TMN을 위한 실제자원 시뮬레이터 구현

정회원 송 병 권*, 김 건 웅**

Implementation of Real Resource Simulator for TMN

Byung-kwen Song*, Geonung Kim** *Regular Members*

요 약

본 논문에서는 실제 자원의 개발 전에도 망 관리 시스템의 개발 및 운용 테스트를 수행하도록 지원하는 실제 자원 시뮬레이터(RRS: Real Resource Simulator)를 소개한다. RRS는 객체의 상태를 유지하는 MOT(Managed Object Table)와 사용자가 정의한 동작 특성을 유지하는 SDT(Simulation Data Table), 랜덤(Random) 값과 랜덤 주기(interval) 값을 발생할 지원 함수들, 순차적인 사건 발생 또는 값의 수정을 지원하는 스케줄링 테이블 그리고 이들을 전체적으로 관장하는 메인 커널로 이루어져 있다. 또한 사용자는 확장된 GDMO 문법을 이용하여 RRS의 동작 특성을 지정할 수 있는데, 본 논문에서는 이를 처리할 확장된 GDMO 컴파일러를 설명하고, 사용자와 RRS간의 동작, RRS와 에이전트간 동작, RRS, 에이전트, 망관리 시스템으로 이루어진 환경에서의 동작 확인 결과를 보인다.

ABSTRACT

In this paper, we propose a RRS(real resource simulator) that supports the development and operational test of network management system before the development of real resources. The components of the RRS are the MOT(Managed Object Table) that holds the current status information of real resources, the SDT(Simulation Data Table) that holds the characteristics of real resources defined by user, the support functions that generate the random values and random interval values, the scheduling table that holds the sequence of events, and the main kernel. Users can set up behaviors of the RRS by extended-GDMO description. We present the structure of our extended-GDMO compiler and activities of RRS. We also show the interaction between user and the RRS, interaction between the RRS and the agent, and interaction among the NMS, the agent and the RRS.

I. 서 론

TMN(Telecommunication Management Network)은 표준화된 개방형 방식의, 총체적이고 일원화된 통신망 운용관리체제를 구축하기 위해 권고된 것으로 관리 정보 모델링을 위해 OSI(Open System Interconnection)에서 제안한 객체지향 모델링 기법을 채택하고 있고, 관리 정보의 접근 및 교환을 위하여 OSI의 CMIP(Common Management Information Protocol)/CMIS (Common Management Information Service) 프로토콜을 채택하고 있다[1][2][3][4][5][6][7][8][9][10].

TMN에서는 망 관리 역할에 따라 관리자(manager)와 에이전트(agent)로 구분하고 있다. 관리자는 에이전트에게 관리 요청을 전달하고 에이전트로부터 관리 요청에 대한 수행 결과를 반환 받거나, 특별한 사건이 발생했음을 알리는 통고(notification) 메시지를 수신한다. 또한 에이전트는 관리자로부터 관리요청을 수신한 다음 실제 자원(real resource)에 접근하여 해당 값을 가져온 후, 그것을 관리자에게 반환하거나 실제 자원에서 발생한 통고를 전달한다. 따라서 관리자는 에이전트를 통하여 실제 자원에 대한 관리를 종합적으로 수행한다.

* 서경대학교 정보통신공학과 (bksong@skuniv.ac.kr), ** 목포해양대학교 해양전자·통신공학부(kgu@mail.mmu.ac.kr),
논문번호 : 020163-0412, 접수일자 : 2002년 4월 12일

망 관리 시스템에서 실제 자원은 전기통신망을 구성하는 교환기 및 각종 유무선 통신 장비 등 실제 관리하고자 하는 하드웨어 또는 소프트웨어적인 요소들이다. 이러한 실제 자원을 관리하기 위한 망 관리 시스템은 실제 자원이 완성된 후에 개발을 시작하거나, 실제 자원 개발자로부터 사전에 관리 정보를 제공받아 실제 자원 개발과 병행해서 개발한다. 그러나 전자의 경우는 완성된 실제 자원을 기존 통신망에 설치 및 운용했을 때, 해당 장비에 대한 관리 시스템 부재로 일정 기간 동안 관리 공백이 초래되고, 후자 또한 실제 자원 개발자로부터 얻는 정보의 부족으로 인해, 양쪽이 완성된 이후에도 일정 기간 동안 정합시험이나 안정성 시험을 거치면서 문제점을 수정, 보완해야 한다. 또한 망 관리 시스템이나 에이전트들이 개발된 후에 망 관리 시스템 자체에 대한 기능 및 성능을 분석하고자 하면, 실제 망 요소들을 이용해 테스트 환경을 갖춘 후 이를 수행해야 하는데, 이러한 환경 구축에 소요되는 비용이나 시간도 많이 필요하다.

이러한 문제점들을 해결하기 위해선, 실제 자원 개발 전에도, 사용자가 지정하는 형태로 속성(attribute) 값들이나 사건(event)을 생성하여 에이전트에게 돌려주고, 또한 실제 자원 없이도 운용 환경을 꾸밀 수 있도록 하는, 실제 자원 시뮬레이터가 필요하다. 본 논문에서는 이를 RRS(Real Resource Simulator)로 지칭한다. RRS의 주 역할은 망 관리 시스템이 관심을 갖는, 실제 자원이 동작하면서 발생하는 속성 값들의 변화나 사건 등을 사용자가 원하는 방식으로 생성시켜, 관리시스템의 개발 및 운용 테스트를 하도록 하는 것이다.

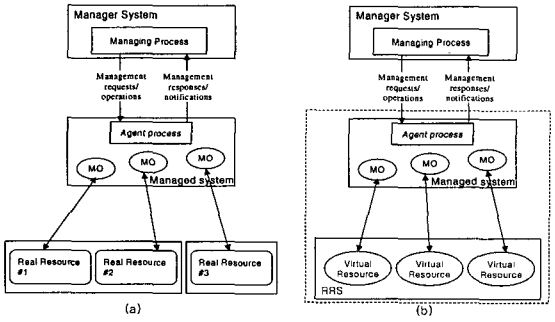


그림 1. 관리자, 에이전트 및 RRS의 관계

그림 1은 관리자, 에이전트 그리고 RRS와의 관계를 나타낸다. (a)는 현재의 망관리 시스템 구성이고 (b)는 RRS를 이용한 환경을 보여준다. 본 논문에서는 구현의 편이성을 위해 (b)의 점선부분과 같이 에이전트 역할과 RRS 역할을 동시에 수행하도록 하였다.

본 논문에서는 [11][12]에서 제안하였던 RRS의 구현 결과를 소개한다. 먼저 2장에서 RRS의 구성 요소들을 소개하고, 3장에서는 RRS를 위해 확장한 GDMO 컴파일러에 대해 설명한다. 다음 4장에서는 구현된 RRS의 전체 동작과 RRS를 이용한 망 관리 시스템 테스트 과정을 소개하고 5장에서는 테스트를 목적으로 구현한 RRS의 동작화면과 에이전트와의 상호 동작 결과를 소개하고 6장에서 결론을 맺는다.

Ⅱ. RRS 구성

RRS는 에이전트에서 사용되는 MIB 정의에 따라 실제 자원에 해당하는 데이터를 생성하고, 망 관리 에이전트에서는 RRS에서 생성된 데이터를 실제 자원을 대신하여 사용할 수 있다. 사용자는 일반적으로 CMIP/CMIS를 기반으로 한 망 관리 에이전트를 개발하기 위해 망 관리 요구의 분석, 망 관리 모델의 정립, MIB의 정의, 에이전트 코드의 생성, 에이전트 프로그래밍, 그리고 에이전트 테스트 등의 절차를 수행한다. 일반적으로 MIB는 망 관리에 관련된 정보만을 포함하고 있는데, 자원의 동작에 대한 정보가 필요하므로 기존 GDMO를 이용한 MIB 기술에 생성 값과 발생 주기에 관한 정보를 추가하도록 하였다. 이와 같은 절차를 통해 RRS는 실제 자원에 해당하는 동작을 수행하여, 에이전트를 개발하거나 테스트할 때 사용될 수 있다.

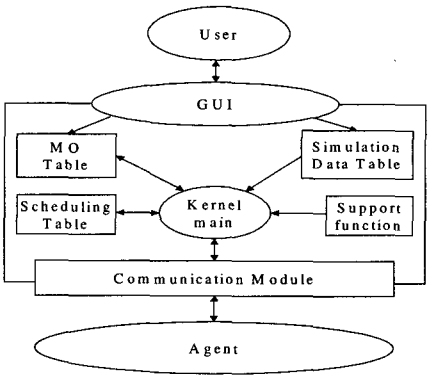


그림 2. RRS 전체 시스템 구조

RRS는 그림 2와 같이, 에이전트와의 다양한 통

신 방식을 지원할 통신 모듈, 각 관리 객체 별로 사용자가 지정한 속성 값 및 통고의 생성 방식 등을 담고 있는 SDT(Simulation Data Table), 현재 생성된 관리 객체들의 정보를 담고 있는 MOT(Managed Object Table), 이들을 바탕으로 속성 값을 변화시키거나 통고를 발생시킬 커널 메인(kernel main)과 이때 이용할 지원 함수(support function)코드와 스케줄링 테이블, SDT 또는 객체 테이블 내의 값 변경, 통고의 발생 등을 직접 수행할 수 있도록 하는 GUI 인터페이스로 이루어져 있다.

여기서 커널 메인은 ① 에이전트로부터 요청 받은 속성 값을 MOT에서 찾아 반환하고, ② 사용자가 지정한 자원함수에 따라 사건을 발생시키고, 이에 대한 통고 메시지를 생성, 에이전트에게 전달하는 기능을 수행한다. 또한 RRS가 망관리시스템의 테스트 목적으로 에이전트의 역할까지 겸하는 경우, 망관리 시스템에게 직접 속성 값이나 통고 메시지를 전달할 수도 있다. 커널 메인은 RRS내의 모든 자원들을 관리하게 되는데, 이에 관련된 관리 객체의 정보들이 MOT에 존재하게 되고, 각 자원마다 사용자가 지정한 시뮬레이션 동작 특성이 SDT에 존재하게 된다. 또한 순서적으로 사건을 발생시키기 위하여, 사건이 발생할 시간에 따라 등록하게 되는 스케줄링 테이블과 운영체제의 타이머를 이용한다.

지원함수에는 커널 메인에서 이용할 랜덤 값을 생성하는 함수들이 존재한다. 이러한 지원함수는 크게 두가지 용도로 쓰이게 되는데, 상태 값 자체의 생성과 사건이 일어나는 시간 간격 결정에 이용된다. 구현된 RRS에서는 Exponential, Continuous Uniform, Discrete Uniform, Binomial, Poisson, Gaussian 등의 확률 분포가 제공된다.

또한 각 관리객체 인스턴스마다, 다음 사건이 일어날 시간 간격을 결정하는데 이용하는 지원 함수와 매개변수(parameter)들이 다르기 때문에, 나중에 수행하여 결정된 사건의 발생이 먼저 결정된 사건보다 선행되어야 하는 경우도 생긴다. 따라서, 이를 위하여 지원 함수의 결과에 따른 사건 발생 시간을 기준으로, 순차적으로 정렬된 스케줄링 테이블이 필요하다. 이러한 테이블의 각 열에서는 관리객체 식별자와 상대적으로 계산된 시간 값을 가지고 있다.

SDT에는 각 관리 객체의 속성 값 및 통고들을 생성할 때 이용할 정보들을 담고 있다. 관리 객체마다 여러 속성과 사건 발생이 가능하므로, 관리 객체 식별자와 속성 식별자, 사건 식별자를 이용, 구분하도록 하고, 랜덤 값을 생성하는데 이용할 지원함수, 사건 발생 간격을 결정할 지원 함수, 이들 지원 함수들의 관련 매개 변수들을 담고 있다.

MOT에는 실제 자원의 현재 값들이 저장된다. 실제 자원 상태는 바로 전 상태에 종속적인 것과 독립적인 것이 있을 수 있는데, 종속적인 경우 바로 전 상태 값을 바탕으로 새로운 상태 값이 생성되어야 한다. 따라서 이를 위해 현재 상태 값들을 저장할 MOT가 있다.

또한 GUI를 통해 커널 동작 전에 SDT의 값들을 초기화하거나, 동작 중에도 사용자의 뜻에 따라 MOT에 저장된 정보들을 변경할 수 있도록 되어 있다.

Ⅲ. 확장된 GDMO 컴파일러

사용자는 시뮬레이션에 관련된 SDT 정보들을 생성하기 위해서 기존의 관리 객체 정의와 별개로 각 관리 객체의 동작을 기술할 수 있어야 한다. 이러한 SDT 정보를 기술하고 처리하는 방안은 여러 가지가 있는데, 본 논문에서는 기존의 GDMO 문법[8]을 확장하여 이를 기술하도록 하고, 기존 GDMO 컴파일러의 기능을 확장하여 이를 처리하도록 하였다.

3.1 GDMO 문법의 확장

사용자는 표준 GDMO 문법에 시뮬레이션 신택스(syntax)을 포함한, 확장된 GDMO 문법을 이용하여 시뮬레이션하고자 하는 실제 자원의 행동을 표현할 수 있다. 원래 GDMO 문법은 관리되는 객체를 9개의 템플릿(template)으로 나누어 정의하도록 되어있다. 이중에서 관리 객체 템플릿은 관리 객체의 구성 요소를 정의하며, 이때 정의되는 구성 요소는 부모 클래스(class)와 관리 객체에 포함되는 패키지(package)들이다. 패키지 템플릿은 관리 객체 템플릿에서 참조하는 패키지의 구성 요소들을 정의하고, 여기에는 속성과 속성 그룹(attribute group), 동작(action), 통고 등이 있다. 이외에 속성 템플릿, 속성 그룹 템플릿, 동작 템플릿, 통고 템플릿, 파라미터(parameter) 템플릿, 네임바인딩(namebinding) 템플릿, 행동(behaviour) 템플릿이 있다. 구현된 RRS에서는 시뮬레이션할 대상을 관리 객체의 속성과 통고로 제한하였고, 확장된 GDMO 문법에서는 패키지 템플릿에서 속성과 통고를 실제 값이 아닌, 시뮬레이션 값을 사용한다는 것을 명시할 수 있다. 따라서, 사용자는 패키지 템플릿을 기술하면서 속성과 통고를 시뮬레이션 한다는 것과, 시뮬레이션에 사용

하는 주기 및 생성 함수의 종류와 각각에 대한 파라미터를 정의할 수 있다. 또한 주기 및 생성 함수를 기술하지 않은 경우는 테스트베드의 운용 중에 GUI를 통해 사용자가 직접 지정할 수 있다. 그림 3은 확장된 GDMO의 패키지 템플릿에 대한 문법을 나타낸다. 여기서 굵은 글자로 나타난 것이 추가된 문법이다.

```
<package-label> PACKAGE
      [BEHAVIOUR<behaviour-definition-label>
                  [, <behaviour-definition-label>]* ; ]
      [ATTRIBUTES     <attribute-label> propertylist [<parameter-label>]*
                  [, <attribute-label> propertylist [<parameter-label>]* ]* ;
]
      [ATTRIBUTE GROUPS     <group-label> [<attribute-label>]*
                  [<group-label> [<attribute-label>]*]* ; ]
      [ACTIONS    <action-label> [<parameter-label>]* [, <action-label>
                  [<parameter-label>]*]* ; ]
      [NOTIFICATIONS
                  <notification-label> [period-definition <parameter-label>]*
                  [, <notification-label> [period-definition <parameter-label>]*]*
; ]
[REGISTERED AS object-identifier] ;

supporting productions
propertylist  →     [REPLACE-WITY-DEFAULT]
              [DEFAULT VALUE          value-specifier]
              [INITIAL VALUE          value-specifier]
              [PERMITTED VALUES       type-reference]
              [REQUIRED VALUES        type-reference]
              [RANDOM VALUES          [random-value-specifier] ]
              [get-replace]
              [add-remove]
period-definition   →     RANDOM VALUES [random-function-definition]
value-specifier     →     value-reference |
                          DERIVATION RULE <behaviour-definition-label>
random-value-specifier →       period-function-specifier
                          generate-function-specifier
get-replace   →     GET | REPLACE | GET-REPLACE
add-remove    →     ADD | REMOVE | ADD-REMOVE
period-function-specifier          → PERIOD random-function-definition
generate-function-specifier → GENERATE random-function-definition
random-function-definition → EXPONENTIAL <real-value>
                          | CONTINUOUS UNIFORM <real-value> <real-value>
                          | DISCRETE UNIFORM <integer-value> <integer-value>
                          | BINOMIAL <integer-value> <real-value>
                          | POISSON <real-value>
                          | GAUSSIAN <real-value> <real-value>
```

그림 3. 확장된 GDMO의 패키지 탬플릿 문법

그림 4는 확장된 GDMO로 기술된 패키지 템플릿의 예를 보여준다.

```
samplePackage              PACKAGE
   ATTRIBUTES
      ObjectId             GET,
      CurrentUser          GET-REPLACE
                  RANDOM VALUES PERIOD EXPONENTIAL 20
                  GENERATE DISCRETE UNIFORM 10 100,
      CurrentProcess       REPLACE-WITH-DEFAULT   GET-REPLACE;
   NOTIFICATIONS
      objectCreation,
      objectDeletion,
      attributeValueChange RAMDOM VALUES;
REGISTERED AS   { samplePackage 1 };
```

그림 4. 확장된 GDMO 문법으로 기술된 패키지 탬플릿 예

여기서는 CurrentUser라는 속성을 시뮬레이션 값으로 사용한다는 것을 선언하였고, 주기 함수로 Exponential 함수를, 값 생성 함수로 Discrete Uniform 함수를 사용한다고 선언하였다. 또한 attributeValueChange라는 통고를, 시뮬레이션으로 생성시켜 이용한다고 선언하였으며, 주기 함수는 시뮬레이터에서 초기화하도록 하기 위해 선언하지 않았다.

3.2 GDMO 컴파일러의 확장

기존의 GDMO 컴파일러는 사용자가 기술한 GDMO를 입력으로 하여 에이전트에서 이용할 관리 객체 코드를 생성한다. 그러나 RRS에서는 실제 자원을 대신하여 시뮬레이션 하는 동작에 관련된 정보 역시 생성되어야 한다. 따라서 확장된 GDMO 컴파일러는 그림 5와 같이 사용자의 입력에서 추가된 문법을 분리하고, 이를 이용하여 SDT 정보를 생성한다.

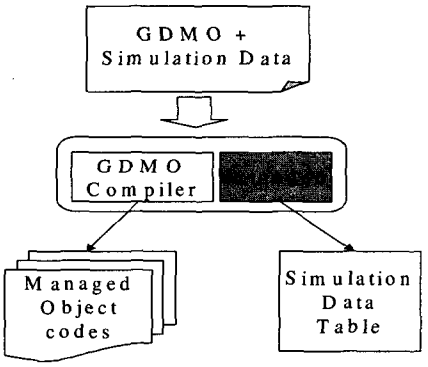


그림 5. 확장된 GDMO 컴파일러의 동작

실제로 시뮬레이션에 관련된 부분은 SDT 정보뿐이지만, 원래의 GDMO 컴파일러의 출력인 관리 객체 코드도 같이 이용할 수도 있으며, 이런 경우에는 망 관리시스템의 에이전트 생성도 동시에 수행할 수 있다. 확장된 GDMO 컴파일러는 사용자가 기술한 문서를 받아들여 원래의 GDMO 부분과 시뮬레이션 관련 부분을 분리하여 내부 클래스에 저장한다. 이와 같이 분리된 데이터를 바탕으로 관리객체 코드와 SDT 파일을 생성하게 된다. 본 논문에서는 UCL의 OSIMIS[13][14][15][16]의 GDMO 컴파일러를 개량하여 확장 GDMO 컴파일러를 구현하였다.

그림 6은 확장된 GDMO 컴파일러의 전체적인

구조를 보이고 있다. 여기에는 내부적으로 전체적인 동작을 수행하는 Main 루틴과 초기화 루틴, GDMO 문서 인식 루틴, 내부 데이터 클래스, 외부 파일 출력 루틴이 있다.

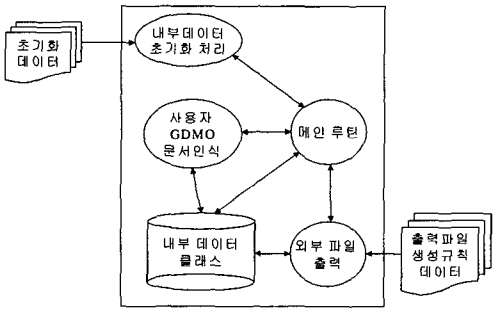


그림 6. 확장된 GDMO 컴파일러의 구조

Main 루틴에서는 확장된 GDMO 컴파일러의 전체 동작을 관장하며, 각각의 내부 요소들과 상호 동작한다. Main 루틴의 주요한 동작은 ① 내부 데이터의 초기화와 ② 각각의 내부 요소의 동작 순서 유지, ③ 각 내부 요소에서 발생한 에러 처리 등이다. 내부 데이터 초기화 루틴에서는 확장된 GDMO 컴파일러가 수행하기 위해 필요한 초기화 파일들을 읽고, 그 정보를 Main 루틴에 전달하는 역할을 수행하고, GDMO 문서 인식 루틴에서는 ① 사용자가 기술한 문서를 읽어서 이를 해석하는 역할과 ② 해석된 데이터를 내부 데이터 클래스에 저장하는 역할, ③ 에러 발생 시 Main 루틴에 보고하는 역할, 그리고 ④ 사용자 문서에 대한 처리가 끝났을 때, 처리 결과를 Main 루틴에 보고하는 역할을 수행한다. 내부 데이터 클래스에는 사용자의 GDMO 문서에서 인식한 데이터들을 저장하고, 외부 파일 출력 루틴에서는 사용자 문서에서 인식된 결과를 이용하여 필요한 관리 객체 클래스 코드와 시뮬레이션 데이터를 생성하는 역할을 수행한다. 또한 초기화 데이터에서는 컴파일러를 동작시키기 위해 필요한 초기화 정보를 유지하는데, 여기에는 기존에 정의된 관리 객체, 속성, 신택스의 정보등이 포함된다. 마지막으로 출력 파일 생성 규칙 데이터에는 내부 데이터 클래스에 저장된 정보를 이용하여, 관리 객체 클래스와 시뮬레이션 데이터를 생성하는 규칙을 저장한다.

확장된 GDMO 컴파일러의 수행 과정은 그림 7에서 보이는 바와 같이 다음과 같다.

① 확장된 GDMO 컴파일러는 초기화 데이터 파일을 읽고, 초기화 데이터 파일의 정보를 이용하여 내부 데이터를 초기화 한다.

② 컴파일러는 사용자가 작성한 GDMO 문서를 읽고, 사용자 문서에 포함된 정보들을 인식한다.

③ 사용자의 문서에서 인식된 정보를 이용하여 내부 데이터 구조에 해당 정보들을 저장한다.

④ 확장된 GDMO 컴파일러는 내부 데이터 구조에 저장된 정보를 이용하여, 출력 파일 생성 규칙에 따라 관리 객체 클래스 파일과 SDT를 생성한다.

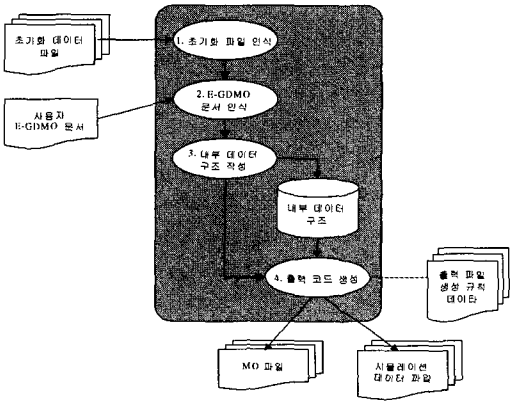


그림 7. 확장된 GDMO 컴파일러의 수행과정

Ⅳ. RRS의 동작 시나리오

4.1 GUI를 통한 SDT 정보 설정

구현된 RRS는 확장된 GDMO 컴퍼일러를 통해 기본 SDT가 만들어지는데, 사용자는 RRS의 동작 중에도 SDT 값을 초기화시키거나 수정할 수 있다. 그림 8은 사용자가 GUI를 통하여 SDT에 있는 값을 초기화시키고, 그 결과를 바탕으로 RRS가 MOT와 스케줄링 테이블을 초기화시키는 과정을 보여준다.

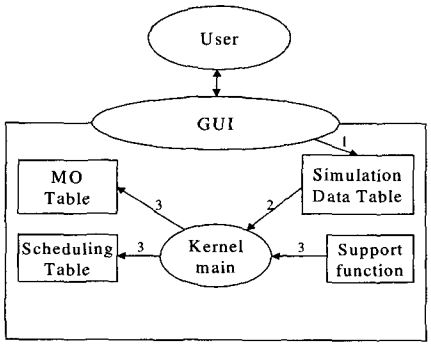


그림 8. GUI를 통한 SDT 초기화 및 동작

① 사용자는 GUI를 통하여 특정 관리 객체의 초기값, 랜덤한 속성 값을 생성하기 위한 지원 함수, 사건 또는 값의 변경이 일어날 시간 간격을 결정할 지원 함수, 그리고 각 함수들의 매개 변수들을 설정한다.

② RRS가 기동하게 되면, 커널은 사용자가 설정한 값들을 SDT로부터 읽어온다.

③ 커널 메인은 SDT에서 읽어온 값을 이용하여 MOT를 초기화하고, 다음 사건 또는 속성 값의 수정이 이루어질 시간을 결정하여 스케줄링 테이블에 등록한다.

RRS 동작 중에도 사용자는 GUI를 통하여 SDT에 저장된 값들을 변경할 수 있는데, 이 경우 해당된 MO들에 대해서만 값을 수정하고, 다음 사건 또는 변경이 일어날 시간을 결정하여 스케줄링 테이블에 등록한다.

① 사용자는 GUI를 통하여 특정 관리 객체의 값, 랜덤 생성 지원 함수 및 랜덤 간격 결정 지원 함수, 그리고 각 함수들의 매개 변수들을 수정한다.

② 변경된 경우, 이를 인터럽트 등의 통신 수단을 통해 커널 메인에게 알린다.

③ 커널 메인은 SDT에서 변경된 값을 이용하여 MOT를 수정하고, 다음 사건 또는 속성 값의 수정이 이루어질 시간을 결정하여 스케줄링 테이블에 등록한다.

앞서 언급한 초기화 과정에서는 SDT 전체에 대한 초기화 과정 이후에 전체 관리 객체들에 대해서 초기화가 수행되는데, 동작 중 변경의 경우에는, 전체가 아닌 해당 MO들에 대해서만 변경이 이루어진다.

4.2 에이전트 요청에 따른 속성 값 반환

그림 9는 에이전트로부터 요청을 받고 이를 검색하여 반환하는 예를 나타낸다. RRS는 현재 MOT에 저장된 값을 바로 돌려준다.

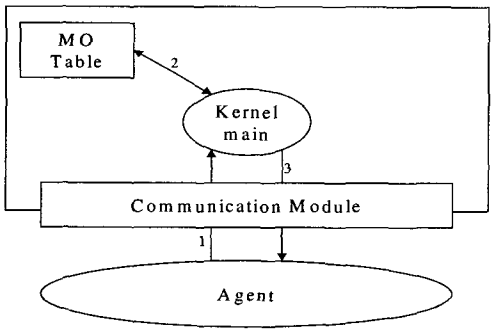


그림 9. 에이전트로부터 요청된 속성의 처리 과정

① 에이전트가 특정 관리 객체에 대한 속성 값을 요청한다.

② 커널은 MOT를 검색하여 속성 값을 얻는다.

③ 결과 값을 에이전트에게 반환한다.

4.3 속성값 생성 및 저장

시뮬레이션 커널이 동작하게 되면, SDT에 정의된 값들에 의해 속성 값들을 생성하고 이것을 MOT에 저장해야 한다. 그림 10은 이 과정을 보이고 있다.

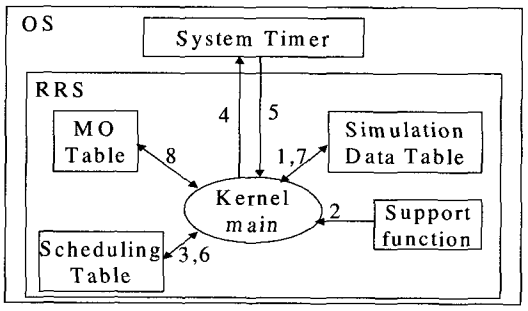


그림 10. 주기적인 속성값의 변경

① SDT가 초기화되었거나 속성값을 변경한 발생한 경우, SDT로부터 속성값을 생성시키기 위한 정보들을 가져온다.

② ①의 정보에 따라 해당 지원 함수를 실행시켜 다음 발생 시간을 구한다.

③ 구해진 발생 시간을 스케줄링 테이블에 등록한다. 이러한 스케줄링 테이블은 시간에 따라 순서적으로 정렬되어 있고, 상대적 타이머(relative timer)를 이용한다.

④ 맨 처음에 변경해야 하는 속성 값인 경우, 남은 시간을 OS의 타이머로 알린다.

⑤ 주어진 시간이 경과한 후 OS의 타이머로부터 시그널(signal)이 들어온다.

⑥ 스케줄링 테이블을 검색하여 해당 속성을 찾는다.

⑦ SDT로부터 랜덤 값을 발생시킬 지원함수를 찾는다.

⑧ 지원함수를 이용, 속성값을 생성하여 MO테이블의 속성 값을 변경한 후 ②로 돌아간다.

4.4 통고의 생성 및 전달

속성의 경우 발생 시간이 되면 지원 함수를 실행하여 그 결과 값을 MO 테이블에 저장하는 것에 반하여, 통고는 발생 시간이 되면 이를 능동적으로 에이전트에게 전달해야 한다. 이에 따라 통고를 처리

하는 과정은 속성의 처리 과정과 다르게 되는데, 그림 11에서 이를 보이고 있다.

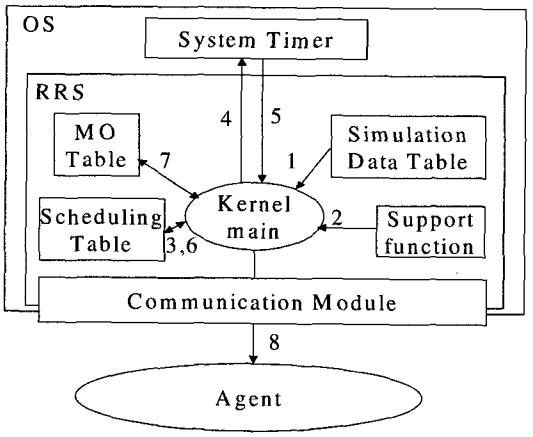


그림 11. 통고의 처리

① SDT가 초기화되었거나 통고가 발생한 경우, SDT로부터 통고를 발생시키기 위한 정보들을 가져온다.
② ①의 정보에 따라 해당 지원 함수를 실행시켜 다음 발생 시간을 구한다.
③ 구해진 발생 시간을 스케줄링 테이블에 등록한다. 이러한 스케줄링 테이블은 시간에 따라 순서적으로 정렬되어 있고, 상대적 타이머(relative timer)를 이용한다.
④ 맨 처음에 실행해야 하는 통고인 경우, 남은 시간을 OS의 타이머로 알린다.
⑤ 주어진 시간이 경과한 후 OS의 타이머로부터 시그널(signal)이 들어온다.
⑥ 스케줄링 테이블을 검색하여 해당 통고를 찾는다.
⑦ MO테이블로부터 통고와 관련된 정보를 찾는다.
⑧ 이를 에이전트에게 전송한 후 ②로 돌아간다.

V. 구현된 RRS의 테스트

에이전트와 RRS간의 동작 확인은 특정 MIB에 대한 가상 에이전트와 RRS를 구현한 후 둘간의 연동 확인절차로 이루어졌다. 개발된 RRS의 동작을 시험하기 위해 사용한 MIB 파일과 ASN.1 파일은 PCS망의 BSS(Base Station System)를 관리하기 위한 MIB를 수정하여 사용하였다. 사용한 MIB 파일과 ASN.1 파일은 testMIB.gdmo 와 testAttr.py로서 22개의 관리 객체와 138개의 속성을 가지고 있으며, 테스트를 위한 통고가 13개 선언되었다. 해당 MIB나 ASN.1 파일에 대한 논의는 이 논문 주제 범위를 넘어가므로 생략한다.

5.1 RRS의 동작 화면

5.1.1 GUI 메뉴

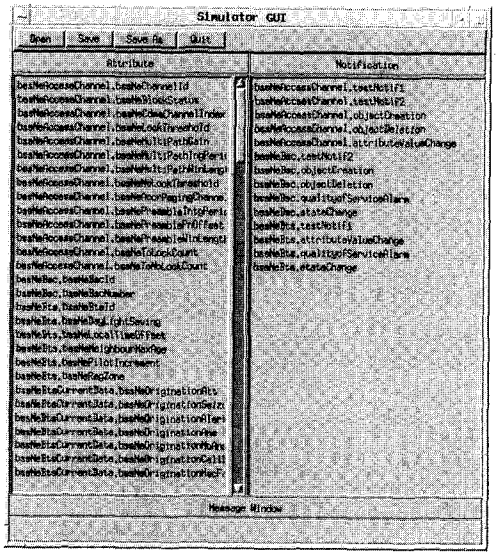


그림 12. 초기 GUI 화면

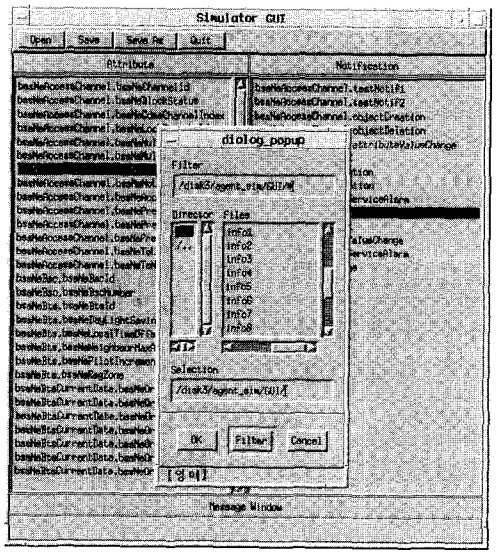


그림 13. Open 메뉴의 선택 화면

RRS를 실행시키면 초기 화면으로 그림 12와 같은 GUI가 생성된다. 생성된 GUI 화면은 MOT에

정의된 데이타를 이용하여 Attribute와 Notification 창을 표시하게 되며, Open 과 Save, Save As, 그리고 Quit 의 메뉴를 갖는다.

RRS는 관련 데이타를 읽은 후, 사용자가 지정한 확률 분포 함수와 파라미터 값을 저장하는 기능을 지원하며, Open 과 Save , 그리고 Save As 는 사용자가 지정한 확률 분포 함수와 파라미터 값을 읽고, 저장하는 기능을 지원하기 위해 사용된다. 그림 13은 이와 같은 동작의 한 예를 보여주는 것으로써 사용자가 이전에 저장한 정보를 읽는 창을 나타낸다.

5.1.2 애트리뷰트 값의 설정

초기 GUI 화면의 Attribute 창에서 원하는 속성을 클릭하면, 선택한 속성의 확률 분포와 파라미터 값을 설정할 수 있다. 애트리뷰트 설정 창에서는 선택한 애트리뷰트의 이름과 신택스의 이름, 그리고 신택스 타입을 보여주며, 확률 분포 함수에 대해 현재 설정되어 있는 값을 보여주게 된다.

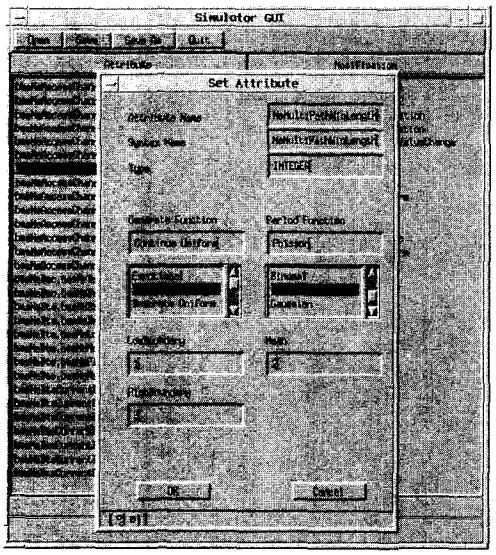


그림 14. 속성 값을 설정한 화면

그림 14는 속성값을 설정한 화면을 나타내고 있다. 사용자가 원하는 생성 함수와 주기 함수를 선택하면, 각 함수에 해당하는 파라미터의 이름과 입력란이 나타나게 되는데, 그림에서는 사용자가 생성 함수로서 Continuos Uniform 분포를 선택하고 LowBoundary 값으로 1 , HighBoundary 값으로 3 을 선택하고, 주기 함수로서는 Poisson 분포를 선택하고, Mean 값으로 2 를 선택한 경우를 나타낸다.

5.1.3 통고 값의 설정

초기 GUI 화면의 Notification 창에서 원하는 통고를 클릭하면, 선택한 통고의 확률 분포와 파라미터 값을 설정할 수 있다. 하나의 통고를 선택한 경우, 창이 새로 생기게 되는데, 여기서는 선택한 통고의 이름과 현재 설정된 확률 분포 함수의 종류와 파라미터 값을 보여주게 된다.

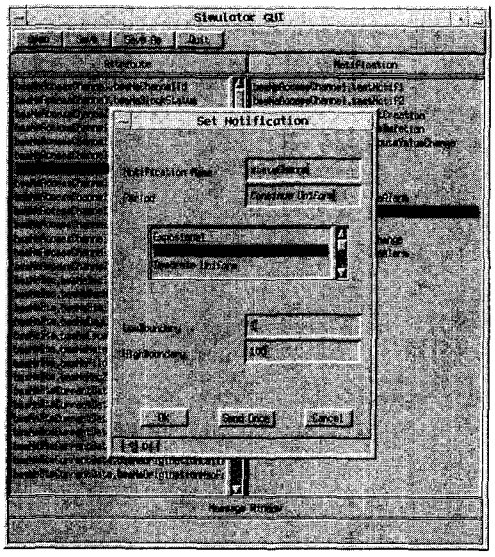


그림 15. 통고의 값을 설정한 화면

그림 15는 통고의 값을 설정한 화면을 나타내고 있다. 사용자가 원하는 주기 함수를 선택하면, 각 함수에 해당하는 파라미터의 이름과 입력란이 나타나게 되며, 가운데 나타난 창 그림은 사용자가 주기 함수로서 Continuous Uniform 분포를 선택하고 LowBoundary 값으로 1 을, HighBoundary 값으로 100 을 선택한 경우를 나타낸다.

5.2 에이전트와의 연동 테스트

에이전트와 RRS간에는 두가지 동작이 이루어지는데, 하나는 에이전트에서 속성값에 대한 요청을 RRS에게 전달하고, RRS가 이를 보내주는 것이고, 다른 하나는 RRS가 발생시킨 통고를 에이전트가 전달받는 것이다. 이러한 동작 결과를 확인하기 위하여 가상 에이전트를 만들었는데, 가상 에이전트는 두 개의 프로그램으로 구성되어 있고, 속성값 전달을 테스트하는 프로그램과 통고를 테스트하는 프로그램으로 나뉜다.

5.2.1 속성값 전달 확인

그림 16은 속성값 전달을 화기인하기 위한 가상 에이전트의 초기 화면이다. 초기 화면에서 가상 에이전트는 사용 가능한 관리 객체들을 보여 주게 되며, 사용자는 관리 객체의 번호를 입력하여 테스트할 관리 객체를 선택하거나, 가상 에이전트를 종료시킬 수 있다.

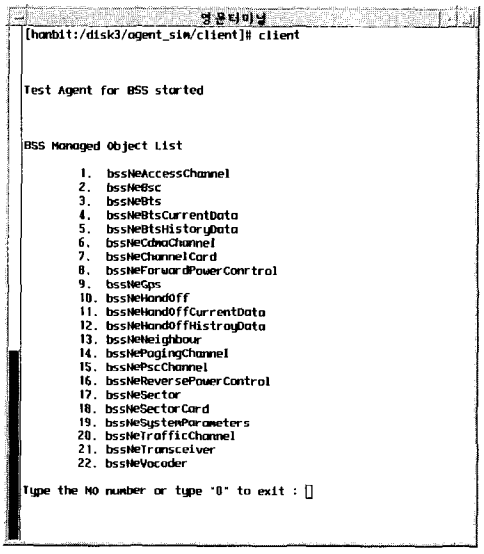


그림 16. 속성값 전달 확인을 위한 가상 에이전트 초기 화면

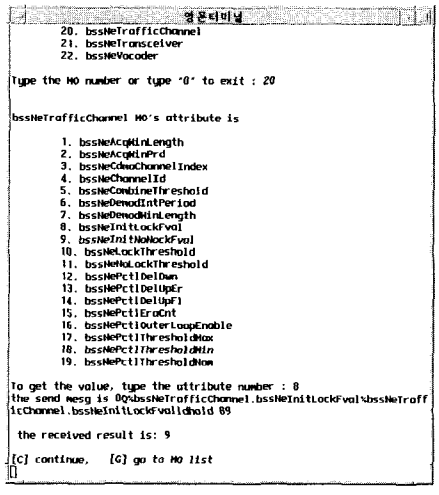


그림 17. 속성값 요청에 대한 응답 확인 화면

그림 17은 사용자가 bssNeTrafficChannel 이라는 관리 객체를 선택한 후 나타난 해당 관리 객체의 속성들 중에서 원하는 속성하여 값을 확인한 결과를 보여준다. 여기서 사용자는 bssNeInitLockFval 이라는 속성을 선택하여 RRS에게 요청하였고, RRS가 거기에 대한 응답으로 돌려준 결과 값인 9 를 출력한 후의 화면을 보여준다. 사용자는 결과를 본 후 다른 속성을 선택하기 위해서는 C 를 선택하고, 다른 관리 객체를 선택하기 위해서는 G 를 선택하게 된다.

이상의 과정에 대한 결과에서 RRS는 사용자가 선택한 속성에 대한 결과값을 정확히 전달한다는 것을 알 수 있다.

5.2.2 통고 전달 확인

통고 전달을 확인하기 위한 가상 에이전트의 동작은 사용자의 입력을 요구하지 않고 지속적으로 RRS가 발생시키는 통고 정보를 기다리며, 통고가 발생한 경우에 이를 출력한다. 그림 18은 통고에 대한 가상 에이전트가 실행하고 있는 화면이다. 가상 에이전트는 통고가 발생한 경우 통고의 이름과 통고를 발생시킨 관리 객체의 이름을 출력하게 되는데, 여기서는 bssNeAccessChannel 이라는 관리 객체에서 attributeValueChange 라는 통고가 발생하고, 그 후 bssNeBsc , bssNeAccessChannel , bssNeBsc 의 순서로 통고가 발생한 것을 보여준다. 이 결과에서 알 수 있듯이 RRS는 통고에 대하여도 정확한 동작을 수행한다.

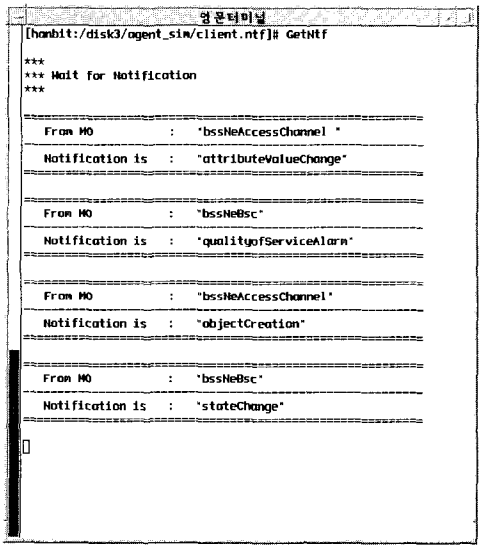


그림 18. 통고에 대한 가상 에이전트 화면

5.3 망관리 시스템과 에이전트, RRS간 동작 확인

실제 망관리 시스템에 RRS를 참여시켜 동작을

확인하였는데, 여기에서는 UCL(University Collage London)에서 개발한 OSIMIS 망관리 플랫폼을 이용하였다. 여기서는 망관리 시스템에서 가한 관리 동작이 에이전트를 통해 RRS에게까지 전달되고, 여기에 대한 결과가 에이전트를 거쳐 망관리 시스템에게까지 나타나는지의 여부를 확인하였다. OSIMIS를 이용한 에이전트와 망관리 시스템 개발에 관련된 내용은 본 논문이 관심을 가지고 있는 부분이 아니므로 생략한다. 그림 19는 OSIMIS의 망관리 브라우저로 에이전트와 접속하고 관리 객체를 선택하여 속성값을 요청한 결과를 보여주는데, RRS가 에이전트에게 전달한 값이 성공적으로 망관리 시스템에게 전달됨을 확인할 수 있다.

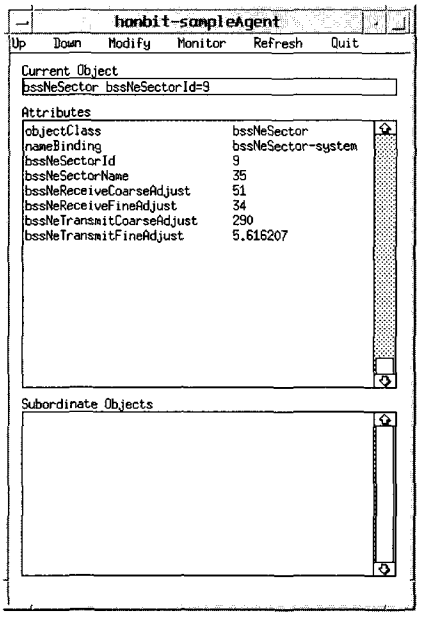


그림 19. 망관리 시스템의 속성값 확인 화면

Ⅵ. 결 론

본 논문에서는 망관리 시스템을 개발할 때, 실제 자원이 존재하지 않는 경우, 실제 자원의 역할을 대신할 수 있도록 구현한 RRS와 이를 위해 확장한 GDMO 문법과 컴파일러를 소개하고, 구현된 RRS의 동작과 망관리 환경에서의 동작 확인 과정을 설명하였다. 구현은 Solaris 환경에서 UCL의 OSIMIS를 기반으로 하였다.

일반적인 망 관리 시스템의 개발은 관리자와 에이전트로 나누어서 개발되지만, 관리될 실제 자원의 다양성을 감안하면, 에이전트의 개발이 더 많이 이루어진다. RRS는 사용되는 MIB의 정의에 따라 에이전트에서 요구되는 데이타를 랜덤하게 생성할 수 있는 기능을 제공하는데, 이때 생성되는 값을 실제 자원의 특성을 파악하여 사용자가 정의한 확률 분포에 의하여 발생함으로써 실제 자원이 없이도 에이전트를 개발할 수 있는 환경을 제공한다. 또한, 새로 도입될 실제자원에 대한 RRS를 미리 개발하여 테스트 환경을 구축함으로써 새로운 장비 도입 전에도 망관리 환경을 확인해볼 수 있는 기능도 제공한다.

그러나 실제 망 환경을 구성하고 테스트하기 위해선 각 망 요소의 동작 특성과 망 사용 특성에 대한 정보들이 축적되고 이를 RRS에 적용하는 것이 타당할 것이다. 따라서 특정 실제자원의 동작 특성과 망 사용특성에 대한 정보 수집을 병행하여 특정 망 요소를 시뮬레이션하는 RRS 개발 연구를 계속 진행할 예정이다.

참 고 문 헌

[1] ITU-T M.3010, "Principles for a Telecommunication Management Network"

[2] ISO7498-4/ITU-T X.700, "OSI Basic Reference Model Part 4: Management Framework"

[3] ISO10040/ITU-T X.701, "Systems Management Overview"

[4] ISO9595/ITU-T X.710, "Common Management Information Service Definition"

[5] ISO9596-1/ITU-T X.711, "Common Management Information Protocol Specification"

[6] ISO10165-1/ITU-T X.720, "Management Information Model"

[7] ISO10165-2/ITU-T X.721, "Definition of Management Information"

[8] ISO10165-4/ITU-T X.722, "Guidelines for the Definition of Managed Objects"

[9] ISO10165-5/ITU-T X.723, "Generic Management Information"

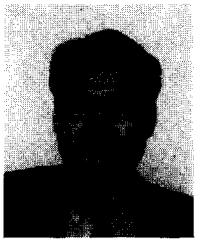
[10] ISO10164-5/ITU-T X.734, "Event Management Function"

[11] 송병권, 김건웅, 진명숙, "망관리시스템을 위한 테스트베드 설계", 한국정보처리학회, 13회 춘계 학술대회 논문집, 2000

[12] 송병권, 김건웅, 진명숙, "TMN을 위한 실제 자원 시뮬레이터 설계", 한국통신학회 논문지 26

권 10A호 pp.1727-1736, 2001
[13] George Pavlou, "The OSIMIS Platform: Making OSI Management Simple", Integrated Network Management IV, 1995
[14] George Pavlou, "High-Level Access APIs in the OSIMIS TMN Platform: Harnessing and Hiding", 2nd conference on Intelligent Services and Network Management, 1994
[15] George Pavlou, "Experience of Implementing OSI Management Facilities", Integrated Network Management II, 1991
[16] G. Pavlou, J. Cowan, J. Crowcroft, "A Generic Management Information Base Browser", IETF TC6/WG6.5, 1992

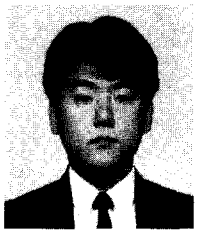
송 병 권(Byung-kwen Song) 정회원



1984년 : 고려대학교 전자공학과(공학사)
1986년 : 고려대학교 대학원 전자공학과(공학석사)
1995년 : 고려대학교 대학원 전자공학과(공학박사)

1984년~1997년 : 삼성종합기술원 선임연구원
1995년~현재 : 서경대학교 정보통신공학과 교수
<주관심 분야> High-speed Network, 분산처리시스템, Mobile computing

김 건 웅(Geonung Kim) 정회원



1990년 : 고려대학교 전자전산공학과(공학사)
1994년 : 고려대학교 대학원 전자공학과(공학석사)
1998년 : 고려대학교 전자공학과(공학박사)

1999년 9월~현재 : 목포해양대학교 해양전자 · 통신공학부 조교수
<주관심 분야> 통신망 프로토콜, 망관리 시스템, 지능망, 정보망, 멀티미디어 통신