論 文

SGML(Standardized Genernal Markup Language)에 대한 기본 파서의 구현

正會員 洪 媼 善* 正會員 鄭 會 京* 正會員 李 壽 淵*

Implementation of SGML Basic Parser

On Sun Hong,* Hoe Kyung Jung,* Soo Youn Lee* *Regular Members*

요 약

이 논문은 SGML(Standardized General Markup Larguage)을 적용하여 작성된 SGML 문서와 DTD(Document Type Definition)를 분석하는 SGML 파서(parser)의 설계를 기술한다.

먼저 SGML 규칙을 내포하는 yacc 정의 화일을 구성하였고, 이 화일에 의하여 SGML DTD와 문서를 적합한 토큰(token)으로 파싱하였다. 이 토큰으로 SGML문서의 논리적 구조를 내부 구조화하고, 엔터티(entity) 테이블(tabl), 엘리먼트(element) 테이블등을 생성하여 데이터베이스(database)를 구축하였으며, 이를 이용하여 속성값 입력등의 다음 단계처리에 이용할 수 있게 하였다. 또한 이 파서는 소규모 참조를 변환하여 확장하는 기능도 갖는다. 이 파서에 몇몇 SGML 문서를 시험 적용하여 바르게 시행됨을 확인하였다.

ABSTRACT

This paper describes on implementation of SGML(Standardized General Markup Language) parser, which can analize SGML documents and its DTD(Document Type Definition) defined according to the SGML(ISO 8879).

We have constructed a yacc definition file to present the rules of SGML DTD and documents, by which incoming SGML DTD and documents can be parsed into the appropriate tokens. with the tokens a database with the stuctures such as entity table, element table and so on is built to validate the logical structure of the incoming SGML documents. The additional functions of this parser include the automatic transforming of the incoming documents with the short references into the complete SGML documents. Several test SGML documents have tested to clarify an implementation of this parser and experimental results are satisfactory.

*光云大學校 電子計算機工學科
Dept. of Computer Engineering, Kwangwoon University
論文番號 : 92-50(接受1991. 12. 2)

I. 서 론

각종 워드프로세서(Wordprocessor) 또는 컴퓨터

를 이용한 문서작성 시스템의 보급이 확대됨에 따라, 이기종간에 문서 정보를 교환하는 일이 빈번해 졌다. 하지만 문서에는 윗쪽 여백, 라인(line) 폭, 서체 정보등 페이지(page)에 표현하는 기법에 관한 정보가 포함되는데, 이러한 정보는 기계마다 서로 다르게 코딩(coding)되므로 서로 다른 기종간의 문서 전송에는 어려움이 따른다.

이러한 문제를 해결하기 위하여 ISO에서는 ISO 8879:Information Processing -Text and Office Systems-Standard Generalized Markup Language(SGML)를 국제 표준으로 정하였다. 이 SGML은 서술적 마크업(markup) 개념을 기본으로 하였으며, 마크업 언어나 포맷터(formatter)의 표준이 아닌 일종의 언어 분석용 언어로서 개괄적 마크업 언어의 문법에 관한 표준이다.[7] 또한 SGML을 돕는 국제 표준으로서 SDIF(SGML Document Interchange Format), DSSSL(Document Style Semantics and Specification Language), SPDL(Standard Page Description Language)등이 표준으로 정해지고 있어 SGML의 보급 및 실현을 가속화 시키고 있는 실정이다. 이에 따라 유럽에서는 FORMEX(Formalised Exchange of Electronic Documents)라는 시스템이 유럽출판국에 의해서 개발되었고, 미국에서는 공군관리 매뉴얼을 SGML에 의하여 전송 및 인쇄를 하는 ATOS(Automated Technical Order System) 프로젝트가 1987년에 개발되었으며, 일본에서도 SGML에 대한 지식이 없는 사람도 사용할 수 있는 SGML 문서구조에디터가 연구되었다.[11] [9]

SGML은 엘리먼트라는 논리적 요소를 기본 단위로 하여 문서를 구성한다. 한 엘리먼트는 아랫 레벨(level)의 서브(sub)-엘리먼트를 갖을 수 있으며 문서는 엘리먼트들을 노드(node)로 하는 트리(tree) 구조라 볼 수 있다.[2] [4]

SGML에서 사용할 문서는 문서형 정의(Document Type Definition:DTD)에 문서의 구조가 정의된다. 이 문서의 구조는 같은 종류의 문서들이 갖을 수 있는 공통구조로서 엘리먼트 선언, 엔터티 선언, 속성 선언등이 있으며, SGML 문서가 사용할 수 있는 모든 엘리먼트에 대한 선언이 포함되며, 이것으로 그 문서의 계층적 구조가 파악될 수 있다. 즉 어느 엘리먼트 아래에 속할 수 있는 서브 엘리먼트는 어느것인지, 그 발생 순서는 어떠한지 발생이 선택적인지 의무적인지등에 관한 정보가 엘리먼트선언 안에 포함된다. 속성 선언은 문서의 구조 보다는 개개 엘리먼트가 갖을 수 있는 특성을 정의하기 위한 것으로 해당 문서 요소의 이름과 함께 속성 이름, 속성 값의 형태, 그리고 특정값이 입력되지 않을 경우에 적용될 임의 지정값을 갖는다. 이러한 여러 선언들의 집합인 DTD의 구문이 SGML에 맞는지 검사하고, 엔터티 참조를 확장하는 등의 기능을 갖는 시스템(system)을 DTD 파서(parser)라 한다.[2] [1] [5]

SGML 문서는 그 문서형을 정의한 DTD와 ISO 8879의 마크업 문법에 의하여 작성되어야 한다. SGML 문서의 기본 단위인 한 엘리먼트에 대한 SGML 문법을 보면 시작 태그(tag), 내용, 끝 태그의 셋으로 구분되고, 시작 태그는 다시 시작 태그 시작, 식별자(이름), 시작 태그 끝으로 나누어 마크업되고, 내용은 여러 형태의 내용 모델(model)에 의한 계층 구조로 이루어진다. 그리고 속성의 값이나 페이지 기술 명령등도 SGML 문서에 입력된다. 이러한 SGML 문서를 해당 DTD의 선언과 비교하여 구조상 오류가 없는지 검사하고, 또, SGML 문법에 맞는지를 조사하는 것을 SGML 문서 파서라 한다.[3]

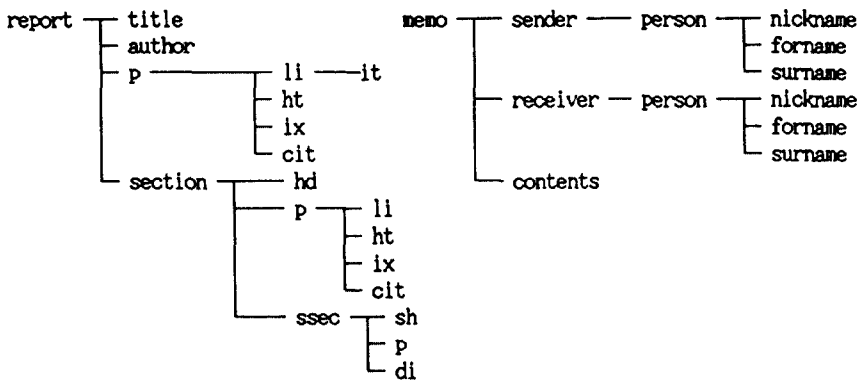
이 논문은 위와 같은 DTD 및 SGML 문서에 대한 파서의 구현 방법에 대하여 기술하였으며, 2장에서는 SGML 문서와 DTD, 3장에서는 SGML 파서에 대하여 설명하였고, 4장에서 구현 및 고찰, 5장에 결론의 순서로 구성하였다.

Ⅱ. SGML 문서와 DTD

한 문서를 SGML에 적용하기 위해서는 우선 엘리먼트를 기본으로 하는 문서의 계층구조를 결정하여야 한다. 예로서 그림 1.에 문서 구조 "report"와 "memo"에 대한 계층 구조를 보인다.

그림 1의 (a)와 같은 report라는 이름의 문서 구조로 볼때, report는 title, author, p, section으로 구성되고, section에는 hd, p, ssec와 같은 구성요소로 이루어져 있다. 이를 SGML 문서화 하기 위해서는 이들 각 구성요소가 DTD의 엘리먼트로 선언되어야 한다.

다음에는 위와 같은 문서 구조를 SGML 구문 규칙에 의하여 DTD를 작성한다. SGML 구문에 의하면 DTD는 mdo(markup declaration open) 기호로 시작하여 mdc(markup declaration close) 기호로 끝나고, mdo 다음에는 DOCTYPE라는 문자열 다음에



(a)report 문서구조 (b)memo 문서구조

그림 1. 문서 구조 예
Fig. 1. Example of Document Structure

문서형의 이름이 오며, 이어서 엔터티 선언, 엘리먼트 선언, 속성 선언등이 마크업 된다. 각종 선언은 그 고유의 문법이 있는데, 이 논문에서는 ISO 8879의 부록 B(표준은 아님)에서 사용한 마크업 기호를 사용하였다. 몇 가지 대표적인 마크업 기호를 표 1. 에 보인다.

표 1. 마크업 기호
Table. 1. Markup Symbol

마크업 이름	기 호	의 미
mdo	<!	마크업 선언 시작
mdc	>	마크업 선언 끝
ero	&	엔티티 참조 시작
refc	;	엔티티 참조 끝
pio	<?	처리 명령 시작
pic	>	처리 명령 끝
grpo	(	그룹(group) 시작
grpc	)	그룹 끝
stago	<	시작 태그 시작
stagc	>	시작 태그 끝
opt	?	선택적임
plus	+	1번 이상
rep	*	0번 이상

엘리먼트 선언은〈!ELEMENT gi-o(cont1, cont2 ?)〉와 같은 형태를 같는데, gi는 선언중인 엘리멘트의 이름이며, cont1, cont2는 다름 엘리멘트의 이름이다. gi 다음의 -0는 앞에 있는 -는 엘리멘트가 SGML 문서에서 마크업될 때 시작 태크가 생략될 수 없음을 나타낸다. 만약 0이면 시작 태크가 생략될 수 있음을 의미한다. 뒤의 0은 끝 태크에 대한 것으로 현재는 생략 가능함을 표시하고 있다. cont1과 cont2 사이의 ","은 연결자로서 기호명은 seq이고, 전후의 엘리멘트가 순서대로 발생해야 함을 의미한다. 다른 연결자로 and(&)와 or(:)가 있는데 &는 순서에는 관계없으나 반드시 동시에 있어야 함을 의미하며, : 앞, 뒤 엘리멘트중 하나는 없을 수도 있음을 말한다. 엔터티 선언은〈!ENTITY % ent-nm "ent-string"〉의 형태이며, ent-nm은 이름이며 ent-string은 내용이다. 이 엔터티를 참조하는 방법은 "&ent-nm;"하면 되는데 그러면 ent-string 이라는 내용으로 대치된다. 엔터티 참조는 DTD의 내용 모델에서 할 수 있으며, 그리고 SGML 문서내에서도 할 수 있다. 속성 선언은 〈!ATTLIST gi attr-name attr-value "default-value"〉의 형태로 선언되며, gi는 선언중인 속성이 관계되는 엘리멘트의 이름이고, attr-name, attr-value는 각각 속성의 이름과 값이다. 그리고 default-value는 속성의 값이 지정되지 않을때에 적용될 값이다. 기호 "〈!"과 ">"은 세가지 선언에서 공통으로 사용되었는데 마크업 기호로서 "〈!"는 mdo로 선언의 시작임을 의미하고, ">"는 mdc로서 선언이 끝났음을 말한다. 그리고 대문자로 표기된 ELEMENT, ENTITY, ATTLIST는 SGML의 예약어로서 변경될 수 없다. 이와 같이 작성된 DTD는 어느 문서들의 공통형태를 갖게 된다. 예는 그림 2.와 같다.

```
<!DOCTYPE memo  [
<! ELEMENT memo          - O (sender, receiver, contents)>
<! ELEMENT sender        O O (person)+ >
<! ELEMENT receiver      O O (person)+>
<! ELEMENT person        O O (nickname | (forename?, surname))>
<!  ELEMENT (forename,
          nickname,
          surname)       - - (#PCDATA)>
<! ELEMENT contents      O O (#PCDATA)>
<!ATTLIST sender          location  CDATA #REQUIRED
                          age       NUMBER        #IMPLIED
                          term       NUMBER        2
>
<!ATTLIST receiver        address   CDATA #REQUIRED
                          age       NUMBER        "PLIED"
>
]>
```

(a)memo의 DTD

```
<!DOCTYPE report [
<! ELEMENT report          - - (title, author, p*, section)>
<! ELEMENT title           - O (#PCDATA)>
<! ELEMENT author          - O (#PCDATA)>
<! ELEMENT p               - O (li*, ht?, ix?, cit?)>
<! ELEMENT ht              - O (#PCDATA)>
<! ELEMENT li              - O (it+)>
<! ELEMENT (it, ix, cit)   - O (#PCDATA)>
<! ELEMENT section         - O (hd, p*, ssec)>
<! ELEMENT hd              - O (#PCDATA)>
<! ELEMENT ssec            - O (sh, p*, li)>
<! ELEMENT sh              - O (#PCDATA)>
<! ATTLIST li    number  CDATA   #IMPLIED>
]>
```

(b)report의 DTD

그림 2. DTD의 예

Fig. 2. Example of DTD

DTD는 문서의 형을 정하기 위한 것으로 실제의 텍스트가 포함된 문서의 내용은 아니다. 실제 문서는 SGML문서로서 마키업되는데 반드시 DTD에 정의된 문서 형태의 논리적 구조에 맞도록 문서 요소에 대한 태그등이 마크업되면서 문서가 작성되어야 한다. 그리고 SGML에는 DTD에 대한 구문 규칙과 함께 SGML문서에 대한 문법도 정의되어 있다. 그것에 의하면 "<!DOCTYPE 문서형명 .."으로 시작하여 각 엘리먼트에 대한 내용, 속성등이 마크업된다.

Ⅲ. SGML 파서

SGML 파서는 파싱 대상에 따라서, DTD를 파싱하는 DTD 파서와 SGML 문서를 파싱하는 SGML 문서 파서로 구분할 수 있다. 파싱은 크게 어휘 분석과 구문 분석으로 나누어지는데, DTD와 SGML 문서는 공통된 마크업 기호를 사용하므로 동시에 DTD 및 SGML 문서에 대한 어휘 분석을 하도록 하였다. 여기서는 UNIX가 제공하는 lex와 yacc를 이용하여 SGML 파서를 설계하였으며, DTD를 입력받아서 정의하고 있는 문서의 논리적 구조를 분석하여 내부 구조화하고, 엔터티 테이블, 엘리먼트 테이블, 속성 테이블을 형성하고, 엔터티 확장을 하도록 하였다. 그리고, 더 기본적인 기능으로서 DTD가 SGML 구문에 맞는지 확인하고 오류가 있을때는 오류 메시지(message)를 출력하도록 하였다. 이어서 SGML 문서를 처리하는데 SGML 구문 규칙에 맞는지, DTD의 구조가 옳게 적용되었는지 등을 조사하고, 각 엘리먼트의 내용이 옳은지 확인하여 내부구조에 입력하며, 속성 값이 지정되어 있을 때는 그것도 받아서

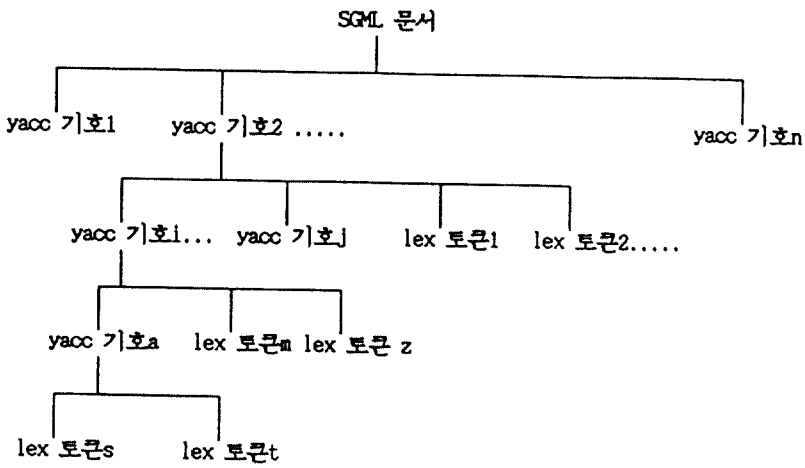


그림 3. yacc.y의 구조
Fig. 3. Structure of yacc.y

내부구조에 집가토록 하였다.

1. 어휘 분석

SGML에서 사용하는 어휘로는 마크업 기호, DOCTYPE, ELEMENT등의 마크업을 위한 예약어, 텍스트(text)에 이용될 일반 문자등이 있다. 이들 어휘 및 어휘의 형태를 lex 문법에 맞추어 lex 정의 화일인 lex.l로 만들었다. lex는 lex.l의 매 어휘마다 yytext라는 문자열을 발생하고, 동시에 yytext에 대응하는 lex 토큰(token)을 생성하여 yacc에 제공한다. lex 토큰은 정수형이지만 필요에 따라 문자열 값까지도 yacc로 전달할 수 있다.

2. 구문 분석

DTD와 SGML문서에 대한 총체적인 SGML 문법이 yacc.y라는 yacc 정의화일로 표현하였다. yacc.y는 yacc 기호들이 순서대로 나열되며, 상호 포함관계를 내포하도록 작성하였다. 즉 한 yacc 기호는 다른 yacc 기호나 lex 토큰으로 구성되는데, 이는 한 yacc 기호가 어떤 yacc 기호들을 갖는지, 또는 어떤 lex 토큰을 갖는지, 어떤 순서로 와야 하는지를 설명한다. 이 관계를 그림으로 표현하면 그림 3과 같다.

그림 3을 보면 SGML 문서는 yacc 기호들의 집합이고, yacc 기호들은 yacc 기호와 lex 토큰으로 구성됨을 알 수 있다. 여기서 유의할 점은 최하위 레벨은 반드시 하나 또는 둘이상의 lex 토큰만이 와야 한다. 그리고 이 yacc.y의 yacc 기호들은 DTD 및 SGML 문서의 가능한 모든 구조를 총괄적으로 내포하고 있으며, SGML 문서의 엘리먼트와 1 : 1의 관계를 갖는 것은 아니다.

여기에서 구성한 yacc.y는 가능한 모든 형태를 포함하고 있으므로 어떠한 형태의 문서를 갖는 DTD나 SGML 문서를 입력하여도 파싱이 가능하다.

2.1 DTD 파싱

우선 DTD를 파싱하면서 문서의 구조를 파악하여 내부 구조화 하며, 구문이 옳은지 검사하여 잘못을 발견하였을 때는 오류 메시지를 출력하는데, 이러한 기능은 yacc내의 동작부분에 의해서 처리하였다. 동작부분은 C언어 프로그램으로 되며 중괄호로 블럭화하여 yacc기호들 사이의 처리를 원하는 임의의 위치에 삽입할 수 있다. 문서의 내부 구조는 상호 연결되어 하나의 문서형을 갖추게 되는데 한 예로 그림 4에 엘리멘트에 대한 내부 구조의 연결 관계를 보인다. 동작부분은 yacc 정의화일에 존재하며, 입력 화일을 검색중에 어떤 yacc 기호가 검색되었을때 처리하고자 하는 명령들을 갖는다.

그림 4.의 E-TYPE는 엘리멘트에 대한 이름이 기억될 곳인데, GEN-ID나 NAMEGROUP을 갖을수 있다. 전자는 엘리먼트 선언시 하나의 이름을 갖는 경우에 적용하였으며, 후자는 복수개의 이름이 같은 형태로 선언될 때 적용하였다. 그리고, 하나의 엘리

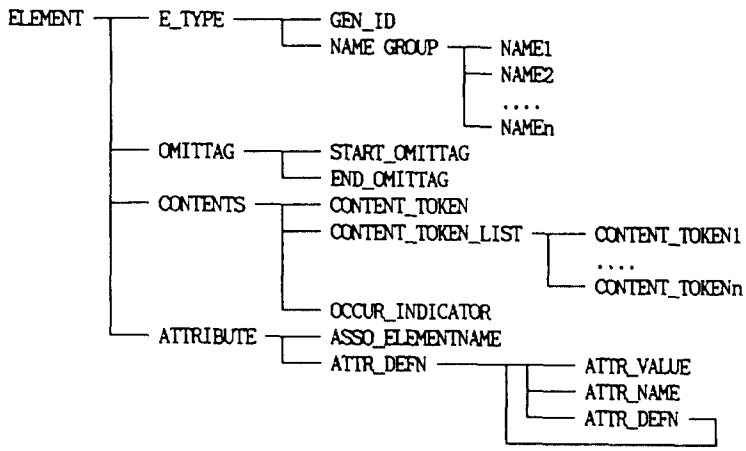


그림 4. 엘리멘트 구조
Fig. 4. Element structure

먼트에 E-TYPE이외에도 OMITTAG, CONTENTS, ATTRIBUTE를 연결시켰다. 그 중에서 ATTRIBUTE는 속성에 대한 선언을 읽을때 내부 구조 내용이 결정되며, CONTENT-TOKEN에 실제 문자 데이터가 들어가는 것은 DTD에 대한 파싱이 끝난 후 SGML 문서를 파싱할 때이다. SGML 파서는 DTD를 파싱하여 위와 같은 내부 데이터 구조를 만들며, SGML 문서의 파싱을 쉽게 하기 위하여 별도로 엘리먼트 리스트(list), 엔터티 리스트, 속성 리스트들을 작성하게 하였다.

엘리먼트에 대한 내부 구조는 그림 4와 같은데, C 언어를 이용하였으며 프로그램내에서는 ELEMENT라는 포인터 타입의 구조체로 선언하였다. 이것의 멤버들은 그림에서와 같이 E-TYPE, OMITTAG, CONTENTS, ATTRIBUTE들인데, 이 멤버들도 구조체 포인터이고, 이들 각각의 구조체에 대한 멤버도 역시 하위레벨에 보여지는 이름을 갖는 구조체들이고, 최하위 레벨의 멤버는 문자열, 정수등이다. 그리고 DTD에서 정의된 모든 엘리먼트들은 하나의 배열로 선언하였으며, 그 요소는 ELEMENT라는 구조체 포인터로 하였다.

그리고, 그림 5에서 보면 CONTENTS 구조중에는 순환구조를 갖는 CONTENTTOKEN-LIST와 같은 것들이 있는데, 이 경우에는 C언어의 구조체에서는 그 멤버중의 하나를 자신과 같은 타입의 구조체 포인터로 지정하였다.

그림 6에 내용구조중의 한 부분을 yacc 정의화일로 작성한 것을 보인다.

그림 6에 나타낸 화일중 최하위 yacc 구분자로 LCLetter, RNI, PCDATA가 있으며, LCLetter 다음의 {lcletter-P($1);}은 동작부로서 $1은 정의부분중 처음 yacc 구분자를 지칭하므로 여기서는 LCLetter를 지칭하며, lcletter-P()라는 함수를 호출하는 매개변수로 이용되었으며, 이 함수는 별도의 화일내에 존재하며 후에 링크된다.

lcletter-P($1)은 입력 화일에서 lcletter에 대응하는 토큰값을 위에서 받은 내부구조체에 해당되는 요소에 저장하는 일을 한다. 그리고, 그림 6을 아래쪽부터 계속 설명하면, ElementToken은 LCLetter와 OccurIndicator에 의하여 정의되고, 이 ElementToken은 상위의 PrimConToken의 한 성분이 되는데, 그때 동작부에는 elementtoken-P()라는 함수가 있는데 이것은 LCLetter와 OccurIndicator로 구성되는 ElementToken 구조체에 하위에서 작성된 Lcletter 구조체와 OccurIndicator를 대입한다. 계속하여 한 단계위로 올라가 보면, PrimConToken은 ContentToken을 구성하는데 그때의 동작부에는 Primcontoken-P() ; Rni-Pcdata=0 ; 의 두 문장이 있는데, Primcontoke-P()는 아래에서 작성된 ElementToken 구조체를 Primcontoken의 멤버로 지정하는 등의 기능을 갖으며 Rni-Pcdata는 외부 변수로서 프로그램의 제어에 이용하기 위한 것이다.

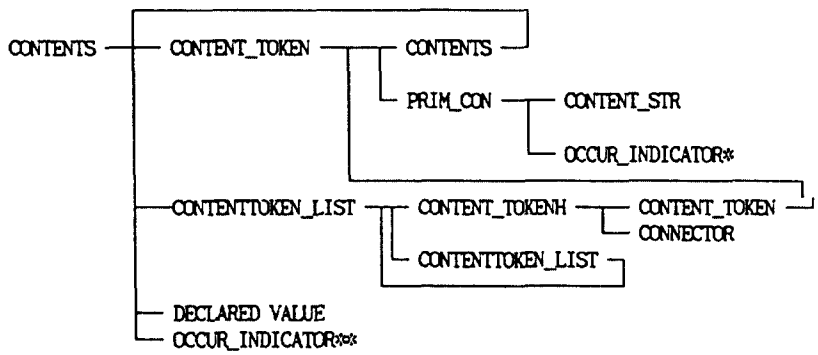


그림 5. 내용 구조
Fig. 5. Content structure

```
ContentTokenList :   ContentTokenH
                     {
                      contenttokenh_P();
                      }
                     | ContcntTokenList
                     {
                      contenttokenlist_P();
                       }
                       ContentTokenH
                     {
                      contenttokenh_P();
                       }
                     |empty
                     ;

ContentTokenH    :
                     Ts
                     {
                      ts_P();
                       }
                       Connector
                     Ts
                        ContentToken
                     {
                      contenttoken_P();
                       }
                     :

ContentToken     :    PrimConToken
                     {
                      primcontoken_P();
                      Rni_Pcdata = 0;
                       }
                     | ModelGroup
                     {
                      modelgroup_P();
                      contgrfg = 1;
                       }
                     |empty
                     ;

PrimConToken     :    RNI
                      PCDATA
                     {
                      Rni_Pcdata = 1;
                     }
                     | ElementToken
                     {
                      elementtoken_P();
                       }
                     | DataTagGroup
                     ;

ElementToken     :   LCLetter
                     {
                      lcletter_P(S1);
                      }
                      OccurIndicator
                     ;
```

그림 6. yacc정의 화일의 부분(내용 구조 부분)
Fig. 6. A part of yacc definition file(content part)

그리고 ContentToken 아래에는 PrimConToken 만이 올수 있는 것이 아니고, ModelGroup도 올수 있는데, 이것은 내용중에서 여기에 보이는 다른 것들보다 훨씬 상위에 있는 yacc 구분자로서, 순환적인 부분이다. ContentToken은 ContentTokenH를 거쳐 ContentTokenList에 포함되게 되고, ContentTokenList 아래에는 ContentTokenH가 오거나 아니면 ContentTokenList와 ContentTokenH가 차례로 올수도 있는데, 후자의 경우는 순환인 경우에 대한 yacc정의 방법이다.

yacc 정의 화일은 약 23K바이트 정도의 크기이며, DTD에 대한 ISO 8879의 정보를 그림 6과 같은 형태로 문서 요소외에 내용, 속성, 엔터티등을 정의하였는데, LINK, SHORTREF등은 아직 처리를 못하였다. 그림 6은 내용부분 중에서도 한 부분일뿐이다.

2.2 SGML 문서 파싱

SGML 파서는 DTD에 이어 SGML 문서를 받아서 SGML 문서가 ISO 8879의 문법에 맞는지, 그리고 앞의 DTD의 문서구조와 일치하는지를 확인하도록 하였으며 세부 사항을 기술하면 아래와 같다.

1) 문서형 명이 DTD의 이름과 같은지 검사한다.

2) 시작 태그의 엘리먼트 이름이 엘리먼트 리스트에 존재하는지 확인하고, 스택(stack)에 입력하여 스택 포인터(pointer)에 의해 현재 엘리먼트의 문서구조상 레벨(level)을 알 수 있게 한다.

3) 상위 레벨의 엘리먼트를 찾아서, 그것의 내용 모델에 현재 엘리먼트가 포함되는지를 확인한다. 그리고, 한 엘리멘트의 내용으로서의 엘리멘트 발생 빈도를 계수하여 DTD상의 정의에 일치하는지 검사한다. 또 내용으로서 이용되는 엘리멘트 사이의 연결자(,, |, &)에 맞도록 그 순서가 맞는지, 있어야 하는 엘리멘트가 없는지등의 오류를 검사한다.

이러한 처리를 하기 위해서는 DTD를 파싱하며 만들었던 스택 포인터를 이용하고, 그림 4의 엘리먼트 구조중 CONTENTS를 조사하여, 발생 빈도를 검사하기 위해서는 "OCCUR-INDICATOR"와 비교하며, 발생순서등을 확인하기 위해서는 CONTENTTOKEN-LIST하위 레벨인 CONTENT-TOKEN아래 레벨에 위치하는 CONNECTOR와 비교한다. 이를 위하여 그림 5에 CONTENTS구조를 좀더 상세하게 표현하였다. 이 그림에서 OCCUR-INDICATOR가 두곳에 있는데 *표는 하위레벨의 내용 요소인 PRIM CON에 대한 것이고, **표는 상위 레벨의 것으로서 실제로 그 아래 레벨의 모든 내용 요소에 공통으로 적용되는 것이다.

SGML의 구문 에러를 검색하기 위해서는 현재 파싱중인 엘리먼트의 엘리먼트 타입(이름)을 DTD 파싱등 작성된 문서요소에 대한 구조체 포인터를 요소로 갖는 배열에서 각 요소의 엘리먼트 타입과 비교하여 같은 이름을 찾은 경우에는 엘리먼트 스택에 입력한다. 그리고 찾지 못한 경우에는 잘못된 엘리먼트이므로 에러 메시지를 내보내고 파싱을 끝낸다. 다음에 스택에 의하여 상위(parent) 엘리먼트를 가져와서 그 구조체에 대한 내용 모델에서 현재의 엘리먼트를 가져와서 그 구조체에 대한 내용 모델에서 현재의 엘리먼트명이 content-token으로서 존재하는지 조사하고, 존재한다면, 연결자(connector)와 발생지시자

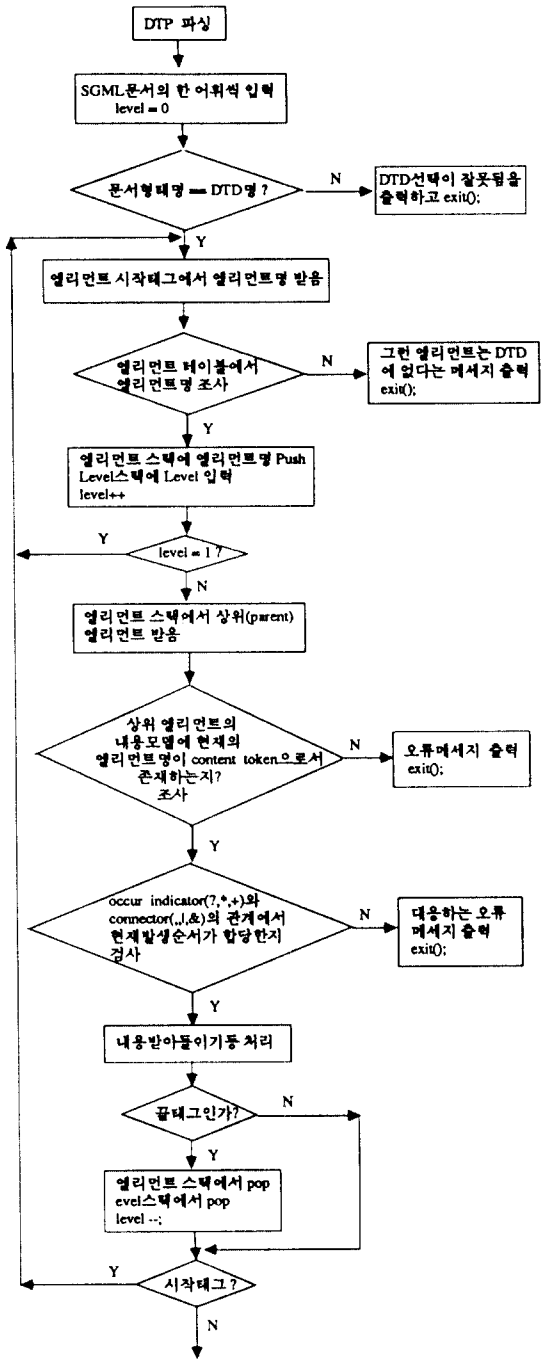


그림 7. 문서의 논리적 구조에 대한 오류 검사 흐름도
Fig. 7. Flowchart of Error Check

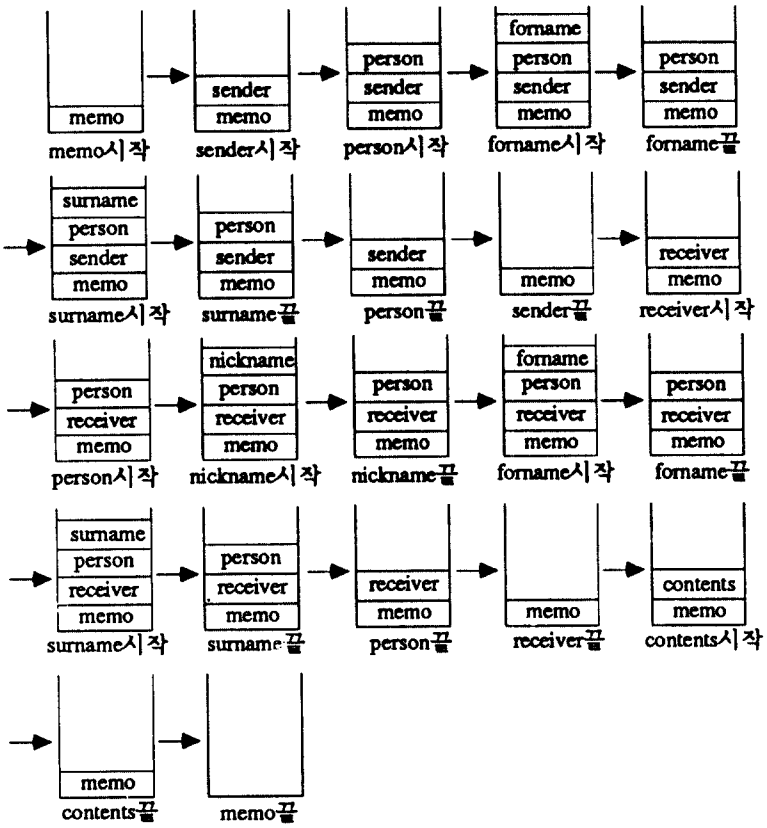


그림 8. 엘리먼트 스택의 변화과정("memo"문서 파싱중)
Fig. 8. Procedure of Element Stack

(?, |, &)의 관계와 비교검토하여 DTD에서 현재의 위치에 필요로 하는 엘리먼트와 맞는지도 검사한다. 지금 파싱중인 엘리먼트가 논리적 위치가 옳다고 검사되면, 다음에 내용에 대한 처리등을 하고, 다시 엘리먼트에 대한 시작태그가 올 경우는 위와 같은 과정을 반복하는데 이 때는 상위 엘리먼트가 바로 앞의 엘리먼트가 된다. 그리고 끝 태그일 경우에는 엘리먼트 스택에서 가장 나중에 입력한 엘리먼트를 출력(pop)한다. 따라서 엘리먼트 스택에는 처리중인 엘리먼트의 상위 레벨이 차례로 있게된다. 이러한 과정에 대한 개략적인 흐름도를 그림 7에 보인다. 그리고 그림 2의 (a)에 보인 DTD를 문서 형태로 한 SGML 문서를 그림 10에 보이는데, 이 문서를 위와 같은 과정으로 진행할 때 엘리먼트 스택의 변화과정을 그림 8에 보인다.

4)속성 값은 엘리먼트의 시작 태그중에 속성에 대한 값이 "속성 이름=값"의 형식으로 지정하며 이러한 경우를 만나면 이를 받아들여 그림 4의 ATTR-VALUE 요소에 입력하는데, 현재 파싱중인 엘리먼트 이름과 ASSO-ELENAME이 같은지도 확인한다.

5)엘리먼트의 최하위 레벨인 잎 노드는 실제의 텍스트 데이터등이 되는데 이들 문자열을 CONTENT-TOKEN의 데이터로서 내부구조에 대입한다. 그림 5에서는 PRIM-CON 아래 레벨의 CONTENT-STR에 해당한다.

Ⅳ. 구현 및 고찰

SGML 파서는 그림 9와 같은 구성도로 나타낼수 있다.

그림 9의 구성도의 좌측 상단은 SGML에서 사용되는 마크업 기호와 예약어들을 lex정의 화일로 작성하고, 이것을 기본으로 하여 UNIX상의 lex에 의해

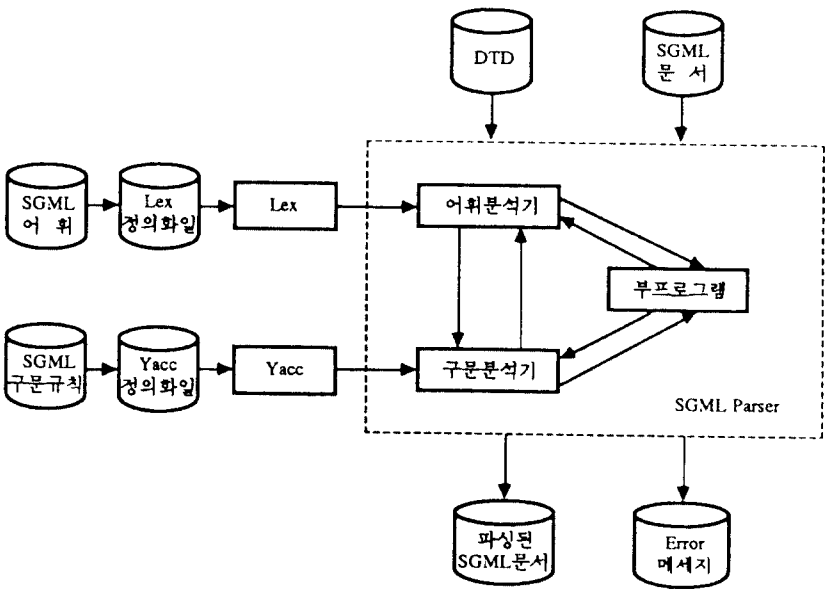


그림 9. SGML 파서 구성도
Fig. 9. Diagram for SGML parser

어휘분석기를 만드는 과정을 표현하였고, 좌측 하단은 ISO 8879에서 정의된 SGML 구문규칙을 yacc정의 화일로 집약하여, yacc에 의해 구문 분석기를 작성하는 것을 보여준다. 중심부의 SGML 파서 내부의 부프로그램은 어휘 분석기나 구문 분석기의 동작부분에 의해서 호출되어 이용된다. 위쪽은 입력되는 DTD와 SGML 문서이며, 아래편은 출력으로서 파싱된 SGML 문서와 오류 출력을 나타낸다. 위 시스템은 UNIX환경에서 설계되었으며, 편지, 보고서, 메모(memo)등의 문서에 대한 DTD와 SGML 문서를 입력하여 검사하였으며, 오류 검출을 하기 위하여 DTD 정의가 SGML 문법에 틀린것, 그리고 SGML 문서가 DTD 정의의 문서 구조형과 맞지 않는 경우에 대하여도 검사하였다.

그림 10은 그림 1의 구조를 갖는 문서에 대한 DTD 및 SGML 문서이며 이것을 SGML 파서에 입력하였을 때의 출력도 역시 그림 10과 같다. 그렇지만 SGML 파서의 출력은 그림 10을 파싱하면서 생성된 내부 구조에 의한 출력으로서, 일반 워드프로세서로 만들어진 입력 화일과는 다르다.

실제로 몇가지 에러 발생하는 경우 예를들어 설명한다. 그림 10의 문서를 입력하면, 파싱하고 에러 없음이 확인되었다. 그렇지만 그림 10의 DTD중 "memo" 엘리먼트에 대한 내용 모델을 (sender, contents, receiver)로 하면, 즉 receiver와 contents의 두 content token의 순서를 바꾸어 보았을때, "element, contents was needed."라는 에러 메시지가 발생되며, 파싱이 중지된다. 이것은 "memo"라는 엘리먼트에는 "sender", "contents", 그리고 "receiver"가 연결자 ","(seq)에 의하여 연결되어지므로 아래의 SGML 문서에서도 그런 순서로 엘리먼트가 와야 하는데, SGML 문서 쪽은 원래대로 "receiver"가 "contents"보다 앞에 있기 때문에 "contents" 엘리먼트가 올 차례이기 때문이다.

다음에 "memo"에 대한 내용 모델의 연결자를 ","가 아닌 "|"로 고쳐서 실행해 보았더니, 에러없이 파싱이 되었다. 그것은 "|" 연결자는 순서대로 하지 않아도 되기 때문이다.

그리고 그림 10의 "person" 엘리먼트가 둘있는데 "sender" 다음의 것은 그 아래에 "forename"과 "surname"만이 있는데, "receiver" 아래의 "person"에는 "nickname", "forename", "surname"이 있다. 이들은 DTD에서 "|" 연결자로 연결되었기 때문에 에러가 아니며, 실제로도 파싱이 끝까지 되어 원래의 문서와 같은 문서가 출력되었다. "person"의 경우 "forename"이 없을 수도 있는데 그것은 DTD에서

```
<!DOCTYPE report [
<! ELEMENT report          - - (title, author, p*, section)>
<! ELEMENT title           - O (#PCDATA)>
<! ELEMENT author          - O (#PCDATA)>
<! ELEMENT p               - O (li*, ht?, ix?, cit?)>
<! ELEMENT ht              - O (#PCDATA)>
<! ELEMENT li              - O (it+)>
<! ELEMENT (it, ix, cit)   - O (#PCDATA)>
<! ELEMENT section         - O (hd, p*, ssec)>
<! ELEMENT hd              - O (#PCDATA)>
<! ELEMENT ssec            - O (sh, p*, li)>
<! ELEMENT sh              - O (#PCDATA)>
<! ATTLIST li   number  CDATA   #IMPLIED>
]>
<! DOCTYPE report >
<report>
<title> The Advantage of Sgml </title>
<author> Martin Brian </author>
<p> SGML provides standardized technique for marking up
electronically prepared copy that does not presume any
typographic knowledge on the part of users. It can be
applied by almost any author or editor to code virtually
any type of book. </p>
<p> SGML defines <li number = alpha>
<it> the structure of the document</it>
<it> the characters to be used in the document</it>
<it> text entities that are to be used more than once in
the document</it>
<it> externally stored information that is to be incoperated
into the text</it>
<it> special techniques used to mark up text</it>
<it> the way text is to be processed. </it> </li></p>
<section>
<nd>The structure of document </hd>
<p>The <ht> structure </ht> of a document can be thought of
in terms of a series of nested<ix>elements </ix> the start
and end of each element being at some clearly definable point
in the text. </p>
<ssec> <sh> The structure of a Memo </sh>
<p>The memo shown in figure2. 1 of <li> An Author's
Guide to the Standard Generalized Markup Language </li>
show how the structure of a memo can be thought of as
<li> <it> a form  line </it> <it> One or more'to' lines</it>
<it>a Date lines</it></li></p>
<p><li>
<it>An optional subject line </it><it>one or more paragraphs
of a text. </it></li></p>
</ssec>
</section>
</report>
```

(a)report

```
<!DOCTYPE memo  [
<! ELEMENT memo          - O (sender,  contents, receiver)>
<! ELEMENT sender        O O (person)+ >
<! ELEMENT receiver      O O (person)+>
<! ELEMENT person        O O (nickname | (forename?,  surname))>
<!  ELEMENT (forename,
           nickname,
           surname)      - - (#PCDATA)>
<! ELEMENT contents      O O (#PCDATA)>
<!ATTLIST sender         location  CDATA          #REQUIRED
                         age       NUMBER         #IMPLIED
                         term       NUMBER         2
```

```
>
<!ATTLIST receiver      address   CDATA      #REQUIRED
                        age       NUMBER     "PLIED"
>
]>
<!DOCTYPE memo >
<memo>
        <sender location = seoul kangnamgu >
                <person>
                    <?   cat>
                        <nickname> Bear </nickname>
                        <forename>Sylvia</forename>
                        <surname>Warmer</surname>
                </person>
        </sender>
        <receiver age = 22 address = "pusan">
                <person>
                        <forename>Sylvia</forename>
                        <nickname> Bear </nickname>
                        <surname>van Egmond</surname>
                </person>
        </receiver>
        <contents>
                Tomorrow's meeting will be postponed.
        </contents>
</memo>
```

(b)memo

그림 10. SGML 파서를 시험하는 DTD와 SGML문서(계속)
Fig. 10. Test document of SGML parser(cont.)

"person" 엘리먼트의 내용 모델을 정의할 때 "forename" 다음의 발생지시자로 "?"을 썼기 때문에 "forename"은 있을수도 있고 없을 수도 있는 것이며, 실제로 증명되었다.

이 SGML 파서를 이용하기 위해서는 어느 정도 까다롭다고 볼 수 있는 SGML 문법을 파악하여야 하기 때문에 전문가가 아닌 일반 사용자가 SGML 문서를 사용하기는 어렵다. 이러한 점을 해결하기 위한 수단으로 SGML 문서 구조 편집기를 설계하기도 한다. SGML 문서 구조 편집기는 선택된 DTD에 맞추어 SGML 문서를 편집하여 출력하는데 이때 이 SGML 문서가 옳게 작성되었는지 판단하는 도구로서 이 SGML 파서가 이용될 수도 있다. 오류가 있으면 오류메시지를 내보내며, SGML 편집기로 하여금 수정도 가능하게 함으로써, SGML 편집기의 품질이 향상될 것이다.[6][9]

SGML은 실용적인 면에서나 학술적인면 양쪽에 유리하다. 출판계에서는 마크업의 비율을 적감토록 하며, 책을 출판하는데 걸리는 시간도 줄이도록하며, 텍스트 데이타 베이스의 유연성을 크게 한다. 사무용에서는 문서의 종류, 워드프로세서 및 출판상치의 종류에 관계없이 문서 교환을 허용한다. 동시에 SGML의 마크업 규칙이 엄격하기 때문에 다양한 목적을 갖는 문서의 구조를 생성할 수 있다. 이러한 이점 때문에 Robertson의 SGML에 기초한 일반적인 마크업 시스템인 "SGML Markup for Publishing at Leeds", Crabtree의 임의의 일반 마크업을 특정코드로 변환해 주는 "Miles 33 and SGML"등의 연구가 진행되었고, 옥스포드 영어 사전(Oxford English Dictionary : OED)의 전자 버전 만드는데도 SGML을 사용하였다. 그리고 휴렛패카드사의 기술 관계 문서를 전세계에 걸쳐 분포되어 있는 50여개의 지사에 전달하는데도 SGML을 이용하였는데 이를 위하여 "MARKUP"이라는 SGML 파서가 이용되었다.[8][13][14]

V. 결 론

이와 같이 SGML이 유럽지역에 이어 미국에서도 이용이 늘어나고 있는 즈음, 국내에서도 SGML에 대한 연구가 필요하다고 본다. 여기서는 ISO 8879를 기본으로하여 SGML에 관한 전반적인 문법을 yacc정

의 화일화 하여 SGML에 의한 DTD 및 SGML문서의 구문 규칙이 옳은지 판단하는 것을 기초로 하여 처리중인 SGML 문서와 엘리먼트, 속성, 내용들에 대한 내부 구조를 만들었다. 이것들은 이후에 연계되는 페이지 기술등의 다른 처리에도 응용될수 있을 것으로 생각된다. 끝으로 DTD 및 SGML이 바르다고 분석되었을 때는 바른 SGML 문서를 출력하고, 잘못된 점이 발견되면 이에 대한 오류 메시지를 출력하며 실행이 중단된다.

유럽에서는 문서의 재이용, 편집의 합리화에 중점을 두고 SGML을 개발하여 공문서의 발행, 매뉴얼의 제작등을 중심으로 사용되고 있으며, 미국에서도 SGML에 대한 연구가 활발하여 IEEE, 미국 화학회 등의 학회와 미국 출판연합회에 의하여 SGML 응용 레벨에 대한 ANSI 규격, Z 39.59를 정하기도 하였다. 그리고 일본에서도 1989년에 SGML 간담회가 발족되어 SGML의 보급이 시작되었다. 그러나 국내에서는 이렇다할 연구 결과가 없는 것으로 알며, 이 논문은 국내의 SGML 연구의 출발이라고 여겨진다. 앞으로 국가적인 차원에서 본격적인 연구가 필요하다고 생각된다.[10][11][12]

이러한 파서를 설계하면서 어려웠던 점은 ISO 8879를 정확하게 이해하는 것이었다. 그리고 SGML 중의 사항들은 대부분 선택적인 항을 포함하고 있어서 파싱할때 모순성이 일어나는 점도 곤란했다. 그러나 SGML 표준의 부록에 있는 속성, 엘리먼트, 그리고 엔터티를 선언하는 예등이 크게 도움이 되었다.

이 논문에서 설계한 SGML 파서는 기본적인 기능을 갖추었을 뿐으로 ISO 8879에서 정의하고 있는 LINK, OMITTAG등의 기능에 대한 처리가 이루어지지 못하였다. 앞으로 더욱 개선하여 그러한 기능들을 추가시키고, 한글로 작성된 SGML 문서도 파싱할 수 있도록 연구하여야 겠다. 한편 SGML 문서를 파싱하며 만들어진 내부 데이터 구조를 이용하여 실제 문서를 작성하는 분야의 연구도 병행하여야 한다고 생각된다.

참 고 문 헌

1. ISO 8879 : Information Processing-Text and Office Systems-Standard Generalized Markup Language(SGML), 1988.
2. Martin Bryan, "An Author's guide to the Standard Generalized Markup Language," 1988.
3. ISO TR-xxx, Information Processing Systems Text and Office Systems Operational Model for Text Description and Processing Languages, 1987.
4. David Barron, "Why use SGML," Electronic Publishing, Vol.2(1), pp.3-24, 1989.
5. "Introducing the Standard Generalized Markup Language(SGML)," Omnicon open system's, Data Transfer, June 1989.
6. Donand D. Chamberlin and Charles F. Goldfarb, "Graphic Applications of the Standard Generalized Markup Language(SGML)," computer & Graphics Vol.11, No.4, pp.343-358, 1987.
7. Jos Warmer and Sylvia Van Egmond, "The implementation of the Amsterdam SGML," Electronic Publishing, Vol.2(2), pp.65-90, 1989.
8. Lynne A. Price, MARKUP : Hewlett-Packard's SGML Implementation ; SGML User's Group Bulletin, 2(2), pp.116-118, 1987.
9. Kiyoshi Toyoda, Erikumagai and Tatsuo Bando, "SGML Document Structure Editor," National Technical Report Vol.36, No.5, Oct. 1990.
10. ISO/TR 9544 Computer-Assisted Publishing-Vocabulary(Geneva : ISO), Jul. 1988.
11. ISO/TR 9573 Techniques for using SGML (Geneva : ISO), Dec. 1988.
12. Joan M. Smith and Robert stutely, "SGML : The User's Guide to ISO 8879," Ellis Horwood Ltd., Chichester, 1989.
13. E. Weiner, "The electronic English dictionary." Offord Magazine 6-9, 1987.
14. M.Crabtree, "Miles 33 and SGML," SGML User's Group Bulletin, 3(1), pp.22-24, 1988.

洪 順 善(On Sun Hong) 正會員

1982년 2월 : 광운대학교 전자공학과(공학석사)

1988년 2월 : 광운대학교 전자계산기공학과 박사과정 수료

1984년 3월～1990년 2월 : 연암공전 전산과(조교수)

1990년 9월～현재 : 광운대학교 전산원 조교수

鄭 會 京(Hoe Kyung Jung)정회원

1961년 1월 28일생

1985년 2월 : 광운대학교 전자계산기공학과 졸업(공학사)

1987년 2월 : 광운대학교 전자계산기공학과 졸업(공학석사)

1989년～현재 : 광운대학교 전자계산기공학과 박사과정 재학중

1990년 1월～1990년 7월 : 영국 ICL 연수

※주요관심분야 : ODA, SGML, Hypermedia, GUI

李 壽 淵(Soo Youn Lee) 正會員

1969년 : 광운대학교 전자통신공학과(공학사)

1977년 : 연세대학교 전자공학과(공학석사)

1983년 : 일본 교토대학교 정보공학과(공학박사)

1973년 3월～현재 : 광운대학교 전자계산기공학과 교수