韓國 컴퓨터情報學會 論文誌
第9卷 第3號, 2004. 9.

2004-9-3-1-5

MIB 생성을 위한 GDMO 개발 환경 설계 및 구현

정 진 영*, 오 대 균**, 김 영 철***

A Design and Implementation of GDMO development environment for MIB generation

Jin-Young Jung *, Dae-Gyun Oh **, Young-Chul Kim ***

요 약

TMN(Telecommunication Management Network)에서의 네트워크 관리는 네트웍 통신 장비 및 운영 시스템인 관리 객체들을 정의하고 관리하는 것이다. GDMO(Guideline Definition Managed Object)는 이 객체들을 기술하기 위하여 이용된다. 그러나 GDMO는 네트워크를 직접 관리하기 위하여 이용되지는 않지만, 대신 객체 지향 패라다임을 가진 다른 언어로 번역되어 이용된다. 본 논문에서는 MIB를 생성하기 위하여 GDMO를 망관리에 필요한 다른 객체 지향 패라다임 언어로 자동 번역하는 번역기를 설계하고 구현한다. 또한 본 시스템은 GDMO의 개발 환경을 지원하기 위하여 다양한 그래픽 인터페이스를 지원한다.

Abstract

The management of network in TMN(Telecommunication Management Network) defines and manages the objects which are the operating system and communication equipments in network. The GDMO(Guideline Definition Managed Object) is used to describe those objects. GDMO is not directly used for managing network, but translated language into a language with object-oriented paradigm, then which is used. This paper presents design and implementation of the translator which automatically translates the specification of GDMO to the object-oriented language for generating MIB. And the system includes various graphic user interface to advance the developement environment of GDMO.

▸ Keyword : TMN, GDMO, MIB, ASN.1

• 제1저자 : 정진영
• 접수일 : 2004.08.16, 심사완료일 : 2004.09.01
* 대전보건대학 멀티미디어과 조교수, ** 한국에너지기술연구원, *** 숭실대학교 시스템 소프트웨어 공학박사

Ⅰ. 서 론

망관리에서의 정보 교환은 다양한 종류의 운영체제와 통신장비 사이에서 이루어진다. 망관리 분야 중 TMN은 다양한 하드웨어와 공급업체간의 호환성의 결여로 발생되는 망 관리 작업의 부담을 극복하며, 다양한 종류의 운영체제와 통신 장비 사이에 표준화된 인터페이스를 이용하여 정보의 교환이 이루어지도록 하는 것이다. TMN의 방법론은 OSI 시스템 관리의 객체지향 파라다임을 적용하고 있다. OSI에서 시스템 관리 모델에서 다루어지는 정보는 관리되는 자원으로 "관리객체(managed object)"로 나타낸다.

OSI 관리 정보 모델(MIM)은 시스템 관리 프로토콜에 의해 전송되는 관리 정보의 구조를 제공하기 위한 것으로 관리 객체를 다루는 정보 모델이다. 그림 1은 OSI 정보 모델을 보여주고 있다.

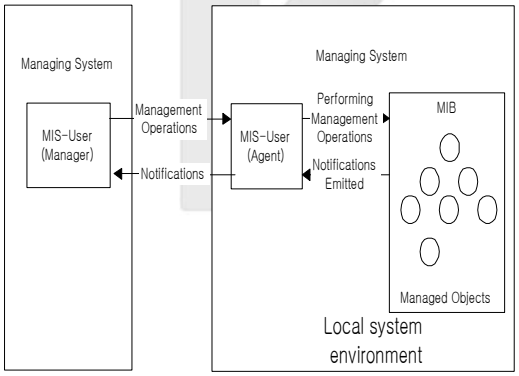

그림 1. OSI 관리 모델
Figure 1. Management Model

관리 정보 모델의 장점은 첫째, 관리 시스템이 실제 자원을 구현과 독립되어 관리적인 측면에서 볼 수 있는 관점을 제시하고, 둘째, 원격 시스템과 관리 정보를 교환하는데 있어서 통일된 형식으로 올바르게 관리 정보를 해석할 수 있는 방법을 제시하며, 세째, 캡슐화, 모듈화된 관리 시스템을 구성함으로써 시스템의 확장성과 재사용성을 높이는 것이다. 이러한 OSI 관리 모델과 같은 구조적인 프레임워크를 이용하여 여러 종류의 장비에 대한 관리를 수행하고 있다. 그러나, 이러한 체계하에서는 인터페이스와 행위를 표준화된 형태로 정의하고 관리할 수 있는 도구가 요구된다. 이를 위해서 GDMO가 표준으로 제정되었다.

GDMO는 관리 정보를 구조적으로 모델링하는 기술 도구이다. 그러나 GDMO 그 자체로 망 관리에 이용되기는 어려우며 객체지향 언어로 번역되어져야 한다. GDMO를 객체지향 프로그램 언어로 번역하는 이유로 첫째, 네트워크 및 시스템 소프트웨어 개발자에게 객체지향 언어는 관리 객체의 기술이 GDMO에 의한 명세보다도 더 명확하며 이해하기 쉽다. 둘째, 객체지향 언어로 번역된 결과는 즉각적으로 MIB 구현에 필요한 OODB의 스키마로 표현될 수 있다.

본 논문에서는 GDMO 번역기 및 개발환경을 설계하고 구현하였다. GDMO 개발환경은 그래픽 사용자 인터페이스를 제공함으로써, 보다 효율적인 GDMO 명세를 작성할 수 있도록 하였다.

Ⅱ. GDMO 개발환경

GDMO는 우선 관리 객체 클래스 템플릿을 사용하여 망 자원이나 관리 정보를 추상화된 관리 객체로 규격화하여 정의한다. 또한 그 템플릿 안에서 Package, Behaviour, Attributes, Attribute Groups, Actions, Notifications, Name Bindings, Parameters와 같은 여러 템플릿를 다시 정의하거나 선언함으로써 그 관리 객체의 특성을 규정한다. 관리 객체 클래스 템플릿 내에 포함되어지는 템플릿들은 관리 객체 클래스 템플릿이나 Package 템플릿 내에서 정의하지 않고, 자신의 템플릿에 특성을 정의할 수 있다. 이들 템플릿을 사용하는 관리 객체 클래스 템플릿이나 Package 템플릿은 그 내부에서 선언하여 사용하도록 함으로써 GDMO의 객체 지향적 설계의 목적인 재사용성을 향상시킨다.

Name Binding 템플릿은 두 관리 객체 클래스 템플릿 간의 포함 관계를 설정하고 명명에 사용되는 속성을 정의한다. 또한 이들 템플릿의 데이터 타입들은 다시 ASN.1 모듈의 ASN.1 타입과 값으로 정의된다.

TMN에서는 관리 객체를 정의하기 위해 GDMO를 이용하며, 망관리 시스템의 많은 부분을 표준화하고 있다. 관리

객체에 대한 정의는 각각의 망 자원들을 규격화할 뿐만 아니라, 망 자원간의 관계를 표현하는 관리 객체간의 포함 관계를 규격화한다.

GDMO로 기술된 관리 정보 모델을 그대로 망 관리 구현 도구로 이용할 수 없다. 즉, GDMO로 기술된 관리 정보 모델은 C++ 언어와 같은 객체지향 파라다임을 지원하는 언어로 변환하는 번역기가 필요하다. 따라서 GDMO의 표현방법을 C++의 클래스의 표현 방법으로 매핑하는 과정이 필요하다. 그림 2는 GDMO 번역기의 구조를 표현한 것이다.

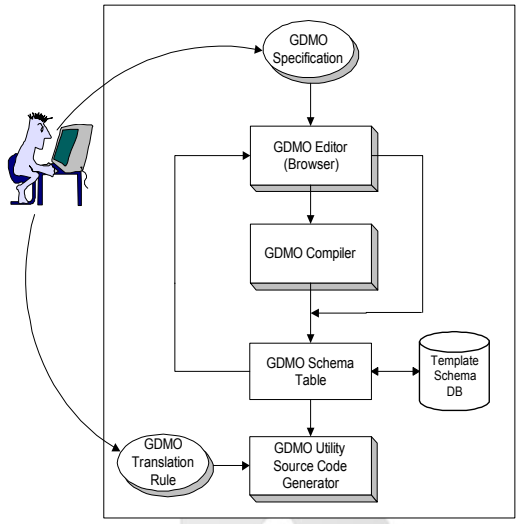

그림 2. GDMO 프로그램 개발환경의 구조
Figure 2. Structure of GDMO Program Development Environment

GDMO 브라우저는 사용자의 GDMO 명세를 입력으로 받아 GDMO 컴파일러나 중간 자료 구조인 GDMO 스키마 테이블로 번역하여 삽입한다. 이렇게 작성된 GDMO 스키마 테이블은 템플릿 스키마 데이터베이스에 저장된다. 저장된 중간 자료 구조는 GDMO 원시 코드 생성기에 의해 객체지향 파라다임을 가지는 언어로 번역되게 된다.

원시 코드 생성의 결과는 객체지향 파라다임을 지원하는 언어로 표현된다. 이것은 새로운 데이터베이스에 저장되어 망관리에 사용된다.

III. 기존의 GDMO 구현 사례

GDMO의 관리도구로는 HP OpenView GDMO 모델링 도구와 NOMAD GDMO 컴파일러 등이 있다. HP OpenView는 네트워크 시스템, 응용프로그램 및 데이터베이스를 포함하는 통합된 시스템을 관리를 하는 도구이다. 이 소프트웨어 도구는 OSI NM 객체 명세를 이용하여 그래픽 설계와 수정을 도와준다. GDMO 브라우저는 그래픽 사용자 인터페이스를 제공하며 객체 사전을 사용하여 관리 객체를 저장한다. 관리 객체의 입출력을 위해 GDMO 입력 도구와 GDMO 출력 도구를 제공한다.

OSCOM에서 만든 GDMO 컴파일러는 GDMO와 ASN.1 명세을 저장하는 GDMO 사전이 있으며, 관리 객체와 ASN.1 정의를 포함한 텍스트 파일을 입력으로 받아 처리하는 컴파일러인 GDMO Dictionary Loader가 있다. GDMO 브라우저 및 사전의 정의를 쉽게 접근하도록 해주는 GDMO 질의 언어를 제공한다.

이외에도 본 연구와 관련된 연구로는 ASN.1 개발환경이 있다. ASN.1은 GDMO를 관리하기 위한 구문 구조로 이용하기 때문에 관련 연구가 있다. ASN.1 컴파일러 개발 작업으로는 MAVROS[9], BBN[10], Snacc[11], ISODE의 PEPY/POSY[12], UBC의 CASN1[13] 컴파일러 등이 있다. MAVROS는 크게 2부분으로 구성되어 있다. 하나는 전처리기로 모든 ASN.1 명세의 타입을 지원하며, 모든 ASN.1 value들을 초기화하는 단계이다. 두번째 부분은 생성기 부분으로 여러 다양한 인코딩 루틴들을 생성하며, 프로그래밍 디버깅 도구도 지원한다. John Lowry의 BBN 컴파일러는 ASN.1 명세를 C++로 변환할 수 있으며, 자동 구문 검사(automatic syntax checking)를 한다. 또한 snacc은 ASN.1를 C나 C++ 코드로 바꿀수 있는 변환기이며, GUI(Graphical User Interface)를 지원한다.

Ⅳ. 설계 및 구현

4.1 GDMO 편집기

GDMO 편집기는 GDMO 명세를 생성, 수정, 삭제 등의 편집을 지원하는 메뉴를 제공한다. GDMO 편집기는 여러 선택사항을 제공하고 있다. DB에 저장된 GDMO 템플릿를 그래픽 형태로 나타내어 사용자 인터페이스를 강화하고 GDMO 템플릿에 익숙지 않은 사용자도 쉽게 이용할 수 있도록 한다. 편집기는 사용자에 의해 입력된 GDMO 명세 내용을 컴파일하거나 직접 GDMO 스키마 테이블의 형태로 바꾸어 데이터베이스에 저장한다.

GDMO 브라우저는 OSI 네트워크 관리 객체 모델을 하는 사람에게는 중요한 도구이며 그래픽 사용자 인터페이스을 통하여 GDMO 정의를 생성, 수정할 수 있다. 브라우저가 제공하여 주는 기능은 다음과 같다.

- 그래픽 디스플레이와 상속성 및 내용 수정 기능
- 그래픽 처리에 의한 GDMO의 문법화된 수정 및 보기
- 새로운 템플릿의 생성 : 존재하는 템플릿의 마우스 조작에 의한 재사용 및 수정
- 관리 객체 클래스에 상속 관계의 보기
- 포함트리(Containment tree)의 보기
- 그래픽 기록 트리와 연관된 기록 브라우저를 통한 GDMO 정의의 기록
- 브라우저에 작성된 GDMO를 컴파일하여 스키마 테이블의 형태로 데이터베이스에 저장

(그림 3)는 GDMO 편집기의 주 메뉴 화면이고 (그림 4)는 9개 템플릿화면 중에서 Attribute 템플릿을 보여주고 있다. 데이타베이스 선택이 완료되어 사용자가 여러 작업들을 수행하게 하는 주화면이 제시된다. 주화면의 구성은 그림 3에서 볼 수 있는 것처럼 상단의 메뉴, 9개의 템플릿을 작성할 수 있는 선택 버튼, 그리고 텍스트 에디터를 호출할 수 있는 버튼과, 포함 및 상속 트리를 보여 주는 버튼으로 구성된다.

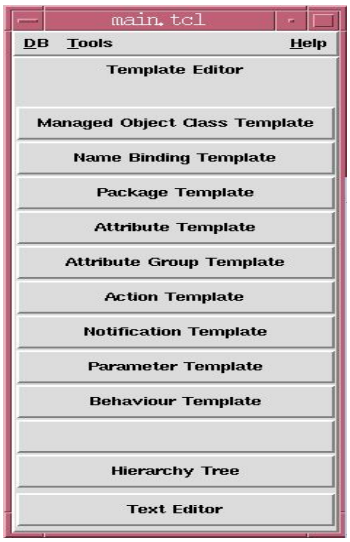

그림 3. 편집기의 주화면
Figure 3. Main View of Editor

그림 4. 어트리뷰트 템플릿 브라우저
Figure 4. Attribute Template Browser

9개의 템플릿 중에 Attribute 템플릿은 그림 4와 같다. Attribute 템플릿은 각각의 Attribute 템플릿의 속성을 정의할 수 있도록 해 준다. GDMO 편집기의 내부구조는 (그림 5)와 같다.

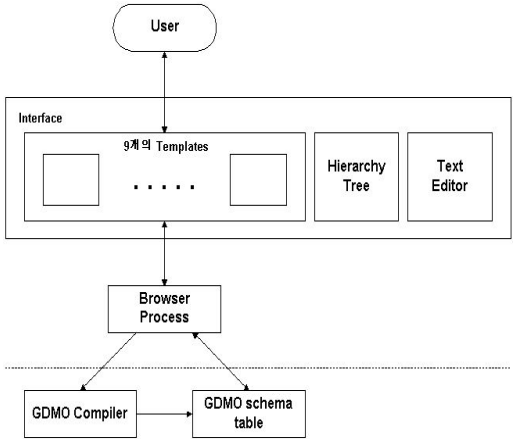

그림 5. GDMO 편집기의 내부구조
Figure 5. Internal Structure of GDMO Editor

인터페이스는 9개의 템플릿 브라우저 및 상속, 포함 트리 브라우저와 텍스트 에디터를 가지고 있다. 사용자는 이러한 브라우저를 통하여 GDMO 명세를 입력하고 브라우저 프로세스는 이러한 입력을 처리하여 중간 스키마 테이블의 형태로 변역하는 과정을 거치게 된다. 편집기는 찾기 기능을 가지고 있다. 템플릿의 이름이나 템플릿의 종류를 선택하면 GDMO 스키마 테이블의 내용을 템플릿 브라우저로 표현하여 준다.

4.2 GDMO 컴파일러

(그림 6)은 GDMO 컴파일러를 묘사한 것이다. GDMO 컴파일러는 텍스트로 작성된 GDMO 명세 문법을 검사하여 템플릿을 스키마 테이블로 바꾸어 주는 역할을 한다. GDMO 명세를 스키마 테이블로 바꾸기 위해 어휘 분석 및 구문 분석 단계를 거쳐야 한다. 어휘분석[3]은 GDMO의 9가지 템플릿의 정의를 lex를 사용하여 토큰을 생성하고 yacc을 사용하여 구문 분석 및 의미분석[3]을 거친 후 스키마 테이블의 형태로 번역한다.

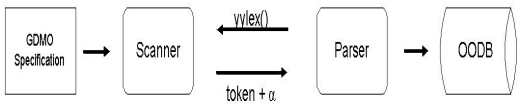

그림 6. 어휘 분석기 및 구문 분석기
Figure 6. Lexical Analyzer and Syntax Analyzer

```
definition  : classdef  { FREE($1);}
            | packagedef    { FREE($1);}
            | parameterdef  { FREE($1);}
            | attributedef  { FREE($1);}
            | groupdef  { FREE($1);}
            | actiondef     { FREE($1);}
            | notificationdef   { FREE($1);}
            | behaviourdef  { FREE($1);}
            | name_binding
            ;
classdef    : labelString
{
New_Template(gdmo_odb, mo_class, $1, MOC_MAP);
}
MANAGED OBJECT CLASS
derivedef characterizedef conditionaldef
registration SEMICOLON
{
  ooRef(ObjectIdentifierValue)      oidVal;
  OOOIDSET(oidVal, ($9));
   mo_class->objectIdentifier = oidVal;
   free($9);
   if(Find_Table($1)) Delete_Table($1); $$ = $1;
}
;
```

그림 7. 문법과 의미분석
Figure 7. Grammar and Semantic Analyzer

컴파일러 구성에서 어휘분석은 첫 번째 단계로서 GDMO 명세을 입력으로 받아 일련의 토큰들로 나누는 일을 한다. 어휘 분석기는 구문 분석기의 필요에 따라 수시로 호출되는데, 이때 한 개의 토큰과 부수적인 정보(토큰값)를 호출한 구문 분석기로 전달한다. 토큰의 종류에 따라서는 토큰의 스트링을 DDL 클래스에 저장되어진다. 어휘 분석기는 GDMO 명세의 한 토큰을 인식하고 이를 미리 약속한 숫자로 된 내부적인 표현으로 변환한다. 구문 분석기는 GDMO 문법의 각 규칙에 해당하는 문법 지향 정의를 작성하여 GDMO 스키마 테이블로 출력하도록 설계·구현되었다. 그림 7은 yacc으로 작성된 GDMO 문법의 일부를 묘사하고 있다.

4.3 GDMO 스키마 테이블

GDMO 템플릿 스키마 테이블은 GDMO 컴파일러에 의해 컴파일된 문법 정보를 데이터베이스에 저장하기 위한 구조이다. GDMO 컴파일러가 생성하는 중간 자료 구조는 객체지향 형태로 되어있다. 그래서 이러한 구조를 저장하는 스키마 테이블의 형태는 OODB가 적당하다. OODB를 사용하면 자료 접근시 인터페이스가 쉽고, 객체지향 파라다임

의 특성을 가진다는 장점이 있다.

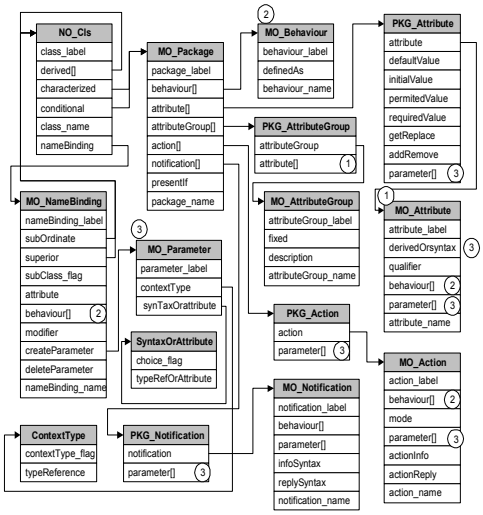

그림 8. 템플릿의 중간 구조
Figure 8. Intermediate Structure of Template

(그림 8)은 9개 템플릿의 중간 구조의 연관성을 보여주고 있다. 각 클래스에 정의된 자료 구조는 다른 클래스를 자신의 멤버로 사용하고 있음을 알 수 있다. 데이터베이스 객체 모델을 설계할 때 중간 자료 구조를 정의하는데, 중간 자료 구조의 형태는 C++ 언어의 클래스와 유사하다. 이를 위해 Objectivity의 DDL(Data Definition Language)을 사용하여 스키마 데이블을 구성하였다.[1] 9개의 템플릿에 해당하는 중간구조는 클래스 형태인 DDL로 구성된다. 이렇게 구성된 클래스는 GDMO 번역기와 브라우저에 의해 필요한 정보를 추출하여 DDL의 인스턴스에 삽입한다. 간단한 예로 관리 객체 템플릿은 그림 9의 형태로 구성되며, 관리 객체 템플릿을 위한 중간 구조의 DDL 클래스는 그림 10과 같다.

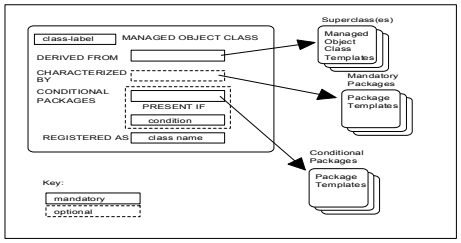

그림 9. 관리 객체 클래스의 구조
Figure 9. Structure of Managed Object Class

```
class MO_Cls : public ooObj {
private:
public:
  ooVString          template_label;
  ooVString          derivedfromString;
  inline ooRef(MO_Cls)derived[] : copy(delete);
  inline ooRef(MO_Package) charBy[] : copy(delete);
  inline ooRef(MO_Package) condition[] : copy(delete);
  ooRef(ObjectIdentifierValue)      objectIdentifier;
  ooVString          comment;

  inline ooRef(MO_NameBinding) nameBinding[] :
                                          copy(delete);

  MO_Cls()    {  }
  ~MO_Cls()   {  }
  char* template_labelP()      {
          return (template_label.head());
          }
  void PrettyP(int indent = 0);
  void CID_GEN(FILE *fp);
};
```

그림 10. MO 클래스 정의
Figure 10. MO_Class Definition

MO 클래스는 관리 객체 템플릿을 표현하는 중간 구조이다. 이 클래스의 template_label은 관리 객체 템플릿의 class-label의 정보를 가지며, derived는 배열 구조로서 상속되는 MO의 정보를 포함하고 있다.

MO_Cls 클래스의 charBy와 condition 배열 구조는 CHARACTERIZED BY 문장과 CONDITIONAL PACKAGES에서 상속되는 템플릿을 표현하는 것이다. nameBinding 배열은 nameBinding 템플릿의 정보를 가지도록 설계되었다. objectidentifier는 REGISTED AS 문장의 정보를 추출하여 보관한다. 이러한 형태로 9개의 템플릿의 중간 구조가 형성된다.

4.4 원시코드 생성기

MIB 클래스들이 하나의 객체로 인스턴스화 되었을 때 외부에 보여지는 인스턴스의 타입은 최고 상위의 클래스로 일반화하여 부여 주어야 한다. 즉, 임의의 인스턴스에 정의된 인터페이스를 통해서 오퍼레이션이 이루어지기 위해서는 모든 클래스의 인터페이스가 같거나, 클래스의 종류에 따라 오퍼레이션을 구분할 수 있는 방법이 있어야 한다. 일반적인 방법으로 최상위의 클래스 인터페이스로 일반화가 이루어지므로 상속받는 하위의 클래스에도 인터페이스가 계승된

다. 그러나 각각의 클래스가 갖고 있는 속성이 틀리기 때문에 메소드는 특성화가 되어야 하며, 이는 멤버 함수에서 특성화된다. 일반화된 인터페이스는 가상 함수로 정의되고, 각 MOC에서 정의된 관리 오퍼레이션 서비스 기능을 제공한다. 관리 오퍼레이션 서비스는 ITU-T X.710에서 정의되는 기능으로 객체간의 상호작용이 이루어지는 인터페이스(외부의 객체에 보여지는 인터페이스가 필요하다)가 필요하다. 이러한 인터페이스는 CMIP M-operation을 가능하게 하는 기능으로 다음과 같다.

- M-GET 오퍼레이션 기능을 지원하기 위해서 객체의 각 속성을 얻을 수 있는 멤버 함수
- M-SET 오퍼레이션 기능을 지원하기 위해서 객체의 각 속성을 결정할 수 있는 멤버 함수
- M-ACTION 오퍼레이션 기능을 지원하기 위해서 ACTION을 실행하는 객체의 멤버 함수
- M-NOTIFICATION 오퍼레이션 기능을 지원하기 위해서 NOTIFICATION을 실행하는 객체의 멤버 함수
- M-CREATE 오퍼레이션 기능을 지원하기 위해서 CREATE을 실해하는 객체의 멤버 함수
- M-DELETE 오퍼레이션 기능을 지원하기 위해서 DELETE을 실행하는 객체의 멤버 함수

GDMO 코드 생성기에 의해 생성되는 코드는 객체지향 파라다임을 제공하는 어떠한 언어로도 번역될 수 있다. 그러나 이렇게 하기 위해서는 각 언어로 번역하는데 필요한 번역 규칙이 필요한다. 본 논문에서는 객체지향 언어인 C++로 변화하는데 중점을 두어 C++ 번역 규칙을 작성하였다.

GDMO를 C++ 클래스로 변환하기 위한 기본적인 원칙은 GDMO 템플릿를 하나의 C++ 클래스로 변환하는 것이다. 템플릿 스키마 데이터베이스에 들어있는 각 템플릿의 값을 그림 1에서와 같이 GDMO 스키마 테이블의 형태로 가져오고 이를 원시코드 생성기가 입력으로 받아 C++ 클래스를 생성한다.

이때 데이터베이스에 들어있는 GDMO 중간 구조의 정보를 얻어 C++ 클래스로 번역하는 규칙이 필요하다. 이러한 규칙은 다음과 같은 원칙에 의해 작성되어진다.

C++ 클래스로 변형된 인터페이스는 프로그래머에게 관리 객체의 behavior를 구현하고 접근할 수 있어야 하고 CMIS 서비스를 제공하는 객체지향 API가 제공되어야 한다. ASN.1에 의해 표현되는 GDMO API에 많은 부분이 종속적이 된다. 가능하면 C++ 클래스는 ASN.1 타입을 프로그래머에게 은익되어져야 한다.

다음 (그림 11)은 관리 객체의 번역 규칙을 표현한 것이다. 이러한 번역 규칙은 각 템플릿에 각각 다르게 작성되어져 있다. 이러한 번역 규칙은 다음과 같은 원칙에 의해 작성하였다.

```
class [PREFIX] class-label [POSTFIX]
        : public derived-class-label ... {
public :
[PREFIX] class-label [POSTFIX]();
~[PREFIX] class-label [POSTFIX]();
ERRORS get (AttributeId *attributeId,
        Attribute *result);
ERRORS set (ModifyOperator *modifyOperator,
    AttributeId *attributeId, ANY *attributeValue);
ERRORS action (ActionInfo *actionInfo,
        ActionReply *actionReply);
ERRORS create(Attribute *attribute);
ERRORS [Pre-Condition] Create-CB();
ERRORS [Post-Condition] Create-CB();
ERRORS [Pre-Condition] Delete-CB();
ERRORS [Post-Condition] Delete-CB();
ERRORS notification(
        EventReportArgument *eventReportArgument,
        EventReportResult *eventReportResult);
ERRORS Possible2Del();
ERRORS Get-AVA (AttributeValueAssertion *value);
ERRORS Get-IdList(AttributeIdList *oidList,
        int *count);
int Get-Cnt();
[PREFIX] class-label [POSTFIX] *Clone();
void Copy([PREFIX] class-label [POSTFIX] *value);
ERRORS Print(FILE *fp, int indent);
ERRORS filter(FilterFlag filte_flag, AttributeId *oid,
        ANY *value, int &result);
int Cmp-nameBinding-label(
        NameBinding *nameBinding);
Class [PREFIX]Package-label[POSTFIX]
        package-label;
other Package Template ...

Attribute-Syntax *Get-attribute-label();
ERRORS Get-attribute-label(Attribute-Syntax *value);
ERRORS Set-attribute-label(Attribute-Syntax *value);
ERRORS action-label(ActionInfo*info,
        ActionReply *reply);
        other Action Template ...
}
```

그림 11. 관리 객체의 번역 규칙
Figure 11. Exchange Rule of Managed Object

- 사용하기 편리한 프로그래밍 인터페이스를 제공한다.
- OSI 관리 모델 기능을 지원하는 추상화된 계층을 제공한다.
- 통신의 복잡성을 감춘다.
- 관리 어프리케이션의 점진적인 개발이 가능하다.
- GDMO 명세의 모든 정보를 포함한다.

V. 구현 환경

본 구현은 SunOS 5.5에서 구현되었으며 SPARC compiler version 4.0.1 C++를 사용하였다. 본 구현의 중간 자료 구조를 저장하는 데이터베이스는 객체지향 데이터베이스인 Objectivity를 사용하였으며, 중간 구조를 정의하기 위해 Objectivity의 DDL를 사용하였다. 구문 분석기에는 UNIX의 yacc을 이용하였으며, GDMO 브라우저에는 Tcl/Tk[2]를 사용하였다.

VI. 결론 및 향후과제

망 관리 정보 모델에서 신뢰성있는 망 기능을 제공하기 위해서는 TMN 기능을 지원하는 정형화된 소프트웨어 기본 구조의 연구와 망 관리 정보모델을 기반으로 하는 자동화된 망 관리 원시코드 발생기가 필요하다. 따라서 본 논문은 망 관리 시스템에서 GDMO의 프로그램 개발환경를 구현하였다. 사용자 인터페이스를 강화하기 위해 다양한 그래픽 브라우저를 구현하였으며, 객체지향 파라다임을 가지는 GDMO 스키마 테이블을 설계하고 구현하였다. 또한 GDMO를 다양한 객체지향 언어로 번역할 수 있는 원시 코드 자동 생성기를 구현하였다.

향후에는 GDMO 개발 환경상에서 객체지향 언어의 생성에 필요한 여러 단계를 최적화하여 효율을 향상시킬 계획이다.

참고문헌

[1] Objectivity/DB C++ Application Development : Version 3, Objectivity Inc, 1995.

[2] John K. Ousterhout, "Tcl and the Tk Toolkit," Addison-Wesley, 1994.

[3] Thomas W. Parsons, Introduction to Compiler Construction, Computer Science Press, 1992.

[4] H. F. Korth, A. Silberschatz, Database System Concepts, McGRAW-Hill, 1991.

[5] Stroustrup, Bjarne. The C++ Programming Language. 2nd ed. Addison-Wesley, Reading, Mass., 1991.

[6] "TMN C++: GDMO++, Managed Object Application Programming Interface," NM Forum, 1996.

[7] ITU-TS Recommendation X.700, Commaon Management Information Service Definition for CCITT Application, ITU, Geneva, 1991..

[8] D. Streedman. ASN.1 The Tutorial and Reference.1990.

[9] C. Huitema, General Presentation of the MAVROS Compiler ,INRIA, 1990, available via at http://www-sop.inria. fr/ rodeo/mavros /mavros-home.html.

[10] IOS developers, BBN Systems and Technology ASN.1 Compiler version 1.4, 1995, available via at http://ests.bbn.com/ASNSRC.html.

[11] M. Sample, Snacc ASN.1 Compiler version 1.2, 1997, available via at http://www.fokus. gmd.de/ovma/mug/archives/mug-software/ snacc.html.

[12] ISODE, 1997, available via at http: //ftp.doc. ic.ac.uk/packages/isode.

[13] N. Drakos, CASN1-ASN.1 to C Compiler, 1995, available via at http://www.atri.curtin. e.edu.au/~duke/ honours/compiler /casn1/casn1.html.

저 자 소 개

정 진 영

1992년 한남대학교 전자계산학과 졸업(공학사)
1994년 한남대학교 대학원 컴퓨터 공학과(공학석사)
2002년 한남대학교 대학원 컴퓨터 공학과(공학박사)
1997년 ~ 현재
대전보건대학 멀티미디어과 조교수
<관심분야> 멀티미디어 문서처리(XML), 객체지향모델링 및 방법론, 분산시스템 및 실시간시스템 등

오 대 균

1984년 숭전대학교 경영학과(경영학사)
1994년 한남대학교 대학원 컴퓨터 공학과(공학석사)
2002년 한남대학교 대학원 컴퓨터 공학과(공학박사)
1979년 ~ 현재
한국에너지기술연구원 재직
2004년 멀티미디어 기술사 취득
<관심분야> XML, 실시간 처리시스템, DMB

김 영 철

1990년 한남대학교 컴퓨터공학과 (공학사)
1996년 숭실대학교 대학원 전자계산학과(공학석사)
2003년 8월 숭실대학교 대학원 컴퓨터학과(공학박사)
2002년~현재
(주)뉴스텍 시스템 이사,
명지전문대학 겸임
<관심분야> (웹)프로그래밍 언어, 컴파일러, 컴퓨터 통신, XML, 망관리