망 관리 시스템 테스트베드를 위한 GDMO 컴파일러의 확장

송병권*, 김건웅**, 진명숙***
*서경대학교 정보통신공학과
**목포해양대학교 해양전자통신공학부
***경인여자대학 멀티미디어정보전산학부
e-mail:bksong@bukak.seokyeong.ac.kr, kgu@mmu.mmu.ac.kr,
jinms@dove.kyungin-c.ac.kr

Extension of GDMO Compiler for Network Management System Test Bed

Byungkwen Song*, Geonung Kim**, Myung-sook Jin***
*Dept. of Information & Comm. Eng., Seokyeong University
**Faculty of Electronics & Comm. Eng., Mokpo National Maritime University
***School of Multimedia Information Computing, Kyung-In Women's College

요약

실제 자원과 이에 대한 에이전트의 개발이 완료되기 전에는 망 관리시스템의 개발과 운용이 어려운 문제점을 해결하기 위한 방안으로 망 관리 시스템을 위한 테스트베드를 이용하는 방법이 있다. 이러한 테스트베드를 이용하기 위해선 사용자가 실제 자원의 동작을 반영하는 관리 객체의 속성과 통고에 대한 데이터를 조정할 수 있어야 하는데, 그중 하나의 방법이 기존의 관리 객체에 대한 기술을 지원하는 GDMO(Guidelines for Definition of Managed Objects)를 확장하여 이용하는 것이다. 본 논문에서는 망 관리 시스템의 개발과 운용을 돕는 테스트베드를 지원하기 위한 GDMO 문법의 확장 방안과 이를 위한 확장된 GDMO 컴파일러를 소개한다.

1. 서론

ITU-T에서 권고한 TMN[1]에서는 망 관리 역할에 따라 관리자(manager)와 에이전트(agent)로 구분한다. 관리자는 에이전트에게 관리 요청을 전달하고 에이전트로부터 관리 요청에 대한 수행 결과를 반환받거나, 특별한 사건이 발생했음을 알리는 통고(notification) 메시지를 수신한다. 반면에 에이전트는 관리자로부터 관리 요청을 수신한 다음 실제 자원(real resource)에 접근하여 해당 속성(attribute)값을 가져온 후, 그것을 관리자에게 반환하거나 실제 자원에서 발생한 통고를 전달한다. 여기서의 실제 자원은 전기통신망을 구성하는 교환기 및 각종 유무선 통신 장비 등 실제 관리하고자 하는 하드웨어 또는 소프트웨어적 요소가 된다.

이러한 실제 자원을 관리하기 위한 망 관리 시스템은 일반적으로 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 통신망에서 운용하고자 할 때, 해당 장비에 대한 관리 시스템 부재로 일정 기간 동안 관리 공백이 초래될 수 있고, 후자의 경우에는 사전에 얻을 수 있는 관리 정보가 정적인 정보에 국한될 수 있기 때문에 일정 기간 동안 정합 시험이나 안정성 시험을 거쳐야 한다. 이러한 문제점을 해결하는 한 방안이 실제 자원 개발 전에도 실제 자원의 동작을 시뮬레이션(simulation)하여 운용 환경을 제공하는 테스트베드(test bed)[1)]를 이용하는 것이다.

테스트베드는 다음 그림 1과 같이, 에이전트와의 다양한 통신 방식을 지원할 통신 모듈, 각 관리 객체별로, 실제 자원의 동작을 시뮬레이션 하는데 이용할 정보를 담고 있는 SDT(Simulation Data Table), 현재 생성된 관리 객체들을 담고 있는 테이블(MO Table), 이들을 바탕으로 속성 값을 변화시키거나 통고를 발생시킬 커널 메인(kernel main)과 이때 이용할 지원 함수(support function)코드, SDT 또는 객체

---

1) 동일 저자의 "망 관리 시스템을 위한 테스트베드 설계" 논문 참조

테이블 내의 값 변경, 통고의 발생 등을 직접 수행할 수 있도록 지원하는 GUI로 이루어져 있다.

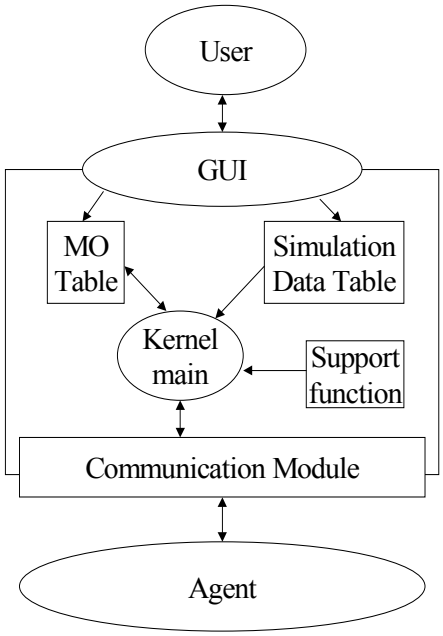

그림 1. 테스트베드의 구성

이 중, SDT에는 각 관리 객체의 속성 값 및 통고들을 생성할 때 이용할 정보들을 담고 있다. 여기에는 랜덤 값을 생성하는데 이용할 지원함수, 통고 발생 간격을 결정할 지원 함수, 이들 지원 함수들의 관련 매개 변수들을 담고 있다.

사용자는 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 문법을 확장하여 이를 기술하고, 또한 기존 GDMO 컴파일러를 확장시켜 처리하는 방법을 소개한다. 이를 위해 먼저 2장에서는 확장된 GDMO의 문법을 소개하고, 3장에서는 이를 처리하기 위한 GDMO 컴파일러의 동작과 전체 구조를 소개하며, 4장에서 결론을 맺는다.

2. 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 9 개의 템플릿(template)으로 나누어 정의하도록 하였다. 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.

앞서 언급한 테스트베드에서는 수행할 시뮬레이션 대상을 관리 객체의 속성과 통고로 제한하였다. 따라서, 확장된 GDMO 문법에서는 관리 객체에 포함되는 속성과 통고를 시뮬레이션할 수 있도록, 관리 객체의 속성과 통고를 기술하는 패키지 템플릿에 시뮬레이션을 위한 문법을 추가하였다. 그림 2는 E-GDMO의 패키지 템플릿에 대한 문법을 나타낸다.

```
<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 2. 확장된 GDMO의 패키지 템플릿 문법

그림 2와 같이 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 사용하는 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 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 3 확장된 GDMO 문법으로 기술된 패키지 템플릿 예

그림 3에서 사용자는 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. GDMO 컴파일러의 확장

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

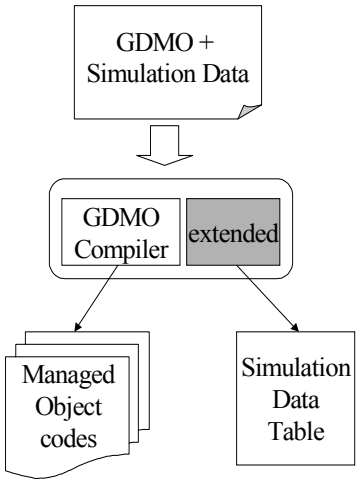

그림 4. 확장된 GDMO 컴파일러의 동작

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.

다음 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.

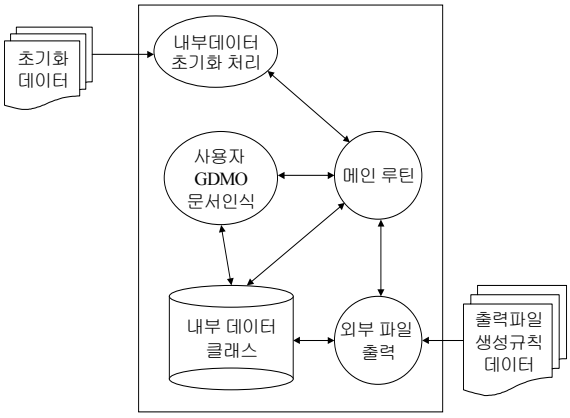

그림 6. 확장된 GDMO 컴파일러의 구조

Main 루틴에서는 확장된 GDMO 컴파일러의 전체 동작을 관장하며, 각각의 내부 요소들과 상호 동작한다. Main 루틴의 주요한 동작은 i)내부 데이터의 초기화와 ii)각각의 내부 요소의 동작 순서 유지, iii) 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 i)사용자가 기술한 문서를 읽어서 이를 해석하는 역할과 ii)해석된 데이터를 내부 데이터 클래스에 저장하는 역할, iii)에러 발생 시 Main 루틴에 보고하는 역할, 그리고 iv)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 데이터

를 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.

다음 그림 7은 확장된 GDMO 컴파일러의 수행과정을 나타낸다.

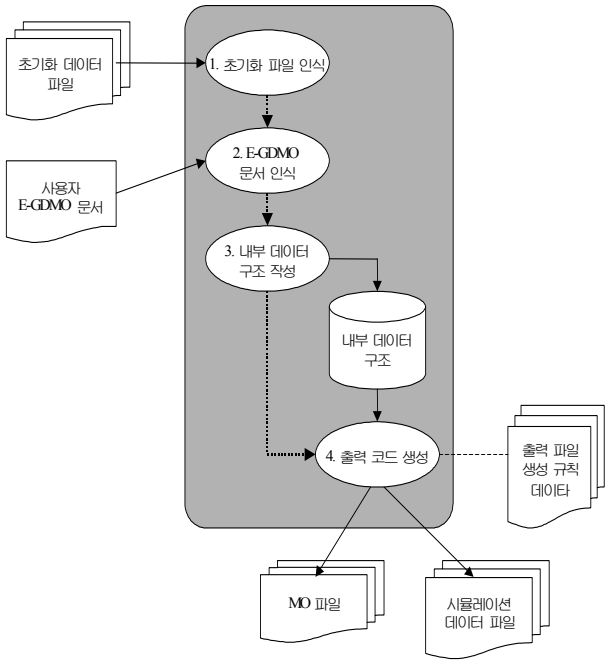

그림 7. 확장된 GDMO 컴파일러의 수행과정

(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.

4. 결론

본 논문에서는 실제 자원이 개발되지 않은 상황에서도 사용자가 정의한 실제자원의 특성에 따라 행동을 대신하는 테스트베드를 지원하기 위한 확장된 GDMO 문법과 이를 지원하는 확장된 GDMO 컴파일러를 소개하였다. 본 테스트베드는 관리 객체의 속성과 통고를 시뮬레이션 하도록 설계되었다. 따라서, 확장된 GDMO 문법은 표준 패키지 템플릿에 포함되는 속성과 통고에 대하여 시뮬레이션을 위한 문법을 추가한 형태로 구성되고, 확장된 GDMO 컴파일러는 이를 받아, 표준화된 문법과 시뮬레이션을 위한 문법을 분리한 다음, 관리 객체 코드를 생성하고 추가된 문법에 따라 SDT를 생성한다. 현재 연구 목적으로 학계와 연구소에서 많이 쓰이는 OSIMIS [11][12][13]를 이용, 구현 중이며, 구현이 완료되면 현재 구현중인 테스트베드와 결합하여 이에 대한 동작 검증을 수행할 예정이다.

참고문헌

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

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

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

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

[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] George Pavlou, "The OSIMIS Platform: Making OSI Management Simple"

[12] George Pavlou, "High-Level Access APIs in the OSISMIS TMN Platform: Harnessing and Hiding "

[13] George Pavlou, "Experience of Implementing OSI Management Facilities"