韓國 컴퓨터情報學會 論文誌
第8卷 第4號, 2003. 12.

2003-8-4-1-5

# ASN.1 원시 코드 자동 생성기

정 진 영*, 김 영 철**

# ASN.1 Source Code Auto-Generator

Jin-Young Jung*, Young-Chul Kim**

## 요 약

ASN.1은 망 관리에 필요한 기초적인 제반기술이다. ASN.1의 개발에는 ASN.1 명세 언어를 파싱하는 컴파일러 작업과 컴파일 결과 생성된 자료들을 DB에 입력하고, 입력된 자료를 사용자에게 프리티프린팅하여 보여주는 작업이 요구된다. 본 논문에서는 ASN.1 명세를 객체지향 언어인 C++로 자동적으로 변환하여 주는 원시 코드 자동 생성기를 설계하고 구현한다. 이와 함께 ASN.1 개발환경에 필요한 그래픽 사용자 인터페이스, DB 인터페이스 및 ASN.1 브라우저를 포함하는 통합 환경을 제공한다. 본 시스템의 구현은 Objectivity 데이타베이스를 이용하였고, 컴파일러 작업에서는 컴파일러 보조 도구인 flex와 byacc을 이용하였으며, 인터페이스 언어로는 Tcl/Tk를 사용하였다.

▸ Keyword : ASN.1, 망 관리, 원시 코드 생성기, ASN.1 브라우저

## Abstract

ASN.1 is the most fundamental technology in network management. The develoment of ASN.1 environment requires three steps : compiler work to parse ASN.1 languages, updating database with the parsing results, and pretty-printing work for data in the database. This paper presents the design and implementation of the translator which automatically translates the specification of ASN.1 to the object-oriented language C++. This system provides a total environment including for various graphic user interface, DB interface, browser to develop ASN.1 for development environment of ASN.1. For the implementation, Objectivity DB is used for database, flex and yacc for compiling, and Tcl/Tk for user interface.

▸ Keyword : ASN.1, Network Management, Source Code Generator, ASN.1 Browser

* 대전보건대학 멀티미디어소프트웨어과 조교수
** 숭실대학교 시스템 소프트웨어 공학박사

# I. 서 론

망 관리를 위해서는 다양한 종류의 운영체제와 통신장비 간의 정보 교환이 필수적이다. 그러나 다양한 하드웨어와 공급업체간의 호환성이 결여되어 있으므로 통합된 망 관리 작업은 용이하지 않다.

이러한 상황을 극복하고자 하는 노력으로 TMN (Telecommunications Management Network)이 등장하였다[1, 2]. TMN은 다양한 종류의 운영체제와 통신 장비 사이에 표준화된 인터페이스를 이용하여 정보의 교환이 이루어지도록 한다. 또한 TMN은 통신망 관리를 위한 구조적인 프레임워크를 제공하여 일반적인 정보모델과 표준 인터페이스를 이용하여 여러 종류의 장비에 대한 관리를 수행한다. 이 체계하에서는 인터페이스와 행위를 표준화된 형태로 정의하고 관리할 수 있는 도구가 요구되며, 이를 위해서 ASN.1(Abstract Syntax Notation One, X.208에서 정의되었음)이 표준으로 제정되었다 [3].

ASN.1 컴파일러는 ASN.1 언어 명세를 C 언어 등으로 짜여진 암호(encoding)와 해독(decoding) 루틴으로 바꾸어주는 작업이다. 이러한 루틴들은 OSI 응용프로그램을 개발하는데 이용될 수 있으며, 비 OSI 통신 프로토콜을 구현하기 위해서도 사용될 수 있다. ASN.1으로 정의된 타입을 프로그래밍 언어(C, C++) 등으로 변환하는 도구를 개발함으로써, 표준 데이터 타입(data type)을 처리하기 위한 인터페이스를 표준화할 수 있다. 또한 C++ 언어를 사용하여 객체 지향적인 개념을 충분히 이용할 수 있다. 즉, 단일화 된 유틸리티 함수만을 외부에 보여지도록 인터페이스를 정의함으로써, 상세 내용을 캡슐화시켜고 모듈의 복잡도를 줄일 수 있다. 더 나아가 ASN.1 컴파일러 기초 기술을 확고히 다짐으로써 망 관리 정보 모델을 자동화할 수 있는 제반 기술을 획득할 수 있다.

본 논문의 목적은 다음과 같다. 첫째는 ASN.1 컴파일러 기초 기술을 확고히 다짐으로써 망 관리 정보 모델을 자동화할 수 있는 제반 기술을 획득하는데 있다. 둘째는 망관리를 위해서 다양한 종류의 운영체제와 통신 장비 사이에 표준화된 인터페이스를 이용하여 정보 교환을 용이하도록 하고, 마지막으로 통신망 관리를 위한 구조적인 프레임 워크를 제공해 주기위하여 인터페이스와 행위를 표준화된 형태로 정의하고 관리할 수 있는 도구를 개발하는데 목적을 두고 있다.

본 논문의 구성은 다음과 같다. 제 2장에서는 ASN.1의 컴파일러 연구 사례에 대해서 언급한다. 제 3장에서는 망 관리를 위한 기본 제반 지식인 ASN.1 개발 환경에 대해 설명한다. 4장에서는 구현 환경 및 평가에 대해서 설명하고, 마지막으로 5장에서는 결론 및 향후 연구 방향에 대해서 설명한다.

# II. 연구사례 및 배경

망 관리를 위한 ASN.1 개발 환경에 대한 연구는 ASN.1 표현 방법에 관한 표준이 정의되면서부터 이루어져 왔다. 지금까지 개발된 ASN.1 컴파일러들은 대부분 DB와 연동되지 않고 있으며, 단지 구문분석과 의미분석에 의한 다른 언어로의 변환기 역할만을 수행하고 있다. 기존의 ASN.1 컴파일러에는 MAVROS [4], BBN [5], Snacc [6], ISODE의 PEPY/POSY [7], UBC의 CASN1 [8] 등이 있다.

MAVROS는 크게 두 부분으로 구성되어 있다. 하나는 전처리기로 모든 ASN.1 명세의 형을 지원하며, 모든 ASN.1 값들을 초기화하는 단계이다. 두 번째 부분은 생성기 부분으로 여러 다양한 인코딩 루틴들을 생성하며, 프로그래밍 디버깅 도구도 지원한다. BBN 컴파일러는 ASN.1 명세를 C++로 변환할 수 있으며, Solaris, MSDOS, Win3.1에 이식이 가능하다. 이 컴파일러는 자동 구문 검사 기능이 있다. Snacc은 ASN.1을 C나 C++ 코드로 바꿀 수 있는 변환기이며, GUI를 지원한다. CASN1 ASN.1 컴파일러는 3 단계 컴파일러이다.

1, 2 단계는 3 단계를 위한 전처리기를 위한 함수로 구성되어 있으며, 3단계의 입력이 된다. 이때 생성되는 파일은 "out1"과 "out2" 파일이다. 이 컴파일러의 제 1단계에서는 복합 형 정의 및 이름이 할당된다. 제 2단계에서는 ASN.1 명세를 상향식(bottom-up) 순서로 형을 정의하였으며, 결과 형들은 C 언어의 구조와 같은 구조로 바꾸는 단계이다. 제 3단계에서는 오류 회복과 C 코드를 생성한다.

그러나 이러한 기존의 컴파일러들은 특정한 컴퓨팅 환경에서만 동작하므로 이식성이 약하다. 또한 이 컴파일러에 의해 생성된 결과물은 특정 목적에 의존하고 있기 때문에 이를 이용하는 사용자나 설계자, 관리자들은 자신의 목적에 맞게 처리할 수 없다. 따라서 자신의 요구 사항에 맞추어 사용하기 위해서는 컴파일러의 결과물을 변환시켜야 하는데, ASN.1의 복잡한 자료 구조 때문에, 이 작업은 매우 어렵다. 무엇보다도 가장 큰 문제점은 기존의 컴파일러들이 대부분 1994년 개정된 ASN.1 표준안 이전에 개발되었기 때문에 많은 기능을 변경하여야 한다. 따라서 지금까지 개발된 컴파일러들은 객체(GDMO)를 처리하지 못하기 때문에 관리에 많은 어려움이 있다. 따라서 본 논문에서는 객체를 이용하여 관리하기 위하여 원시 코드인 C++ 언어로 변환함으로써, 쉽게 객체를 관리할 수 있도록 한다.

## III. ASN.1 개발 환경

다음 〈그림 1〉은 본 논문에서 구현한 ASN.1 개발 환경의 구성을 보여준다. 그림 1에서 보여주는 것과 같이 사용자는 ASN.1 명세 언어와 응용프로그램을 조작할 수 있다. 즉, 궁극적으로 ASN.1 개발 환경은 망 관리에 필요한 응용프로그램이나 정보를 생성하는데 있다.

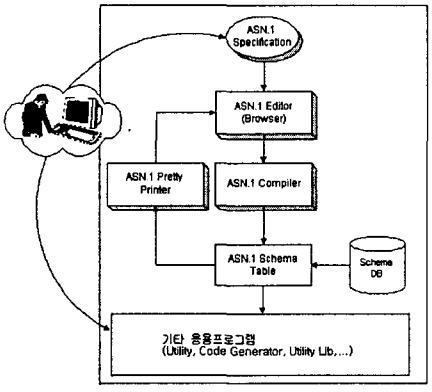


그림 1. ASN.1 컴파일러 개발환경
(figure 1. ASN.1 Compiler Development Environment)

### 1. ASN.1 컴파일러 개발환경

#### 1.1. 편집기(ASN.1 Editor)

ASN.1 편집기는 ASN.1 명세를 생성, 수정, 삭제 등의 편집을 위한 통합된 환경을 제공한다.

#### 1.2. ASN.1 Pretty Printer

프리티프린팅은 DB 스키마 테이블에 있는 내용들을 조회하여, 적절한 indent를 사용하여 형식화된 text 형태로 사용자에게 보여줌으로써 시각적인 효과를 주었다. 이러한 과정은 DB에 자료를 입력할 때 "mapping"의 과정을 통하여 이루어진다.

#### 1.3. ASN.1 컴파일러(ASN.1 Compiler)

ASN.1 개발환경에서 컴파일러 작업은 무엇보다 중요하다. 컴파일러 작업에 필요한 요소들은 ASN.1 명세를 파싱하고, 생성된 자료들을 DB에 입력하는 작업을 한다.

#### 1.4. ASN.1 스키마 테이블

스키마 테이블은 ASN.1 명세로 정의된 ASN.1 형과 값을 표현하는 데이터 형으로 ASN.1 편집기 및 브라우저에 정보를 지원할 수 있는 내부적인 자료 구조이다. 스키마 테이블의 구조는 ASN.1 도구를 구성하는 가장 기본적인 구조로 본 구현에서는 ASN.1 내부 자료 구조와 같은 형식으로 구성하였다.

스키마 테이블은 ASN.1 명세의 컴파일이 수행된 후 삽입 과정을 통하여 구성된다. 스키마 테이블은 모듈, 타입, 값 테이블로 이루어져 있다. 특히, 값 스키마는 ASN.1 값을 표현하는 데이터 형으로 응용 프로그램에서 이용할 수 있다. 본 시스템에서 구현한 스키마 테이블은 앞절에서 설명한 내부 자료 구조의 형식과 같다. 단지 차이점이라면 Objectivity DB [18]에서 허용하는 형식으로 형을 바꾸어 주는 작업이 필요하다. 예를 들면, 3.3절에서 보여준 모듈 자료 구조에 대한 스키마 구조는 다음과 같다.

```
class dModule : public ooContObj {
  public :
    ooVString          dname;
    ooRef(dOidValue) dmodule_OID_number;
      dTAG_STYLE   dtag_style;
      ooRef(dExport_Import)          dexport_list;
      ooRef(dExport_Import)          dimport_list;
      ooRef(dModule)  dimport_parent;
      ooRef(dAssignment)             dassign;
      ooRef(dTypeId)   dtype_id;
      ooRef(dValueId)  dvalue_id;
      ooRef(dModule)  dnext;
}
```

Objectivity DB에서의 문자열은 "ooVString"으로 기술되며, 객체지향 언어인 C++을 지원한다. 따라서 본 시스템의 구현은 내부 자료 구조에서 정의한 구조를 Objectivity DB에서 허용되는 언어로 바꾸어 주는 작업만이 필요하다.

## 2. ASN.1 컴파일러(ASN.1 Compiler)

### 2.1 어휘분석과 구문분석

ASN.1 컴파일러를 구현하기 위해서는 많은 컴파일러 도구들이 필요하다. 대부분 이용되는 컴파일러 도구들은 어휘분석 도구로서 flex[9]를 이용하며, 구문분석을 위해서 yacc을 이용한다[10].

〈그림 2〉는 ASN.1 컴파일러에 대한 환경을 보여준다.

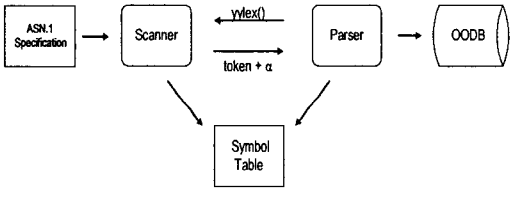


그림 2. 어휘분석기와 구문분석기
(figure 2. Lexical Analysis and Syntax Analysis)

일반적으로 lex의 많은 버전들은 제한된 토큰 크기를 가지고 있다. ASN.1에서는 따옴표(" ")로 이루어진 문자열에서와 같이 매우 긴 토큰을 가질 수 있으므로 사용할 수 없다. 그러나 GNU의 flex는 토큰 저장을 위한 메모리 할당을 동적으로 수행하므로 이러한 제약이 없기 때문에 본 시스템 구현에 flex를 이용하였다.

〈그림 2〉의 의미 분석기와 구문 분석기 사이의 "token + a"에서 "a"는 위에서 설명한 가상 토큰에 해당된다. 또한 심볼 테이블은 위에서 설명한 "cross-reference"를 해결하기 위한 방법이다. 또한 의미 분석과정은 내부적으로 모듈, 형, 서브형, 값 등 필요한 모든 정보를 파서가 파싱하면서 DB에 입력한다. 이때 사용되는 기술 중 하나가 "매핑" 기능이다.

본 시스템의 구문 분석 과정에서 이용한 문법은 1990년에 제정된 x.208 규약에 의한 문법을 기준으로 하였다. 표준으로 정의된 문법은 구문 분석기의 구성에 사용되는 yacc과 같은 도구를 사용하기에는 부적합하므로 정의된 문법을 동일한 부류의 구문의 LALR 문법으로 변환하였다.

문법의 변환 과정은 기본적인 관계 표현을 적용함으로써 변환이 가능하며, 다음과 같은 관계 연산자를 적용하였다.

- 분해 : 분해 과정은 문법 생성 규칙에서 "prefix", "suffix", "bifix"의 형태를 가지며, 다음과 같이 변환된다. 이러한 분해 과정은 부분적으로 적용될 수 있다.
  - prefix : X → βγ1 | βγ2 | ... | βγn | B는 X → βY | B와 Y → γ1 | γ2 | ... | γn이 된다.
  - suffix : X → α1β |α2β | ... |αnβ | B는 X → Yβ | B와 Y → α1 |α2 | ... |αn이 된다.
  - bifix : X → αβ1γ | αβ2γ | ... | αβnγ | B는 X → α Yγ | B와 Y → β1 | β2 | ... |βn 이 된다.
- 축소 : 축소 과정은 문법 생성 규칙에서 같은 규칙을 합치고, 새로운 비단말 기호를 만들어내는 과정이다.
  - X → α1 |α2 | ... |αn은 X → Y | αi+1 |αi+2 | ... |αn과 Y → α1 |α2 | ... |αi가 된다.
- 제거 : 제거 과정은 문법을 전체적으로 확장한 후에 필요없는 규칙을 제거하는 과정이다. 제거하는 과정에서 주의할 점은 문법 규칙의 "의미"를 변환시켜서는 안 된다. 예를 들면문법 규칙 X → X | A는 X → A로 제거될 수 없으며, X → lowerval | lowerid는 X → lower가 될 수 있다.
- 재명명 : 모든 문법 규칙에서 판독성을 위하여 문법 규칙의 식별자를 재명명한다.
- 확장 : 문법 규칙의 확장 과정은 위의 4가지 규칙을 적용하여 전체적으로 확장될 수 있다.
  예를 들면 다음과 같은 문법 규칙 A → a B C, B → b A, B → b, C → c는 A → a b A C, A → a b C, A → B, B → b A, C → c로 확장될 수 있다.

본 시스템은 이러한 변환 과정을 ASN.1 문법 규칙에 적용함으로써 yacc에서 이용할 수 있는 LALR 문법으로 변환하였다.

또한 충돌이 일어났을 경우에도 위와 같은 5가지 관계 표현을 적용함으로써 문법의 "의미"를 변환시키지 않고 해결하였다. 이러한 변환 과정은 모듈, 형, 값 등의 크게 3 부분에 부분적으로 적용하였다.

### 2.2. 스키마테이블과 내부자료 구조

다음은 본 논문에서 구현에 이용된 모듈에 관한 자료 구조를 나타낸다.

```
typedef struct module {
    aString    name;
        struct oidValue          *module_OID_number;
        TAG_STYLE tag_style;
        struct export_import     *export_list;
        struct export_import     *import_list;
        struct module            *import_parent;
        struct assignment        *assign;
        struct typeId            *type_id;
        struct valueId           *value_id;
        struct module            *next;
    } aModule;
```

모듈에 관한 자료 구조의 필드들은 순서대로 모듈의 이름을 가리키는 필드, 모듈에 관한 OID를 가리키는 필드, 태그를 포함하는 필드, exports와 imports를 나타내는 필드, 형, 값을 가리키는 필드로 구성되어 있다. 마지막 필드인 next는 한 파일에 둘 이상의 모듈을 가지고 있을 경우를 처리하기 위해 필요하다.

내부 자료구조를 달리 사용하게 된다면 보다 복잡한 자료구조가 될 것이다. 본 시스템에서 이러한 자료구조를 이용한 이유는 분명하다. 하나의 모듈은 많은 형과 값을 포함하고 있으며, 외부 모듈과 "*imports*" 및 "*exports*"가 되므로 이들 정보를 포함하고 있어야 한다. 물론 oidValue는 외부 모듈이 어디에서 정의되어 있는 모듈인지를 알려주는 정보이기 때문에 필요하다.

또한 태그가 EXPLICIT나 IMPLICIT일 때 서로 다른 태그가 표현되므로 이들에 관한 정보도 필요하다. TypeId나 ValueId는 다른 oidValue와 구분하기 위해 사용된 자료구조 이름으로 또 다른 형이나 값을 포함할 수 있으므로 모듈에 관한 정보에 삽입하였다. assignment 자료구조는 형과 값이 순서에 상관없이 나올 수 있기 때문에 필요한 자료구조이다.

다음에 설명되는 내용은 ASN.1 컴파일러에 의해 생성되는 테이블의 예이다. 생성된 테이블 내용은 내부적으로 DB 스키마 테이블에 자동적으로 입력되게 된다. 물론 이러한 스키마 자료는 나중에 사용될 응용 프로그램에 사용하게 된다. 또한 ASN.1 브라우저에서는 DB 스키마 테이블에 저장된 자료를 프리티 프린팅하여 사용자에게 보여준다.

**예제**

형에 관한 예 - TypeA ::= SET {p BOOLEAN, q INTEGER, r BIT STRING}

위에서 기술된 예제는 ASN.1 컴파일러에 의해 그림 3과 같은 테이블을 생성한다.

〈그림 3〉에서 보여주는 정보는 형이라면 어떤 형인지(예제의 structure 형중 SET)를 보여주며, 또한 구조체라면 어떤 구성요소들을 가지고 있는지(예제의 BOOLEAN, INTEGER, BIT STRING)를 보여준다.

또한 assignment 자료구조는 형(예제의 첫 번째 필드)인지 값인지를 보여준다. 매핑은 그림에서 첫 번째 TypeId에서 행해지며, 따라서 그 형에 해당되는 나머지 요소들은 추적함으로써 쉽게 알아낼 수 있는 장점을 가지고 있다. 자료구조의 각 번호는 몇 번째 필드인지를 알려주며 그 이상의 의미를 가지고 있지 않다. 이 번호들은 후에 의미 분석할 때 구현자가 쉽게 이용하기 위하여 붙여진 번호이다.

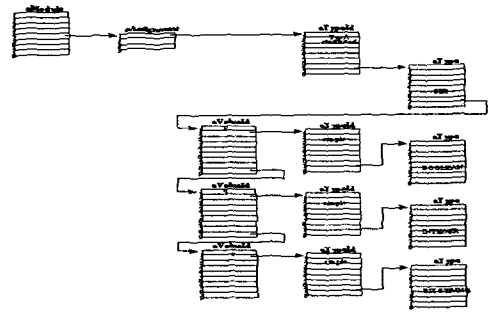

그림 3. Type에 관한 내부 자료 구조
(figure 3. Internal Data Structure for Type)

## Ⅳ. 구현 환경 및 실험

### 1. 구현 환경 및 도구

본 논문에서 사용한 ASN.1 명세 언어의 문법은 표준안에 제정된 문법을 문맥 자유 문법 형태로 바꾸어 사용하였다. 문맥 자유 문법으로 변환하는 과정에서 많은 충돌을 일으켰지만 문법의 의미를 바꾸지 않고 문법을 재기술함으로써 충돌을 해결하였다.

또한 어휘 분석을 위해서 GNU의 flex[9]를 이용하였으며, 구문 분석을 위해서 byacc [10]을 이용하였다. DB와의 인터페이스를 위한 도구로 Tcl/Tk [11]를 이용하였으며, Objectivity DB [12]를 이용하여 Sun Sparc 워크스테이션에서 구현하였다.

## 2. 실험

사용자에게 시각적으로 보여주는 브라우저는 DB에 저장된 ASN.1의 모듈, 형, 값에 관한 내용을 보여준다. 본 시스템의 초기화면은 〈그림 4〉와 같다.

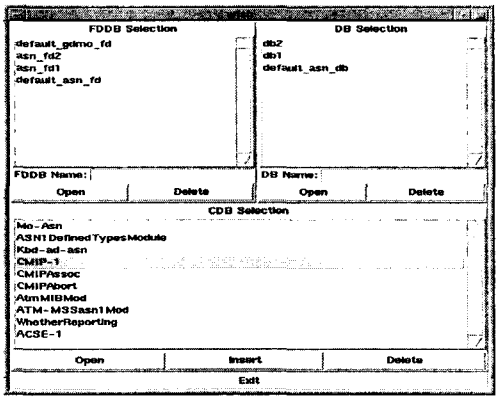


그림 4. ASN.1 브라우저의 초기화면
(figure 4. ASN.1 Browser of Initial Display)

〈그림 4〉는 사용자가 먼저 원하는 DB를 선택하면 이 DB에 포함되어 있는 모듈이 나타난다. 사용자가 원하는 type을 선택하면 해당되는 모듈을 볼 수 있다. 또한, 해당되는 모듈을 선택하면 〈그림 5, 6〉과 같은 브라우져가 나타난다.

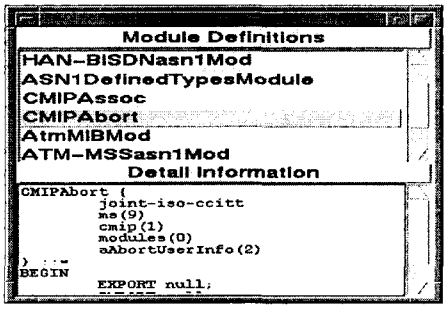


그림 5. Module 브라우저
(figure 5. Module Browser)

〈그림 5〉는 사용자가 선택한 모듈을 DB에서 읽어들여 사용자에게 보여주는 모듈의 리스트와 그 모듈의 상세 정보를 보여준다.

사용자가 모듈 정보를 호출하게 되면 DB로부터 모듈 리스트를 얻어오며, 모듈 리스트에 있는 특정 모듈을 두번 클릭하면 그 모듈의 상세 정보를 다시 DB로부터 읽어 와서 브라우저를 통해 화면에 보여주게 된다.

〈그림 6〉은 사용자가 〈그림 5〉에서 선택한 특정한 모듈의 형과 값에 대한 정보와 상세 정보를 브라우저를 통해 보여준다. 이 과정 역시 모듈에서 설명한 방법과 마찬가지로 이루어진다. 〈그림 6〉의 "search" 버튼과 "CPP" 버튼은 각각 DB를 검색하는 버튼과 해당하는 모듈의 변환된 C++ 코드를 보여주는 버튼이다.

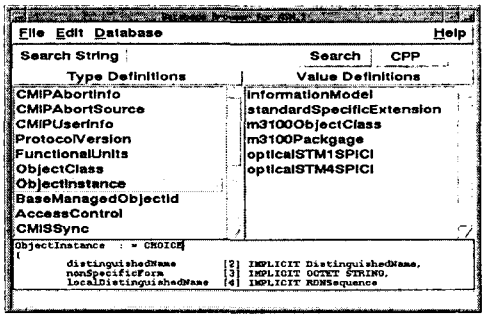


그림 6. Type 및 Value 브라우저
(figure 6. Type and Value Browser)

## V. 결론

본 논문에서는 ASN.1의 명세를 입력으로 받아들여 컴파일을 수행한 후, 컴파일 결과를 DB에 입력하는 ASN.1의 개발 환경에 대하여 설명하였다. ASN.1 컴파일러 작업은 그 명세 언어의 특성상 많은 문제점이 있었으며, 본 논문에서 이러한 문제들의 해결방법을 논의하였다.

ASN.1 환경 개발에서 컴파일러 작업은 많은 작업을 필요로 한다. 본 논문에서는 컴파일러 단계에서 수행하는 어휘분석, 구문분석, mapping 메커니즘 및 DB 스키마 테이블에 대하여 자세히 논의하였다. 특히 mapping 작업은 브라우져에서 pretty printing하는데 중요한 요소임을 논의하였다.

컴파일러 단계에서 수행하는 어휘 분석, 구문 분석 및 DB 스키마 테이블에 대하여 자세히 논의하였다. 또한 생성된 자료를 토대로 변환 규칙을 이용하여 C++ 원시 코드를 생성한 예를 설명하였다. 컴파일한 결과는 DB 스키마 형식의 내부 자료 구조로 변환하여 데이터 베이스에 저장하였다. 또한 DB에 있는 자료를 pretty printing하여 사용자에게 시각적으로 보여지도록 구현하였다.

구현에 이용한 언어로는 Tcl/Tk와 C/C++와 DB 관련 함수들을 이용하였다.

향후에는 본 연구에서 생성된 결과를 토대로 ASN.1 유틸리티를 위한 라이브러리를 구현하고, 망 관리 정보모델의 기능을 위한 원시코드 자동 발생기를 구현하고, DB에 저장된 목적 코드를 이용하여 코드의 단일화 및 정보 교환에 필요한 응용 프로그램도 개발할 예정이다.

## 참고 문헌


[1] Uyless Black, *Network Management standards*, McGraw-Hill, 1994.

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

[3] D. Steedman. ASN.1 The Tutorial and Reference, 1990.

[4] C. Huitema, General Presentation of the MAVROS Compiler ,INRIA, 1990.

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

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

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

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

[9] V. Paxson, On-line manual pages distributed with the Flex software package, 1990, available via anonymous ftp from ftp.ee.lbl.gov or prep.ai.mit.edu.

[10] R. Corbett, BYACC parser generator version 1.9, 1993, available via anonymous ftp.cs.. berkeley.edu:/ucb/4bsd/ byacc.tar.Z.

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

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


## 저 자 소 개


**정 진 영**

1992년 한남대학교 컴퓨터공학과 (공학사)

1994년 한남대학교 컴퓨터공학과 (공학석사)

2002년 한남대학교 컴퓨터공학과 (공학석사)

1997 ~ 현재 대전보건대학 멀티미디어소프트웨어과 조교수

〈관심분야〉 멀티미디어 문서처리(XML), 객체지향 모델링 및 방법론, 분산시스템 및 실시간 시스템 등

**김 영 철**

1990년 한남대학교 컴퓨터공학과 (공학사)

1996년 숭실대학교 대학원 전자계산학과(공학석사)

2003. 8 숭실대학교 대학원 컴퓨터학과(공학박사)

2002 ~ 현재 (주)뉴스텍 시스템 이사, 명지전문대학 겸임

〈관심분야〉 (웹)프로그래밍 언어, 컴파일러, 컴퓨터 통신, XML, 망관리