 시스템들은 주로 호스트의 상태정보만 모니터링을 하고 있다. 따라서, 분산된 호스트들의 플랫폼에 독립적이고, 네트웍의 잠재적인 문제발생을 적절하게 대응을 할 수 있고, 호스트의 상태정보 보다는 호스트의 어플리케이션의 상태정보를 모니터링 하는 시스템이 필요한 실정이다.

본 논문에서는 네트웍에 분산된 어플리케이션의 실시간 모니터링을 통해서 어플리케이션 상태를 효과적으로 관리할 수 있는 모니터링 시스템인 RAMS (Remote Applications Monitoring System)의 설계 및 구현을 제시한다. 이 시스템은 네트웍에 분산된 호스트들의 효과적인 관리를 위해 호스트 어플리케이션의 상태를 모니터링 한다. RAMS는 모니터링 대상인 호스트 어플리케이션의 모니터링을 담당하는 Agent, 호스트의 등록, 관리를 담당하는 Manager로 구성된다. Manager와 Agent는 네트웍 단절 혹은 호스트 failure의 상황에도 자동으로 복구가 가능할 수 있도록 JINI를 최대한 활용함으로써 다른 시스템에 비해 시스템의 구성 및 관리가 용이한 특성을 갖는다.

키워드 : 원격 모니터링 시스템, 지니, 어플리케이션 모니터링, 실시간

Abstract In general, remote monitoring systems monitor the status of distributed hosts and/or applications in real-time for diverse managerial purposes. However, most of the extant systems have a few undesirable problems. First of all, they are platform-dependent and are not resilient to network and/or host failures. Moreover, they normally focus on the resource usage trends in monitored hosts, rather than on the status change of the applications running on them. We strongly believe that the latter has more direct and profound effect on the resource usage patterns on each host.

In this paper, we present the design and implementation of the Remote Applications Monitoring System (RAMS) that enables us to effectively manage distributed applications through a real-time monitoring of their respective resource usages. The RAMS is a centralized system that consists of many distributed agents and a single centralized manager. An agent on each host is in charge of collecting and reporting the status of local applications. The manager handles agent registration and provides a central access point to the selection and monitoring of distributed applications. The salient features of the system include robustness and portability. The adoption of JINI greatly facilitates an automatic recovery from partial network failure and host failure.

Key words : Remote Monitoring System, JINI, Application Monitoring, Real-Time

† 비 회 원 : 한양대학교 컴퓨터공학과
shim@cse.hanyang.ac.kr
mcsong@cse.hanyang.ac.kr
†† 종신회원 : 한양대학교 컴퓨터공학과 교수
jskim@cse.hanyang.ac.kr
논문접수 : 2003년 10월 27일
심사완료 : 2004년 3월 12일

1. 서 론

컴퓨터 및 네트웍 기술의 발전에 따라 최근 들어 네트웍을 통해서 제공하는 서비스(예, 웹 서비스)들은 양적, 질적으로 거대하게 증가하였으며, 서비스를 제공하는 호스트들은 네트웍 상에 분산되어 있다. 그 결과 분산된 호스트들의 상태를 관리자가 로컬에서 체크 및 관

리하기에는 인적자원 및 비용이 상당히 필요하게 되었다. 따라서, 분산되어 있는 호스트들을 관리하기 위해서는 원격으로 호스트의 자원 및 상태를 모니터링 할 수 있는 시스템이 요구가 되어 왔고, 기존의 여러 시스템들이 존재하고 있다. 그러나, 기존의 원격 모니터링 시스템들의 문제점을 살펴보면, 첫째, 호스트 플랫폼에 의존적이기 때문에 이 기종 시스템일 경우 시스템 확장 및 재구성에 어려운 문제점이 있다. 둘째, 네트웍 장애 등의 잠재적인 문제를 적절하게 대응하기 어렵다. 셋째, 어플리케이션의 상태정보 보다는 호스트의 자원 및 상태정보에 대해서 더 중점을 두고 있기 때문에, 부하가 큰 호스트의 어떤 어플리케이션을 이동(Migration)시켜야 하는지 판단하기가 어렵다. 고객에게 안정된 성능의 서비스를 일관되게 제공하기 위해서는 각 호스트에서 실행중인 하나 이상의 서버프로세스 가운데 어느 것(들)이 호스트의 자원을 과도하게 사용하고 있는 지를 파악하여 다른 호스트로 이주 시키는 따위의 관리 작업이 적시에 수행될 필요가 있다. 그러나 이러한 작업은 단순히 호스트 상태의 모니터링 만으로는 불가능하게 된다.

따라서, 본 논문에서는 기존 원격 모니터링 시스템들의 단점을 해결하기 위한 해결책으로, JINI를 기반으로 한 분산환경에서 네트웍에 분산된 호스트들의 어플리케이션 상태정보를 실시간으로 모니터링을 해서, 호스트의 상태를 효과적으로 관리할 수 있는 모니터링 시스템인 RAMS(Remote Applications Monitoring System)에 대한 설계 및 구현을 제시하였다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구 및 기술동향에 대해서 살펴보고, 3장에서는 RAMS의 특징 및 구성에 대해서 살펴보고, 4장에서는 RAMS의 설계 및 구현, 마지막으로 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 관련 연구

2.1 원격 모니터링 시스템(Remote Monitoring System)

원격 모니터링 시스템이란, 분산환경에서 원격 호스트의 자원들을 모니터링 하는 시스템을 말한다. 기존의 시스템에 대해서 살펴보면, 첫째, MRTG-PME 시스템[1]은 SNMP(Simple Network Management Protocol)를 이용한 라우터 트래픽을 모니터링 하기 위한 MRTG라는 패키지를 확장한 시스템이다. Unix/Linux기반으로 FreeBSD, MacOS 그리고 Solaris의 환경에서 원격에 있는 호스트들의 자원들을 모니터링 할 수 있는 시스템이다.

둘째, Sinfo 시스템[2]은 네트웍에 있는 모든 호스트에 브로드캐스트를 통해서 각 호스트의 상태를 모니터링 하는 시스템이다. 모니터링 대상은 CPU, 메모리 사용량, 네트웍 사용량, 그리고 상위 5 프로세스에 대한 정보이다. Sinfo 시스템은 Linux 2.2.X/2.4.X, Solaris 2.8, FreeBSD 4.4의 환경에서만 실행이 가능하다.

셋째, Zabbix 시스템[3]은 SNMP를 통해서 원격 호스트의 자원과 호스트가 제공하는 서비스를 모니터링 하는 시스템이다. 모니터링 대상은 원격 호스트에서 서비스하는 프로토콜인, SMTP, IMAP, POP3, HTTP, SSHD등과 호스트 자원인 CPU, 디스크, 메모리등이다. Zabbix 시스템은 Unix/Linux 기반으로, OpenBSD, FreeBSD, HP-UX, AIX, Solaris, MacOS X, SCO Open Server등의 환경에서 사용이 가능하다.

넷째, Remote Task Manager 시스템[4]은 Windows NT/2000 환경을 기반으로 하는 원격 모니터링 시스템이다. Remote Task Manager 시스템의 모니터링 자원은 원격 호스트에서 사용자가 사용 하고 있는 어플리케이션의 상태, 실행되고 있는 프로세스와 각 프로세스에 할당되어 있는 메모리의 양, 원격 호스트에서 제공하고 있는 서비스들, CPU와 메모리등의 사용량이다.

위에서 살펴본 MRTG-PME, Sinfo, Zabbix, Remote Task Manager 시스템들이 가지고 있는 단점으로는 첫째, 플랫폼에 의존적이기 때문에, 이 기종 플랫폼 호스트의 자원 및 상태에 대한 모니터링이 불가능하다. 둘째, 주 모니터링 대상이 호스트의 자원을 사용하고 있는 어플리케이션의 상태변화가 아니라 호스트의 전체적인 자원 및 상태변화를 주 모니터링 대상으로 하고 있다.

한편, 최근 들어 BEA사의 Tuxedo TP 모니터 시스템[5]은 RAMS 시스템과 유사하게 어플리케이션 모니터링을 제공하는 것으로 알려져 있으나, 플랫폼 독립적이지 못하고, 시스템 장애나 확장시 RAMS와 같은 유연성을 제공하지 못한다.

2.2 JINI Technology

JINI 기술은 썬 마이크로시스템즈에서 임베디드 시스템들을 네트웍에 연동할 수 있을 뿐만 아니라, 다양한 종류의 네트웍 자원에 대해 서로 연동할 수 있도록 만든 기술로서 자바 기반 위에 플러그 앤 플레이(Plug & Play : PnP) 및 동적 서비스 환경을 제공하는 진보된 기술이며 미래 네트웍 구조에 대한 방향을 제시하고 있다. JINI 기술은 여러 종류의 하드웨어와 소프트웨어 플랫폼에서 수행되도록 설계되었으며, 그 기능을 수행하기 위해서는 자바를 필요로 한다.

그림 1에서 보듯이, JINI 서비스는 자바를 기반으로 하기 때문에, 운영체제나 기타 하드웨어 플랫폼에 무관하게 동작하게 된다. JINI에서는 Discovery, Join, Lookup 프로토콜을 이용하여 상호 통신이 이루어지며 서비스를 제공하는 부분과 서비스를 사용하는 부분의 상호작용은 자바 RMI(Remote Method Invocation)를

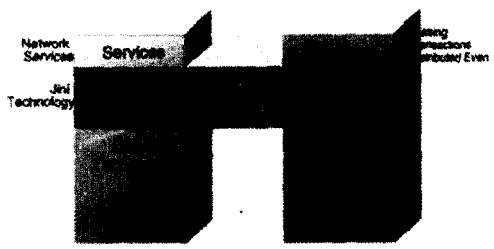


그림 1 JINI Technology Architecture

이용한다. JINI 시스템의 수행과정을 살펴보면 첫째, 서비스 제공하는 부분은 Discovery 프로토콜을 이용하여 서비스를 관리하는 관리자를 찾고, 둘째, Join 프로토콜을 이용하여 서비스 관리자에게 서비스를 등록하며, 셋째, 서비스를 이용하는 부분은 Discovery 프로토콜을 이용하여 서비스 관리자를 찾고 Lookup 프로토콜을 통해 자신이 원하는 서비스를 검색하여, 부합하는 서비스를 사용하게 된다[6,7].

2.3 JNI(Java Native Interface)

자바는 Windows, Unix/ Linux등과 같은 운영체제의 중간단계에서 자바를 해석하고 실행하는 자바 가상 머신(JVM: Java Virtual Machine)을 위치시킴으로써 시스템에 독립적으로 자바를 사용할 수 있다. 하지만, "자바 가상머신이 제공하지 못하는 기능(하드웨어 디바이스를 직접 핸들링 하거나 운영체제의 의존적인 기능의 제어등)을 사용해야 할 필요성이 있을 때는 어떻게 해야 할까?"에 대한 해결책이 바로 JNI(Java Native Interface)이다.

JNI는 C나 C++ 혹은 어셈블리 언어와 같이, 자바 이외의 언어로 작성된 프로그램을 자바 가상 머신 위에서 실행할 수 있도록 인터페이스를 제공해 준다. 따라서 자바는 JNI를 이용해서 필요한 부분만 C나 C++의 언어로 구현해서 사용 할 수도 있다. 뿐만 아니라 C나 C++로 만든 어플리케이션이 자바의 객체를 생성하고 메쏘드를 호출하는 것 역시 가능하다. 즉, JNI를 잘 사용하게 된다면, 자바의 장점에 다른 언어가 가지고 있는 장점을 동시에 사용할 수 있다[8-10].

그림 2에서 보듯이, JNI는 C 언어 부분에서 미리 만들어진 라이브러리들과 함수를 제공하고, 자바 부분에서

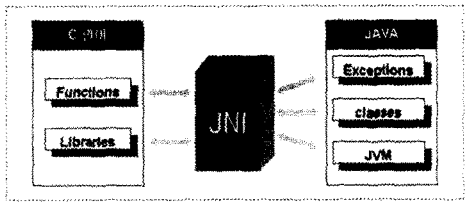


그림 2 자바와 C언어 사이에서 인터페이스 역할의 JNI

도 미리 만들어진 클래스와 예외처리 그리고 가상 머신의 특성 등을 제공하면 된다.

3. RAMS 설계

그림 3에서 보듯이, 분산환경에서의 RAMS는 하나의 Manager와 다수의 Agent간에 통신을 하는 Client/ Server 형태로, 기본적으로 Manager는Server, Agent는 Client의 역할을 한다.

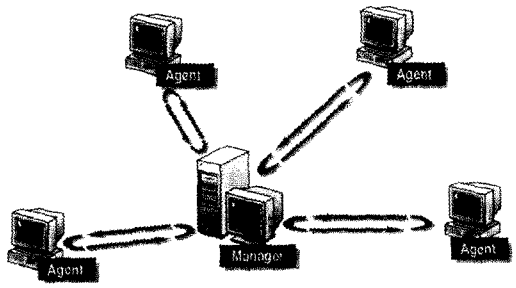


그림 3 분산환경에서의 RAMS의 구조

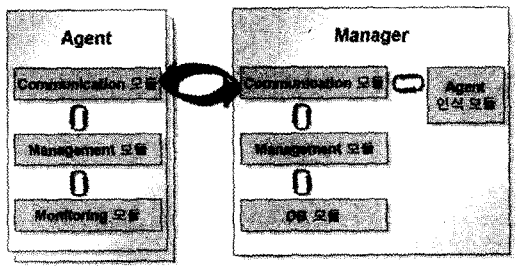


그림 4 RAMS의 구성도

그림 4에서 보듯이, Agent는 Manager와 통신을 담당하는 Communication 모듈, Agent를 전체적으로 관리하는 Management 모듈 그리고 자신의 시스템 자원을 모니터링하는 Monitoring 모듈로 구성이 되어 있다.

Manager는 Agent들을 인식하는 Agent 인식모듈, Agent와 통신을 담당하는 Communication 모듈, Manager를 전체적으로 관리하는 Management 모듈과 Agent 리스트와 각 Agent에 설정된 어플리케이션 리스트 및 설정 데이타와 모니터링 데이타를 저장하는 DB 모듈로 구성이 되어 있다.

3.1 Agent 설계

3.1.1 Communication 모듈

그림 5는 Manager와의 통신을 담당하는 Agent의 Communication 모듈의 class diagram으로, 각 클래스에 대해 살펴보겠다.

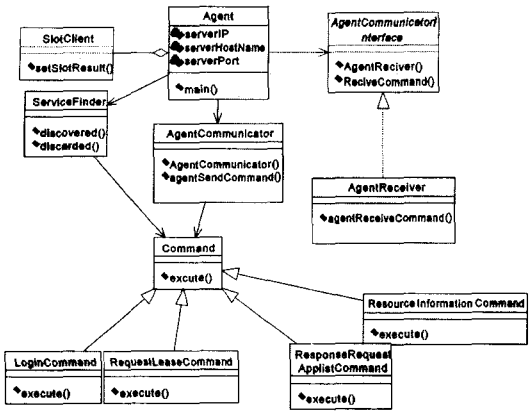


그림 5 Agent Communication 모듈의 Class Diagram

- Agent 클래스 : Agent의 시작 클래스이며, ServiceFinder, AgentCommunicator, AgentReciver 객체를 초기화하는 역할을 한다.
- ServiceFinder 클래스 : Manager에 있는 JINI 룩업 서비스를 발견하기 위한 기능을 제공하는 클래스이다.

```
publicclassServiceFinderimplementsDiscoveryListener{
//groups,attributes와serviceInterface인class를이용,실제서비스를
찾는다
public ServiceFinder(String[] groups, Class serviceInterface,
Entry[]attributes)throwsIOException{
//lookup서비스에서매치되는템플릿을생성,서비스발견
Class[]name=newClass[]{serviceInterface};
template=newServiceTemplate(null,name,attributes);
//모든lookup서비스에대한multicastdiscovery를수행하기위한기능
생성.
reg=newLookupDiscovery(groups);
reg.addDiscoveryListener(this);
}
publicsynchronizedvoiddiscovered(DiscoveryEventdev){
ServiceRegistrar[]lookup=dev.getRegistrars();
```

그림 6 Manager의 룩업 서비스를 찾기 위한 클래스

그림 6은 Agent가 Manager를 찾기 위해서 룩업 서비스에 매치되는 템플릿을 생성하고, 실제 서비스가 발견되면 템플릿을 사용하고, 모든 룩업 서비스에 대해서, 멀티캐스트(Multicast) Discovery를 수행하기 위해서 DiscoveryListener에 등록을 하고 있다.

DiscoveryListener에 등록을 하고, 룩업 서비스가 발견되면 자동으로 discovered() 메쏘드가 수행이 되어서 서비스 객체를 받아 오게 된다.

- AgentCommunicator 클래스 : Agent가 Manager에게 여러 가지 요청을 전송하는 클래스이다.
- AgentCommunicatorInterface 클래스 : Agent의 RMI 객체에 대한 인터페이스를 정의하는 클래스이다. Manager가 Agent에게 데이타 전송 및 호출을 위해 사용한다. AgentReceiver 클래스가 AgentCommunicatorInterface를 구현하고 있다.
- AgentReceiver 클래스 : Manager로부터 요청을 전송받아 그 요청에 대한 실행을 Command 클래스를 상속받고 있는 적당한 하위 Command 클래스들을 선택을 해서 요청을 수행하도록 하는 역할을 한다. 그리고 Agent-CommunicatorInterface에서 정의한 인터페이스를 구현하였다.
- Command 클래스 : Manager와 Agent간 메시지 송/수신을 위한 상위 클래스로써, Design Pattern중에서 Command Pattern[11]을 사용하였다. Command Pattern은 요청을 객체화해서 요청의 수행을 빠르게 해 주며, 요청의 응답에 대한 투명성(어떤 클래스가 요청에 대해서 응답을 하는지 몰라도 되는 성질)을 제공하며, 각 요청의 확장에 대한 용이성을 높여준다. LoginCommand 클래스는 Agent가 Manager에게 접속하는 역할을 하고, RequestLeaseCommand 클래스는 Agent가 Manager에게 리징(Leasing)을 하는 역할을 하고, ResponseRequest-Command 클래스는 Manager에서 선택한 어플리케이션의 리소스 정보를 서버에게 전달하기 역할을 하고, ResourceInformationCommand 클래스는 Manager에서 관심있는 어플리케이션의 리스트를 Manager에게 전달하는 역할을 하는 클래스이다.
- SlotClient 클래스 : Agent가 Manager에게 리징을 요청하면, Manager의 리징 서비스는 요청한 Agent에게 리즈(Leasing 객체)를 주게 된다. 이 클래스는 리스 객체를 유지하고 있는 역할을 한다.

그림 7에서 보듯이, Agent는 ServiceFinder, AgentCommunicator, AgentReciver의 객체를 생성하고, ServiceFinder는 discovered() 메쏘드를 통해서 Manager의 룩업 서비스를 찾은 후 AgentCommunicator를 통해서 Agent 등록 요청인 Login-Command를 Manager에게 보내게 되고, Agent가 계속적으로 살아 있다는 메시지를 Manager에게 보내기 위해서 Manager의 리징 서비스를 사용하는데, 이 요청은 RequestLeaseCommand를 통해서 보내게 된다.

AgentReceiver는 Manager로부터 요청인 어플리케이션 리스트 요청에 대한 응답으로 ResponseRequest-Command, 원하는 어플리케이션의 모니터링 데이타 요청인 ResourceInformationCommand를 통해서 응답한다.

3.1.2 Management, Monitoring 모듈

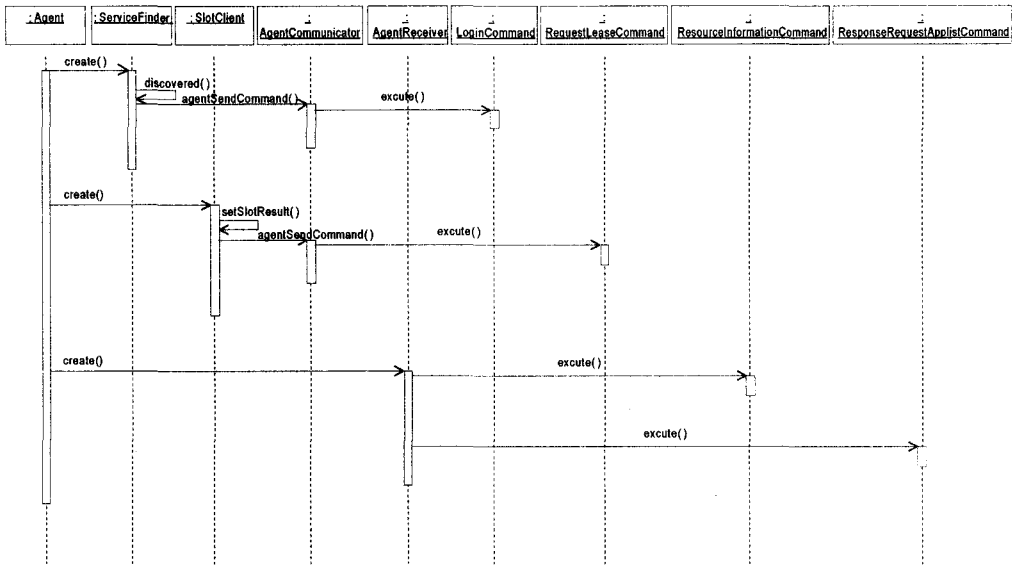


그림 7 Agent Communication 모듈의 Interaction Diagram

그림 8은 Agent의 Management 모듈과 Monitoring 모듈의 Class Diagram으로, Manager 클래스와 Collector 클래스로 구분을 해서 살펴보겠다.

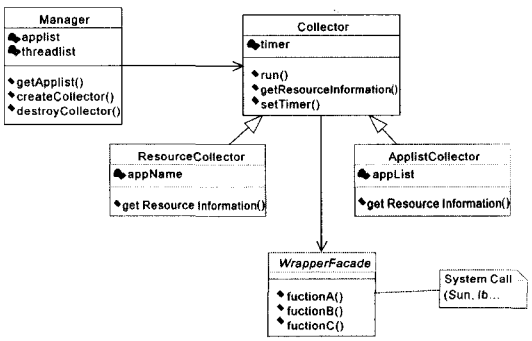


그림 8 Agent Management, Monitoring 모듈의 Class Diagram

- Manager 클래스 : 리소스를 모니터링하는 쓰레드를 관리하는 역할을 한다. 즉, 모니터링 요청이 들어오면, 모니터링을 수행하는 쓰레드를 생성하고, 모니터링 중지 요청이 들어오면, 생성된 쓰레드의 삭제하는 클래스이다.
- Collector 클래스 : Agent의 호스트 리소스인 CPU, 메모리, 각 어플리케이션당 CPU, 메모리 할당량등에 대해서 모니터링하는 쓰레드의 최상위 클래스이다. ResourceCollector 클래스는 Agent의 호스트에서 실행되고 있는 선택된 어플리케이션의 리소스를 수집하는 역할을 하는 클래스이다. ApplistCollector 클래스는 Agent의 호스트에서 실행되고 있는 어플리케이션의 리스트를 모니터링하는 역할을 하는 클래스이다. WrapperFacade는 JNI를 이용하여 Low-level의 함수들을 호출하는 메소드를 선언한 인터페이스이다. Wrapper Facade 패턴[12]은 객체 지향 클래스 인터페이스들을 통해서 Low-level의 함수들과 데이타 구조들을 캡슐화하기 위한 패턴이다.

그림 9에서 보듯이, Manager 클래스는 Collector 클래스를 생성하고 Collector 클래스를 통해서 Applist-Collector에게 getResourceInformation() 메쏘드를 호출하게 되면, ApplistCollector는 WrapperFacade 인터페이스에서 선언한 인터페이스를 통해서 getResource()메쏘드를 호출을 하게 되면, 호스트의 어플리케이션 리스트를 수집해서 수집된 데이타를 전송한다. 또, Collector가 ResourceCollector에게 getResource-Information() 메쏘드를 호출하게 되면, Resource-Collector는 WrapperFacade 인터페이스에서 선언한 인터페이스를 통해서 getResource() 메쏘드를 호출을 하게 되면, 호스트의 어플리케이션 리소스를 모니터링 하고 수집된 데이타를 전송한다.

3.2 Manager 설계

3.2.1 Communication 모듈

그림 10은 Manager가 Agent와의 통신을 담당하는 Manager의 Communication 모듈의 class diagram으로, 각 클래스에 대해 살펴보겠다.

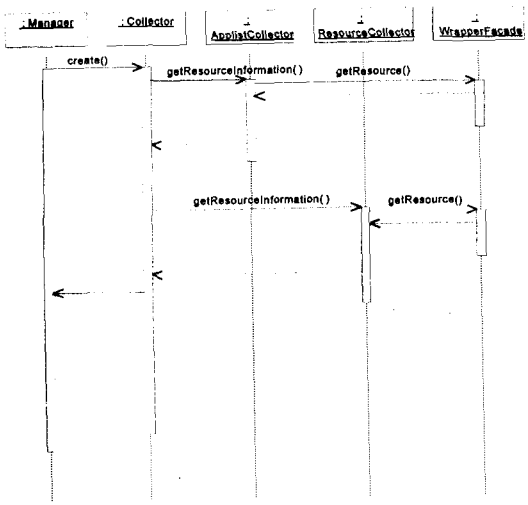


그림 9 Agent Management, Monitoring 모듈의 Interaction Diagram

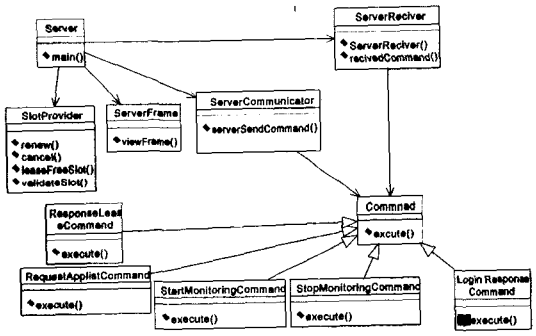


그림 10 Manager Communication 모듈 Class Diagram

- Server 클래스 : Manager의 시작 클래스이며, ServerCommunicator, ServerReciver, ServerFrame, SlotProvider의 객체를 초기화 하는 역할을 한다.
- ServerFrame 클래스 : Manager의 GUI(Graphical User Interface)를 보여주는 클래스이다.
- ServerCommunicator 클래스 : Manager가 Agent에게 어플리케이션 리스트,모니터링 요청등을 전송하는 클래스이다.
- ServerReceiver 클래스 : Agent로부터 요청을 전송받아 그 요청에 대한 실행을 Command 클래스를 상속받고 있는 적당한 하위 Command 클래스들을 선택을 해서 요청을 수행하도록 하는 역할을 한다.
- Command 클래스 : Manager와 Agent간 메시지 송/수신을 위한 상위 클래스로써, Design Pattern중에서 Command Pattern[11]을 사용하였다.

ResponseLeaseCommand는 Agent가 Manager에게 리징(Leasing)을 요청하면, Manager가 Agent를 리징하는 클래스이다. RequestApplistCommand는 Manager가 Agent에게 어플리케이션 리스트를 요청하는 클래스이다. StartMonitoringCommand는 Manager가 Agent에게 선택된 어플리케이션의 모니터링을 요청하는 클래스이다. StopMonitoringCommand는 Manager가 Agent에게 선택된 어플리케이션 모니터링의 중단을 요청하는 클래스이다. LoginResponseCommand는 Agent가 Manager에게 접속하면, 접속에 대한 응답을 Agent에게 보내는 클래스이다.

그림 11에서 보듯이, Server는 ServerFrame, ServerCommunicator, ServerReciver, SlotProvider를 생성한다. ServerFrame은 viewFrame() 메쏘드를 통해서 자신의 GUI 화면을 보여줄 수 있다. ServerCommunicator는 RequestApplistCommand, StartMonitoringCommand, Sto-pMonitoringCommand에게 execute() 메쏘드를 호출한다. ServerReciver는 LoginResponseCommand, Response-LeaseCommand의 execute() 메쏘드를 호출한다.

ResponseLeaseCommand는 SlotProvider에게 renew() 메쏘드를 호출한다. SlotProvider는renew()를 호출한 Agent에 대해서 validSlot() 메쏘드를 통해서 리징을 해주게 된다.

3.2.2 Management, DB 모듈

그림 12는 Manager의 Management 모듈과 DB 모듈로 각 클래스에 대해서 살펴보겠다.

- ServerInfo 클래스 : 원격 호스트의 Agent의 리스트를 유지를 하고 있다. 그리고 Agent의 추가, 삭제 그리고 각 Agent에서 등록한 어플리케이션을 추가, 삭제하는 역할을 한다.
- AgentModel 클래스 : Agent를 추상화한 클래스로써, 원격 호스트의 IP, Hostname등을 가지고 있는 클래스이다.
- ApplicationModel 클래스 : 원격 호스트의 어플리케이션을 추상화한 클래스로써, Agent가 모니터링하는 어플리케이션의 이름과 모니터링하기 위한 인터벌 설정 필드를 가지고 있다.
- DataBase 클래스 : Manager에 있는 Agent의 리스트 및 Agent에 설정된 어플리케이션 리스트에 대해서 데이타베이스에 읽기와 저장을 그리고 Agent에서 모니터링된 데이타를 저장한다.

그림 13에서 보듯이, ServerInfo는 AgentModel에게 registerAgent() 메쏘드를 통해서 Agent를 등록을 하고, getName() 메쏘드를 통해서 Agent의 중복 등록을 막아주고, AgentModel은 registerApp() 메쏘드를 통해서 ApplicationModel에 어플리케이션을 등록하게 해주

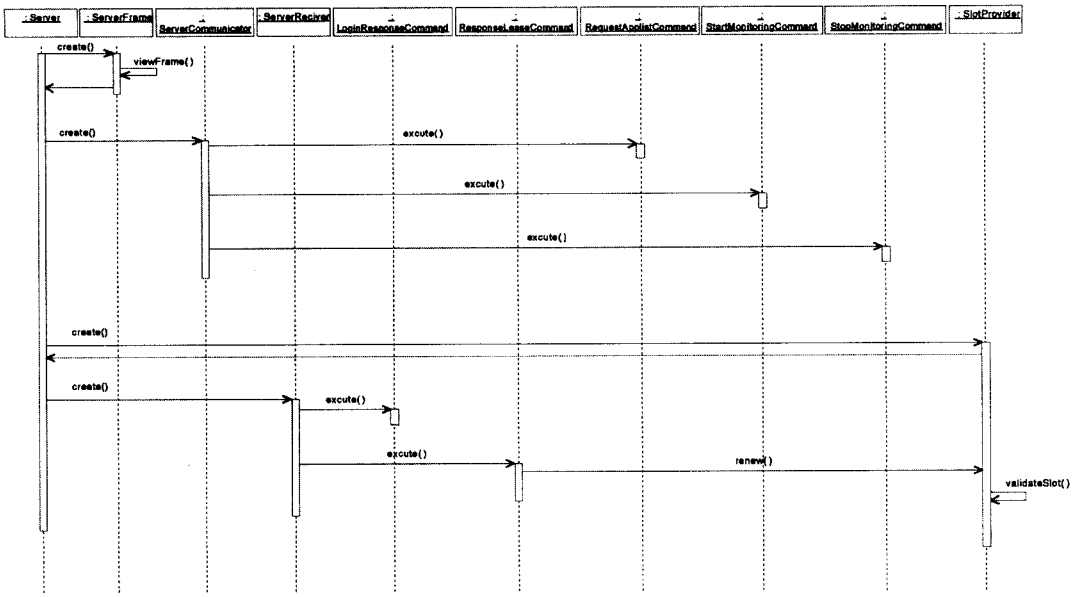


그림 11 Manager Communication 모듈 Interaction Diagram

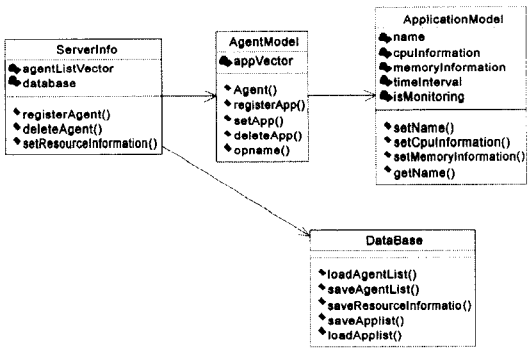


그림 12 Manager Management, DB 모듈의 Class Diagram

고, getName() 메쏘드를 사용해서 어플리케이션의 중복 등록을 막아준다. ServerInfo가 AgentModel에게 deleteAgent() 메쏘드를 통해서 Agent를 삭제요청을 하면, AgentModel은 ApplicationModel에게 deleteApp() 메쏘드를 통해서 어플리케이션 리스트를 삭제하고, 어플리케이션 리스트를 삭제되면, Agent 리스트가 삭제가 된다.

4. RAMS 구현

4.1 Agent 구현

그림 14에서 보듯이, Agent가 실행이 되면, Manager의 룩업 서비스를 처음으로 찾게 된다. 그리고 Manager가 실행이 되었는지를 판단하고, 호스트의 이름과 IP를 Manager에게 전송을 하였다. 다음으로 Manager가

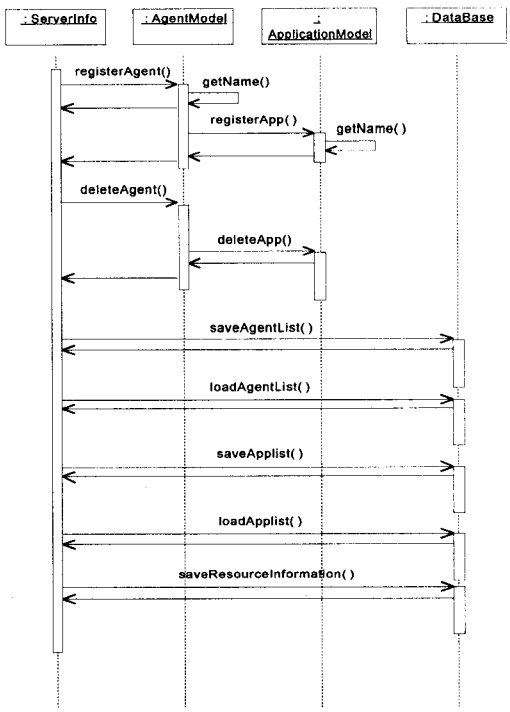


그림 13 Manager Management, DB 모듈의 Interaction Diagram

Agent에게 어플리케이션 리스트를 요청을 해서, Agent가 호스트의 어플리케이션을 수집해서 전송을 하였다.

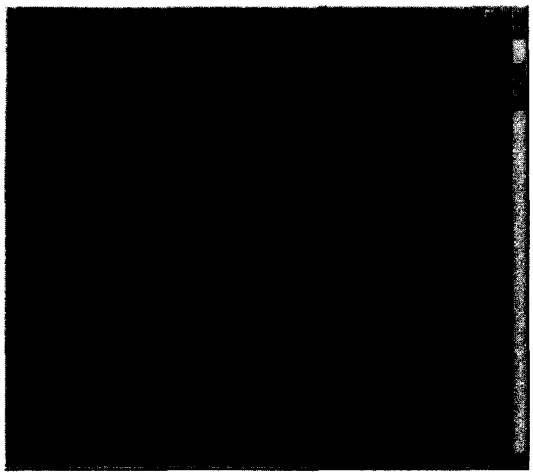


그림 14 Agent 실행 화면

다음으로 Manager가 Agent에게 모니터링 요청을 해서, Agent가 데이타를 수집해서 Manager로 전송을 하고 있다.

4.2 Manager 구현

그림 15는 Manager의 주 화면이다. 등록 화면은 자바 SWING의 JTable을 사용하였으며, 테이블에는 등록된 호스트의 이름과 IP 그리고 호스트에 대해서 간단하게 기술 할 수 있는 Description과 Agent를 체크 할 수 있는 필드로 구성이 되어 있다. 오른쪽의 주 메뉴를 살펴보면, Agent의 추가/삭제, Agent의Application의 편집/Application list 요청, Monitoring 시작/중지 요청과 Agent에서 모니터링 된 데이타를 전송받아 DB에 저장한 데이타를 통해서 Graph로 보여주는 Data Viewer와 Manager의 수행을 종료시키는 Exit Program으로 구성이 되어 있다.

그림 16은 등록된 Agent의 Application을설정하는 화

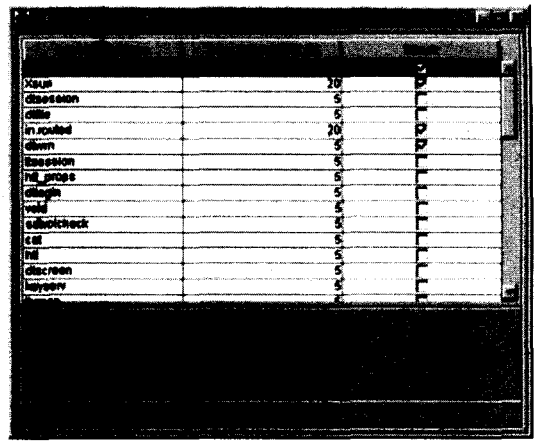


그림 16 Application list와 인터벌 설정 화면

면이다. 테이블을 보면, Application Name에는 실행하는 어플리케이션의 이름, Time Interval은 모니터링 간격 설정, Check는Application을 선택하는 필드이다. 그리고, 어플리케이션에 대한 정보는 테이블 아래 화면에, PID(Process ID), USEER, THR, PRI, RES, STATE, TIME, CPU등의 값을 통해서 보여주게 된다. RES 필드는 물리적인 메모리의 사용량에 대한 값이다.

그림 17에서 보듯이, RAMS는 원격 호스트인 willow의 어플리케이션, PID, USERNAME, THR,PRI(우선순위), 각 프로세스에 할당된 전체 메모리(가상 메모리, 프로그램이 차지하는 공간, 데이타 영역, 동적으로 할당된 영역)의 값인 SIZE, 물리적인 메모리의 사용량인 RES, 어플리케이션의 상태를 나타내는 STATE, 어플리케이션 사용시간과 CPU 점유율에 대해서 설정된 간격으로 계속적으로 변화하는 모니터링 하는 화면이다. STATE의 'sleep' 상태는 메모리에 어플리케이션은 로드 되었지만 실제 작업을 하지 않고 있는 상태를 나타내고 있고,

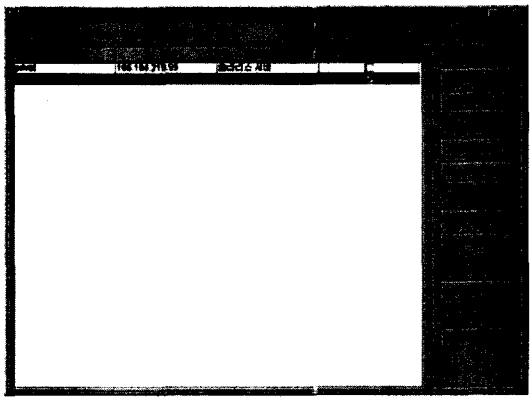


그림 15 Manager 주 화면

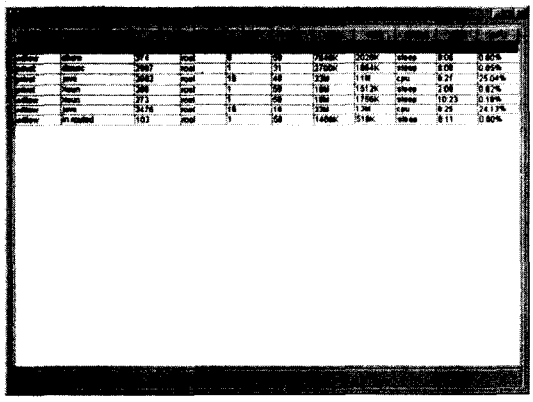


그림 17 RAMS의 실시간 모니터링 화면

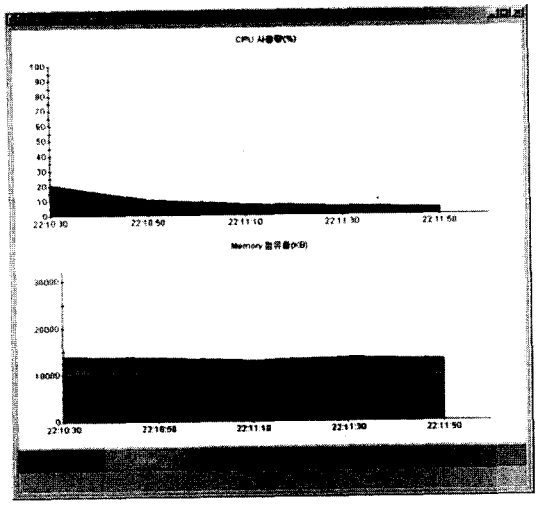


그림 18 호스트 어플리케이션들의 상태정보 화면

'cpu'상태는 메모리에 어플리케이션이 로드되었으며, 실제로 작업을 하고 있는 상태를 나타내고 있다.

그림 18에서 보듯이, 위에 그래프는 CPU의 사용량에 대한 화면이고, 아래 그래프는 물리적인 메모리(가상 메모리 제외)의 사용량에 대한 모니터링 화면이다. 위 그래프의 모니터링 호스트의 사양은 Sparc Machine으로 CPU:66, 메모리:32M의 사양이다. CPU 사용량(%) 그래프에서 보면, 처음 자바를 실행했을 때 CPU 점유율은 상당히 높았으나 차츰 낮아져서 안정된 모습을 보이고 있다. 메모리 점유율(KB)은 큰 차이가 없이 비슷한 공간을 차지하는 것으로 보이고 있다.

5. 결론 및 연구 방향

본 논문에서는 분산환경에서 네트웍에 존재하는 원격 호스트들을 효율적으로 관리하기 위해 실시간 원격 어플리케이션 모니터링 시스템인 RAMS를 JAVA, JINI, JNI, C 라이브러리를 이용하여 신뢰성을 확보하고, 확장성을 유지할 수 있도록 설계 및 구현을 하였다.

단순하게 호스트의 상태 정보를 모니터링 하는 기존의 원격 모니터링 시스템들과는 달리 RAMS는 두 가지 장점을 제공한다. 첫째, 어플리케이션 모니터링을 통해 서비스의 상태를 보다 정확히 파악하여 일정한 성능을 유지하면서 고객에게 지속적인 서비스를 제공할 수 있도록 한다. 이를 가능하도록 하기 위해 RAMS는 각 호스트에서 수행중인 하나 이상의 서버프로그램[1]들 각각에 대하여 자원 및 성능을 모니터링 할 수 있도록 하였고 이를 바탕으로 구체적으로 어떤 프로그램이 호스트의 상태에 영향을 미치고 있는 지 파악이 가능하게 됨으로 자원을 많이 사용하고 있는 프로세스를 다른 호스트로 이주시키는 등의 관리를 통해 고객에게 지속적인 서비스를 만족스럽게 제공할 수 있다.

둘째, 네트웍 및 호스트의 일시적 장애시에도 시스템 모니터링을 쉽게 restart할 수 있는 유연한 플랫폼 독립적 프레임워크를 제공한다. 클러스터의 구성은 다양한 기종으로 형성될 수 있기 때문에 어떠한 호스트를 클러스터에 추가하더라도 그에 필요한 작업이 최소화, 자동화 될 수 있어야 한다. RAMS는 Java에 기반한 JINI 기술과 JNI 기술을 사용함으로써 이 목적이 달성될 수 있음을 확인시켜 주었다.

RAMS의 한 가지 문제점으로 JINI 자체가 갖는 오버헤드가 비교적 크다[2]는 점을 들 수 있으나 이는 JINI 시스템 자체가 안고 있는 문제로서 자체적으로 해결할 수 있는 문제가 아니며, 향후 이 문제는 JINI 구현 기술이 발전함에 따라 극복될 것으로 기대된다. 그리고, 현재 시스템에서는 감시 대상인 어플리케이션이 수행되고 있는 호스트의 성능 또는 장애 발생시의 문제 해결에 초점을 맞추고 있으나, 궁극적으로는 Manager 장애시의 문제 해결 방안이 모색되어야 할 것으로 본다.

향후 RAMS는 원격 어플리케이션을 위한 모니터링 시스템뿐만 아니라 네트웍에 존재하는 이 기종의 모든 시스템에 대한 잠재적인 문제발생에 대한 진단과 예방이 가능한 시스템으로의 확장이 가능할 것으로 기대된다.

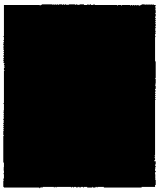
참 고 문 헌

[1] Tobias Oetiker, Dave RandMRTG, "Performance Monitoring Extensions," http://mrtg-pme.sourceforge.net
[2] Kammeyer Karl-Dirk "Sinfo," URL: http://www.ant.uni-bremen.de/whomes/rinas/sinfo
[3] Lattelekom, "Zabbix," URL: http://zabbix.sourceforge.net/
[4] SmartLine Inc, "Remote Task Manager," URL: http://www.protect-me.com/rtm/
[5] BEA Tuxedo, http://www.bea.com
[6] Jan Newmatch, "JINI Tutorial," URL: http://pandonia.canberra.edu.au/java/JINI/tutorial
[7] SUN Microsystems, "JINI Network Technology," URL: http://www.sun.com/jini/
[8] W.Keith Edwards "Core JINI," Prentice-Hall, 2001.
[9] Grodon, Rob "Essential JNI: Java Native Interface," PH/PTR, 1998.
[10] SUN Microsystems JNI Specification, URL: http://java.sun.com/products/jdk/1.2/docs/guide/jni/spec/j

1) 서버 프로그램은 DBMS, 시뮬레이션 엔진 등 업체가 제공하는 서비스를 수행하는 임의의 프로그램이 될 수 있다.

2) JINI 수행에 따른 오버헤드는 기종에 따라 다소 차이가 있으나 실험적 측정결과에 의하면 10~15%의 CPU를 사용하는 것으로 측정되었다.

niTOC.doc.html
[11] Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns," Addison Wesley, 1994.
[12] D.Schmidt, M.Stal, HRohnert Fbuschmann, "Pattern-Oriented Software Architecture," Wiley, 1999.



임 성 훈
2000년 강남대학교 컴퓨터공학과 학사
2003년 한양대학교 컴퓨터공학과 수료
관심분야는 Mobile & Wireless Network 등

송 무 찬
2000년 인천대학교 행정학과학과 학사
2003년 한양대학교 컴퓨터공학과 석사
2003년~ 네오위즈(Neowiz) 무선사업부 무선개발팀. 관심분야는 CORBA, ACE, Design Pattern, XML 등

김 정 선
1986년 서울대학교 컴퓨터공학과 졸업(학사). 1988년 Iowa Statr University 전기 및 컴퓨터 공학과 졸업(공학석사) 1994년 Iowa State University 전기 및 컴퓨터 공학과 졸업(공학박사). 1994년~1996년 한국전자통신연구원(ETRI) 선임연구원. 1996년~현재 한양대학교 컴퓨터공학과 부교수
관심분야는 Parallel & Distributed Processing, Component Based Development, Distributed Object Computing