객체지향 정보 저장소에서 참조 무결성 보장을 위한 XLink 시맨틱스

(XLink Semantics in Object Repository for Guaranteeing Referential Integrity)

박 희 경 † 박 상 원 †† 김 형 주 †††
(Heekyung Park) (Sangwon Park) (Hyoung-Joo Kim)

요 약 XLink는 XML에서 문서들 사이 또는 엘리먼트(element) 사이의 링크를 정의하는 언어로 HTML에서의 하이퍼 링크(hyperlink)와 유사한 역할을 하는데 하이퍼 링크보다 훨씬 정교하고 다양한 방식의 링크를 표현할 수 있다. XLink로 연결된 요소 사이에는 관계성(relationship)이 발생하는데 이런 관계성을 XML 저장소에서지원해 주지 않을 경우 관계성의 표현을 각 응용 프로그램에서 직접 프로그래밍 해야 하기 때문에 프로그램의 개발 및 유지, 보수 단계에서 관계성 관리와 관련된 많은 오버헤드가 발생한다. 본 논문에서는 XML 문서를 저장하는 객체지향 정보 저장소(object repository)에서 XLink를 지원해 주기 위한 시스템을 제안한다. 이 시스템은 XML 문서 저장을 위한 객체 정보 저장소 시스템인 XDOM(Persistent DOM 형태의 XML Repository)상에 구현하였다. XLink를 지원해주기 위해 본 시스템에서 제공해주는 기능은 참조 관계성, 확장 링크(extended link)와 XLink를 위한 DOM 트리 확장을 위한 마운트(mount) 기능이다.

키워드 : XLink, XML, 객체지향정보저장소

Abstract XLink is XML Linking Language that defines links between XML documents or XML elements. It uses XML syntax to create structures that can describe links similar to the simple unidirectional hyperlinks of HTML, as well as more sophisticated links. There exist several relationships between resources that linked by XLink. Without supports from XML Repository for these relationship management, there is a huge overhead related to the management of relationships during both application development and maintenance, since the relationships need to be hard-coded directly into the application program itself.

In this paper, we propose an XLink supporting system in object repository. We describe the design and implementation of this system on top of XDOM(persistent DOMlink XML repository). To support XLink, our system offers referential relationship semantics, extended link and the expansion of DOM tree for XLink.

Key words : XLink, XML, object-oriented repository

1. 서 론

하이퍼텍스트(hypertext)는 문서 중간 중간에 특정 키워드를 두고 문자나 그래픽 화일 등을 유기적으로 결합해 만든 문서다. 문서의 중요한 키워드마다 다른 문서로 연결되는 통로를 만들어 여러 개의 문서가 하나의 문서인 것처럼 보여 주는 방식을 가지는데 이런 통로를 하이퍼 링크라고 부른다. 하이퍼텍스트를 만들기 위해서 사용되고 있는 HTML(HyperText Markup Language)은 쉬운 문법(태그들의 고정 집합)과 에디터 도구들로 인하여 하이퍼텍스트를 만드는 중요한 수단이 되어왔지만, 태그들의 고정 집합이라는 HTML의 특성은 많은 한계를 지니고 있다. 사용자가 태그를 새로 만들 수가 없기에 웹 페이지를 표현하는데 있어서는 강력한 반면 구조적인 데이타를 다루기엔 부족한 점이 많았다. 이런

· 본 연구는 BK21 사업 및 2003년도 한국외국어대학교 교원연구비의 지원을 받았음
† 비 회 원 : 서울대학교 전기 · 컴퓨터공학부
hkpark@oopsla.snu.ac.kr
†† 종신회원 : 한국외국어대학교 컴퓨터및정보통신공학부 교수
swpark@hufs.ac.kr
††† 종신회원 : 서울대학교 전기 · 컴퓨터공학부 교수
hjk@ooplsa.snu.ac.kr
논문접수 : 2001년 12월 27일
심사완료 : 2003년 7월 21일

단점을 해결하기 위해 등장한 것이 XML(eXtensible Markup Language)[1]이다.

XML을 이용하여 사용자는 자신만의 태그를 정의할 수 있고 이런 태그는 문서 포맷 양식을 지정하는 것에서부터 구조적인 데이타를 만드는 것까지 여러 가지로 이용될 수 있다. 인터넷 상의 데이타 교환을 위한 사실상의 표준으로 떠오르면서 XML의 여러 분야에 걸친 표준화 작업이 W3 컨소시엄에서 진행되고 있는데, XLink(XML Linking Language)[2]도 그 중의 하나로 XML 데이타 간의 링크를 정의하는 언어이다. XLink는 하이퍼텍스트에서 하이퍼 링크와 유사한 역할을 하지만 표현할 수 있는 링크의 종류는 훨씬 다양하다. 링크의 정의가 링크에 참여하지 않는 자원(resource)에 있는 제 3자 링크(third-party link)를 이용하여 쓰기 권한이 없는 자원간의 링크도 생성할 수 있고, 다방향 링크(multi-directional link)를 이용하여 사용자의 선택에 따라 다양한 방향으로의 링크를 따라 갈 수 있다. XLink로 표현할 수 있는 링크에 대한 설명은 3절에서 자세히 설명하겠다.

하이퍼텍스트에서 하이퍼 링크를 따라가다 보면 종종 깨진 링크(broken link)를 발견할 수 있다. 깨진 링크는 크게 두 가지가 있는데 하나는 참조되는 대상이 없는 현수 참조(dangling reference)이고 나머지 하나는 링크되는 자원이 다른 내용으로 바뀌어서 원래 링크가 가리키던 내용이 아닌 다른 내용을 참조하게 되는 오문 참조(wrong content reference)[3]이다. 하이퍼텍스트 시스템이 제 기능을 발휘하기 위해서는 이런 깨진 링크가 없어야 하므로 XLink를 이용하여 하이퍼텍스트 시스템을 만들 경우에도 이런 깨진 링크가 없도록 사전에 예방하고, 문제가 생겼을 경우 빨리 발견해서 수정할 수 있도록 하여야 한다. 현재 상용 XML 저장 장치를 이용하여 사이트 내에서 동작하는 XML 응용 프로그램을 작성할 경우 XLink를 정의하고 XLink로 링크되는 자원간에 링크가 깨지지 않도록 참조 무결성(referential integrity)을 보장하도록 하는 것은 '응용 프로그램의 몫'이다. 이런 시스템의 예로 백과사전 사이트를 들 수 있다. 한 사이트에서 돌아가는 백과사전 사이트와 같은 경우 링크되는 자원이 변경되거나 삭제되어서 생기는 문제는 위에서 열거한 현수 참조와 오문 참조 외에도 쓰레기(garbage) 문제가 있다. 링크되는 문서가 링크하는 문서에 종속적인 경우 링크하는 문서가 삭제되면 링크되는 문서는 쓰레기가 된다.

본 논문에서는 하나의 정보 저장소에 저장된 XML 문서들로 이루어진 하이퍼텍스트 시스템을 구축할 경우 참조 무결성을 정보 저장소 단계에서 보장할 수 있는 시스템을 제안한다. 이를 위해서는 XML 저장 장치 단계에서 XLink 시맨틱스를 지원해 주어야 하는데, 저장 장치 단계에서 XLink 시맨틱스를 지원해 줄 경우 응용 프로그래머는 시스템에서 제공해주는 XLink 라이브러리를 이용하여 응용 프로그램을 훨씬 간단하게 작성할 수 있다.

XML 데이타를 효율적으로 저장하고 검색하기 위해 많은 연구가 진행되고 있다. 관계형 데이타베이스나 객체지향 데이타베이스를 이용하여 XML 데이타를 보다 효과적으로 저장하고 검색하기 위한 연구가 활발한데, 특히 객체지향 데이타베이스와 같은 객체지향 정보 저장소는 데이타 모델이 XML의 특성을 자연스럽게 반영할 수 있기 때문에 XML 저장장치로서 각광을 받고 있다. 그리고 객체지향 정보 저장소에 저장된 객체들 간의 관계성을 체계화하여 관계성에 따른 행동 양식을 정의하고 그에 따른 무결성을 보장하기 위한 연구가 진행되어왔다.

본 논문에서는 XML Linking Language[2]에 명시된 XLink 시맨틱스에 XLink로 인하여 XML 자원들 간에 생길 수 있는 참조 관계성 문제를 해결하기 위해서 기존의 관계성 모델을 덧붙인 확장된 XLink 시맨틱스를 XML 저장 장치에서 제공해 주는 시스템을 제안한다. XLink 지원 시스템을 구현할 저장 장치로 객체지향 정보 저장소의 일종인 XDOM[4]을 선택하였다.

본 논문의 구성은 다음과 같다. 2장에서는 XLink에 관한 연구와 링크로 일어날 수 있는 참조 무결성 문제에 대한 연구, 그리고 객체지향 모델링 분야나 ODBMS, RDBMS 분야에서 연구된 관계성의 여러 가지 시맨틱에 대해서 설명한다. 3장에서는 XLink에 대해서 살펴보고, 4장에서는 시스템을 구현할 XDOM에 대해 살펴본다. 5장에서는 본 논문에서 제시하는 XLink 시맨틱스를 객체 정보 저장소 단계에서 제공하기 위한 방안을 제시한다. 6장에서는 시스템 구현과 간단한 예제를 통하여 본 시스템이 제공되었을 경우와 제공되지 않았을 경우 응용 프로그래밍이 어떤 차이를 보이는지 살펴보고, 7장에서 결론을 맺는다.

2. 관련연구

2.1 하이퍼 링크의 참조 무결성

XLink는 하이퍼 링크의 확장이라고 볼 수 있는데 하이퍼 링크에서의 참조 무결성에 관한 연구는 깨진 링크를 찾아내고 예방하는 방법에 대해 이루어져 왔다[3,5]. 깨진 링크는 두 가지로 나눌 수 있다. 첫 번째는 현수 참조 문제로 링크되는 문서가 없는 경우이다. 현수 참조의 경우 로봇 에이전트를 이용해서 현수 참조를 찾아내어 수정하는 방법, 문서를 옮기거나 삭제할 때 기존의 문서 위치에 더미(dummy) 문서를 하나 둬서 옮겨진

문서를 가리키게 하거나 삭제된 사실을 알려주는 방법(forward reference), 문서의 주소로 링크하는 것이 아니라 문서의 고유한 이름으로 링크를 하고 네임 서버를 두는 방법 등이 해결책으로 연구되고 있다[5]. 두 번째는 오문 참조 문제로 링크되는 문서의 내용이 바뀌어서 링크가 처음 의도하던 바와는 다른 내용을 가리키게 되는 경우이다. 오문 참조의 경우는 날짜 스탬프(date stamp)나 버전(versioning) 등의 방법을 이용해서 링크되는 문서의 유효기간을 지정해주는 방법과 링크되는 문서를 문서의 주소로 링크하는 것이 아니라 검색엔진 등에서 그 문서를 찾을 수 있는 질의문으로 문서를 링크하는 방법 등이 있다[6].

2.2 관계성과 참조 무결성

XML의 표준안을 마련하는 W3C에서 링크를 위한 언어로 XLink[2]와 XPointer[7]에 대한 표준을 확정하였다. 데이타를 저장하는 데이타 저장소 레벨에서의 관계성에 대한 연구를 살펴보면 관계형 모델[8]에서는 외래키를 이용하여 개체와 개체 사이의 연결을 간접적으로 나타내고, 객체지향 모델[9]에서는 객체간의 관계성 정의를 사용하여 객체들 사이의 구조적 관계를 데이타베이스 스키마에 직접적으로 표현할 수 있다. 이런 관계성에 관한 연구는 객체들 사이의 관계에 특별한 의미를 부여하여 객체들 사이의 관계를 정의하는 일[10]과 관계성에 따른 객체 사이의 무결성 제약을 보장하는 것으로 나눠볼 수 있다.

객체들 사이의 관계성을 정의하는 것으로 UML (Unified Modeling Language)[11]에서의 관계성을 들 수 있는데, UML에서는 객체들 사이의 관계를 일반화(generalization), 집합(aggregation), 연관(association)의 세 가지로 분류하고 있다.

객체들 사이의 무결성을 유지하기 위한 방안으로 객체들 사이에 관계 무결성(relational integrity)과 참조 무결성, 유일성(uniqueness)을 정의하고, 관련이 있는 객체들 사이에서 일어난 객체의 행동(behavior)에 따른 다른 객체의 행동을 클래스 선언에서 나타낼 수 있도록 한 방법이 있다[12].

두 가지 측면을 결합하여 객체들 사이의 관계성에 체계적인 의미를 부여하고, 이 의미에 따른 객체들의 행동을 규정하는 연구로는 ORION[13]의 복합 객체(complex object)가 널리 알려져 있다. 이 복합 객체는 부분-전체 관계를 나타내는데, 전체 객체와 부분 객체의 관계를 배타성(exclusiveness)과 종속성(dependency)의 두 가지로 표현하였다. SOP에서는 객체지향 데이타베이스에서 ODMG-2.0 모델을 확장하여 관계형 데이타베이스, 객체지향 모델링, 객체지향 데이타 모델링 등의 영역에서 필요로 하는 관계성을 모두 지원하는 관계성 모델을 제안하고 응용 프로그램의 수행시 발생할 수 있는 이상현상에 대한 해결책을 제시하고 있다[14].

3. XLink

XLink로 만들 수 있는 다양한 링크에 대해 살펴보면 그림 1과 같다. HTML에서 '<a>' 태그로 외향 링크

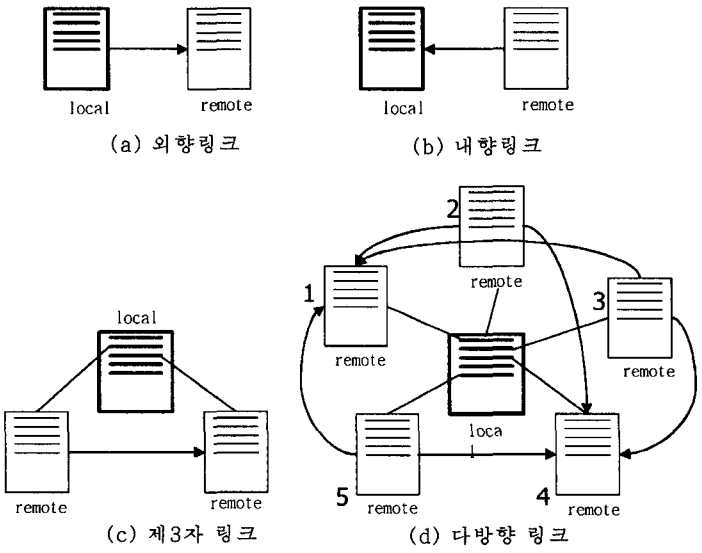

그림 1 XLink로 표현할 수 있는 다양한 링크

(outbound link)만을 지원해 주던 것에 비해 그림 1과 같이 XML에서는 다양한 링크를 지원한다. 그림 1의 각 링크의 의미를 살펴보면 다음과 같다.

- 외향 링크 : 링크가 시작되는 객체를 가리키는 starting과 링크 정의가 같은 문서 안에 존재한다. 지역(local) 문서에서 원격(remote) 문서로의 링크이다.
- 내향 링크(inbound link) : 링크가 끝나는 객체를 가리키는 ending과 링크 정의가 같은 문서 안에 존재한다. 원격 문서에서 지역 문서로의 링크이다.
- 제3자 링크 : 링크의 정의가 링크에 참여하지 않는 제3의 문서에 존재하는 링크이다.
- 다방향 링크 : 링크를 따라가는 사용자의 선택에 따라 링크 목적지가 달라진다. 그림 1에서 2→1, 2→4, 3→1, 3→4, 5→1, 5→4로의 항해가 가능하다.

3.1 XLink 애트리뷰트

XML에서 객체(XML 문서나 XML 엘리먼트) 간의 링크를 정의하는 가장 기본적인 것으로는 그림 2와 같이 XML의 ID와 IDREF 애트리뷰트를 이용하여 링크를 정의하는 방법이다. XLink도 XML 문법을 이용하여 다양한 DTD(Document Type Definition)를 만들어 정의하고, 그에 따르는 XLink가 포함된 XML 문서를 만든다. XLink에 관계된 엘리먼트나 애트리뷰트는 DTD에서 사용자가 임의로 정의할 수 있지만, 본 논문에서는 'http://www.w3.org/1999/xlink' 네임스페이스에 따른 애트리뷰트를 사용한다. XLink 애트리뷰트는 다음과 같은 것들이 있다.

- XLink element type attribute(type)

이 애트리뷰트는 XLink의 종류를 나타내는 애트리뷰트로 simple, extended, locator, arc, resource, title, none 중에서 값을 가질 수 있다.

```
<!DOCTYPE section [
<!ELEMENT section (...)>
<!ATTLIST section target ID #REQUIRED>
...
<!ELEMENT xref (...)>
<!ATTLIST xref link IDREF #REQUIRED>
...
]>
<section target = s6>
<title>This is Section 6</title>
</section>
...
<para>Please refer to
<xref link = s6>Section 6</xref>
for more details</para>
```

그림 2 XML에서의 링크 정의(section.xml)

- Locator Attribute(href)

XLink 응용 프로그램이 원거리 객체를 찾을 수 있게 하는 역할을 하고 simple-type 엘리먼트에서 사용될 수 있고 locator-type 엘리먼트에서는 필수적이다.

- Semantic Attribute(role, arcrole, title)

링크 문맥상 링크의 의미를 나타낸다. role은 extended-type, simple-type, resource-type의 엘리먼트에서 사용되고, arcrole은 arc-type이나 simple-type 엘리먼트에서 사용된다. title은 어떤 타입에서나 사용가능하다. 본 논문에서는 role과 arcrole을 이용하여 XLink에 참조 관계성을 부여한다.

- Behavior Attribute(show, actuate)

simple-type과 arc-type에서 사용할 수 있고 링크의 행동 양식을 정의한다. show는 링크되어지는 객체, 즉 ending을 보여줄 때, 어떻게 보여줄 것인가를 알려주고 new, replace, embed, other, none 중에서 하나의 값을 가진다. actuate)는 링크를 따라갈 것인가를 정의하고 onLoad, onRequest, other, none 값을 가질 수 있다.

- Traversal Attributes(label, from, to)

label은 resource-type이나 locator-type에서 쓰여지고 from, to는 arc-type에서 사용되며 링크를 탐색하기 위해서 사용된다.

3.2 XLink에서 제공하는 링크의 종류

그림 3은 XML 문서와 XLink를 사용해서 만들 수 있는 백과사전 사이트의 한 페이지이고 그림 4는 이를 위한 XML 문서이다. 이 예제에서 3.1절의 애트리뷰트가 실제로 어떻게 사용되는지 알 수 있다.

XLink는 단순 링크(simple link)와 확장 링크의 두 가지 종류의 링크를 제공한다.

- 단순 링크는 로컬 객체에서 원격 객체로의 링크, 즉 그림 1에서와 같이 외향 링크를 정의한다. 그림 4의 'xlinkitem.xml'에서 <term> 엘리먼트가 단순 링크의

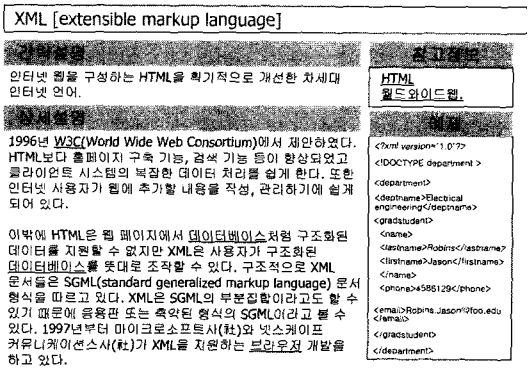

그림 3 백과사전 사이트 예제

```
<item>XML
<title>XML(extensible markup language)</title>
<summary>인터넷 웹을 구성하는 HTML을 획기적으로 개선한 차세대 인터넷 언어.</summary>
<detail>
  <text>1996년</text>
  <term
    xlink:type = "simple"          xlink:href = "termlist.xml#id(w3c)
    xlink:role = "referterm"     xlink:title = "Refer Terminology"
    xlink:show = "new"             xlink:actuate = "onRequest">
   W3C(World Wide Web Consortium)</term>
  <text>에서 제안하였다. ...중략...브라우저 개발을 하고 있다.</text></detail>
<example xlink:type = "extended">
  <resource xlink:type = "resource" xlink:label = "from">
        예제</resource>
  <locator xlink:type = "locator"        xlink:href = "xmlexam.xml"
          xlink:role = "referexam"    xlink:label = "to"/>
  <go xlink:type = "arc"           xlink:from = "from"
      xlink:to = "to"              xlink:role = "showexam"
      xlink:show = "embed"      xlink:actuate = "onLoad" />
</example></item>

<relateditems xlink:type = "extended">
  <relitem
    xlink:type = "locator"    xlink:href = "xmlitem.xml"
    xlink:label = "item"      xlink:role = "referitem"
    xlink:title = "Item" />
  <relitem
    xlink:type = "locator"         xlink:href = "htmlitem.xml"
    xlink:label = "relateditem"    xlink:role = "referitem"
    xlink:title = "HTML" />
  <relitem
    xlink:type = "locator"           xlink:href = "wwwitem.xml"
    xlink:label = "relateditem"      xlink:role = "referitem"
    xlink:title = "WWW" />
<relitemlist
  xlink:type = "arc"
    xlink:from = "item"              xlink:to = "relateditem"
    xlink:arcrole = "relateditemlist"
    xlink:show = "new"               xlink:actuate = "onRequest" />
```

그림 4 그림 3에 대한 XML 문서 예제

한 예이다.

- 확장 링크는 그림 1에서 보는 바와 같이 제3자 링크, 내향 링크, 다방향 링크 등 XLink의 완전한 기능을 제공한다. 따라서 확장 링크는 지역 자원(local resource)를 포함하는 엘리먼트, 원격 자원(remote resource)을 가리키는 엘리먼트, 링크 탐색(arc traverse) 법칙을 나타내는 엘리먼트, 사용하는 사람의 판독을 돕는 엘리먼트 등으로 이루어져 있어 복잡한 구조를 띈다. 그림 4의 'relateditemls.xml'는 'xmlitem.xml'을 starting으로 'htmlitem.xml'와 'wwwitem.xml'을 ending으로 가지는 링크를 나타내고 있다.

4. XDOM : 객체지향 정보 저장소

XML 문서는 트리 형태의 데이타를 포함하고 있기 때문에 XML에서 데이타를 접근할 특별한 방법이 필요하다. 따라서 XML 응용 프로그램에서는 XML 문서를 메모리에 적재해서 필요한 부분을 사용하기 위해 DOM[16]을 이용한다. DOM은 트리 구조의 문서에서 노드를 노드 객체로 만들고 그 객체에 대한 접근 방법을 제공하는데, XML 문서의 엘리먼트, 텍스트 등을 노드로 만들고 이들을 접근하고 조작할 수 있는 API들을 제공한다. DOM을 이용하여 응용 프로그램에서는 보다 편리하게 XML 문서를 다룰 수 있다. DOM은 XML 문서를 파싱한 다음 트리 구조의 노드 객체들을 메모리에

적재하는데, DOM에 대한 특별한 저장 시스템이 없으므로 한번 사용한 객체들을 다음 번에 사용할 수 없고 한번에 메모리에 다 적재하므로 메모리 사용에 문제점이 있다.

이런 문제점을 해결하기 위하여 자바 환경에서 한 번 사용한 DOM 객체를 영구히 사용하기 위해 XDOM[4]이라는 영구적인 DOM 형태의 XML 정보 저장소를 만들었다. 그림 5는 XDOM의 구조를 보여준다. XDOM은 데이타와 로직을 분리하는데 Persistent와 Wrapper 객체의 분리를 통해 이를 수행한다. 이것은 객체지향 정보 저장소에서 논리적인 모델인 Wrapper 객체와 물리적인 모델인 Persistent 객체를 분리할 수 있게 해준다. Persistent 객체는 화일 상에 저장되는 객체로서 객체의 데이타나 다른 객체간의 관계를 이진 형태로 저장하고, Wrapper 객체는 DOM API를 구현한 객체로서 Persistent한 객체로부터 데이타를 가져와서 사용자가 원하는 DOM 객체로 반환한다. Persistent 객체와 Wrapper 객체 사이에 둘 사이의 동기화를 위한 객체 캐쉬(object cache)를 두고, Persistent 객체는 객체 캐쉬에서만 참조되어 JVM(Java Virtual Machine)이 쓰레기 수집(garbage collection)할 수 있게 한다. 객체 캐쉬는 외부 요청이 있을 때, 해당하는 Persistent 객체의 Wrapper 객체를 반환하고 사용자 변수는 Wrapper 객체만을 가리킨다.

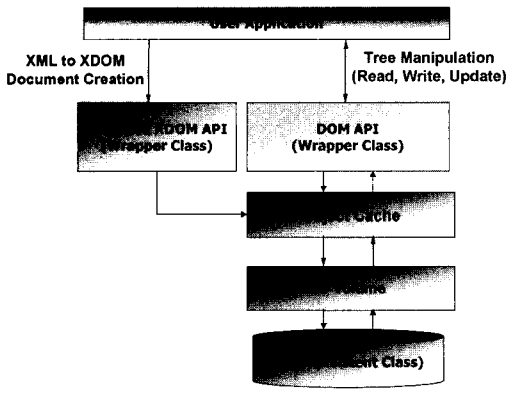

그림 5 XDOM 구조

5. XLink 시맨틱스 지원 시스템 설계

본 논문에서 설계하고 구현한 시스템에서는 사용자가 XDOM에 XML 문서를 저장하고 새로운 문서를 생성할 때 XLink를 쉽게 디자인할 수 있도록 XLink를 지원하는 라이브러리를 제공한다. 예를 들어 백과사전 사이트를 구축한다고 할 때, 본 시스템에서는 백과사전의 각 페이지에 존재할 수 있는 링크를 쉽게 생성할 수 있는 라이브러리를 제공하고, 생성되어 정보 저장소에 저장된 페이지 사이에 존재하는 참조 관계성을 보장해주는 기능을 제공한다.

XDOM에서 XLink를 지원하기 위해서는 다음의 세 가지 기능을 갖추어야 한다. 첫째, 링크에 참여하는 리소스들간의 참조 관계성을 정의할 수 있는 관계성 모델을 제공하고, 관계성 모델에 따라 참조 무결성을 유지시킨다. 링크의 참조 무결성을 위해서는 현수 참조와 오문 참조, 쓰레기 문제를 해결해야 하지만, 본 논문에서 제안하는 시스템에서는 현수 참조와 쓰레기 문제에 대한 무결성에만 초점을 맞추었다. 둘째, XLink가 제공하는 다양한 링크를 디자인할 수 있는 기능을 제공해야 한다. 셋째, XDOM에서 XML 문서를 DOM 형태로 파싱하는 것을 가능하게 해주는데, XML 노드가 XLink로 다른 XML 문서의 노드와 연결되어 있을 경우 연결된 XML 문서도 DOM 트리 확장을 통해 하나의 DOM처럼 보여질 수 있게 하는 마운트 기능이다.

5.1 참조 관계성 모델 정의

이 절에서는 XLink에서 starting과 ending의 관계(이하 starting-ending 관계)를 나타낼 수 있는 관계성 모델을 제안한다. 웹에서 XLink는 다른 도메인에 있는 리소스들 간의 링크도 정의할 수 있지만, 본 논문에서는 한 도메인 상에 있는 리소스들 간의 XLink에만 제한하여 참조 무결성을 지원해주기 위한 관계성 모델을 제시한다.

여러 관계성 관련 논문에서 제안한 부분-전체 관계에 대한 관계성 모델을 참조하여 XLink에 맞는 starting-ending 관계성 모델을 제안할 수 있는데, 본 논문에서는 SOP의 관계성 모델[14]을 바탕으로 XLink에서의 starting-ending 관계성 모델을 제안한다. 본 논문에서는 XLink에서 starting-ending 관계에서 일어날 수 있는 여러 종류의 관계성 시맨틱을 제공하여 응용 프로그램에서 서로 관련이 있는 리소스들이 그 시맨틱에 알맞는 관계성을 선택할 수 있도록 하였다.

관계성 모델을 살펴보기에 앞서 XLink로 링크에 참여하는 XML 리소스들 사이에 어떤 관계성이 생길 수 있는지를 그림 6에서 살펴보자. 그림 6은 그림 3과 그림 4를 도식화한 것이다. 그림 6에서 starting-ending 관계에 있는 XML 객체들은 다음과 같다.

(2,6), (2,(3,4)), (3,7), (4,8), (5,(2,3,4)), (6,(7,8)), (16,17), (16,19), (17,19)

여기서 (2,6), (3,7), (4,8), (17,19)는 locator-type 엘리먼트와 href 애트리뷰트로 가리키는 엘리먼트 사이에 성립되는 관계성이고, (2,(3,4))와 (6,(7,8)), (16,17), (16,19)는 arc-type 엘리먼트에 의한 관계성이다.

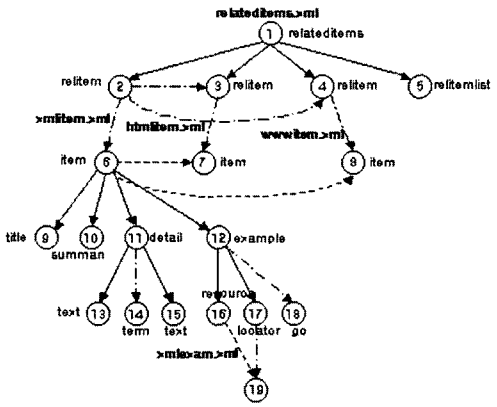

그림 6 XML 객체들 사이의 링크

앞에서 살펴본 XLink로 관계성이 만들어지는 객체 사이의 starting-ending 관계를 다음의 네 가지 요소를 사용하여 모델링하였다.

- 배타성은 Exclusiveness와 Shared, 두 가지 측면이 있다. 배타성은 ending이 단 하나의 starting과 관련이 있다는 것을 나타낸다. 즉 동시에 다른 링크에 속할 수 없다는 것이다. Shared는 동시에 여러 개의 링크에 의해서 starting-ending 관계가 성립할 수 있음을 의미한다.
- 다양성(multiplicity)은 하나의 링크 정의에 대해 몇 개의 링크가 있을 수 있는지를 나타내고 1:1와 n:m 두 가지가 있는데 1:1은 하나의 starting에 대해서 하나의 ending이 존재하는 것이고 n:m은 여러 개의 starting과 ending이 존재하는 것으로 그림 6에서(5, (2,3,4)), (2, (3,4)), (6, (7,8))의 경우이다.
- 종속성은 starting과 ending의 존재여부가 각각 대응하는 ending이나 starting에 달려있는 것을 나타낸다. 종속성은 Deletion, Nullify, Blocking의 세 가지 측면이 있고 이들은 starting과 ending 모두에 적용된다. starting에서 ending에 대해 위의 세 가지 요소를 적용시킬 때의 의미는 다음과 같다. Deletion은 starting이 삭제되면 ending도 함께 삭제되고, Nullify는 starting이 삭제되거나 링크가 삭제되어도 ending은 그대로 존재하고 단지 링크를 표현하는 XLink 엘리먼트만 잘못된 링크가 발생하지 않도록 변경된다. Blocking은 starting을 삭제하려고 할 때 ending이 있으면 starting을 삭제할 수 없다. ending에서 starting에 대해 위의 세 가지 요소를 적용시킬 때도 위의 의미가 동일하게 적용된다.
- 직접성(directiveness)은 starting과 ending이 제3자 링크에 의한 것이냐 아니냐를 나타내고 Direct, Indirect의 두 가지 측면이 있다. 이들은 XLink의 타입이 arc냐 아니냐에 따라 결정되어진다. 그림 6에서 (6,(7,8))은 5에 의해서 간접적으로 생긴 관계성이다.

5.2 참조 관계성 모델 지원

본 시스템에서는 데이타베이스 관리 시스템에서 제약조건을 정의하면 시스템에서 자동으로 제약조건을 유지시켜주는 것처럼 XLink 에 대해서 참조 관계성 시맨틱스를 제공하고 XLink 생성시 starting과 ending 사이에 시맨틱을 설정해주면 시맨틱에 따라 참조 무결성을 제공해준다.

3장에서 설명한 XLink 어트리뷰들을 살펴보면 XLink에는 XLink의 시맨틱스 정보를 담고있는 애트리뷰트로 role과 arcrole과 title이 있다. 이 중에서 XLink의 탐색정보와 관계 있는 것은 role과 arcrole이다. 본 논문에서는 이 role과 arcrole)을 이용하여 XLink에 관계성 시맨틱 옵션을 부여한다. 그림 7은 role과 arcrole을 DTD 형태로 표현한 것이다.

본 시스템에서는 먼저 starting과 ending 사이의 관계를 배타성과 다양성, 종속성, 직접성 네 가지 측면에 따라 starting과 ending, ending과 starting간의 관계를 나누어서 관계성 모델을 디자인하고 응용 프로그래머로 하여금 role과 arcrole을 등록하게 하는데 role과 arcrole을 등록할 때 해당하는 starting과 ending이 어떤 관계성 옵션을 갖는지를 함께 등록해서 ending 또는 starting이 링크가 삭제될 때, 링크의 role이나 arcrole에 해당하는 관계성 옵션에 따라 행동하게 한다.

관계성을 지원하기 위해서 XLink 카탈로그와 XLinkRole 카탈로그를 둬서 링크 정보와 관계성 정보를 유지한다. XLink 카탈로그는 XLink가 정의되면 자동적으로 그 정보를 카탈로그에 기록하는데 {starting Oid, startingURI, endingOid, endingURI, sourceOid, sourceURI}를 가지고 있다. source는 XLink 정보가 기록되어 있는 리소스를 가리킨다. OID는 XDOM 내에서 저장되어 있는 객체를 가리키고 URI는 starting이나 ending이 아직 XDOM 내에 저장되지 않았을 경우를 위한 것으로 문서 이름이나 문서 내에서의 특정 엘리먼트를 가리키는 XPointer 형식을 가지고 있다.

```
<!ELEMENT role EMPTY>
<!ATTLIST role
  name        CDATA                #REQUIRED
  type        (role|arcrole)       #REQUIRED
  endoption   (ED|SD|EN|SN|EB|SB)  #REQUIRED
  startoption (DT|NF|BK)           #REQUIRED>
```

그림 7 role과 arcrole의 DTD 표현

본 논문에서 제공하는 XLink 참조 관계성 타입은 두 가지가 있다. starting에서 ending을 가리키는 관계성 타입과 ending에서 starting을 가리키는 관계성 타입이다.

5.2.1 starting을 가리키는 관계성 타입의 시맨틱

starting을 가리키는 관계성 타입에서는 종속성 만을 표현한다. 배타성은 ending을 가리키는 관계성 타입에서 표현하였고 다양성과 직접성은 3장과 같은 방법으로 표현한다. starting을 가리키는 관계성 타입은 role 카탈로그에서 startoption을 통해 나타내는데 세 종류가 있고 각각의 의미는 다음과 같다.

• DT(Deletion)

ending을 삭제하면 starting도 함께 삭제한다. starting이 ending에 종속적이다. 주로 locator-type의 엘리먼트와 이 엘리먼트가 가리키는 객체 사이에서 일어날 수 있는 ending에서 starting으로 향하는 관계이다.

• NF(Nullify)

ending과 starting이 독립적이다. ending을 삭제하더라도 starting은 삭제되지 않고 단지 XLink 엘리먼트만 수정된다.

• BK(Blocking)

ending을 삭제할 때 참여하는 링크가 존재하고 starting이 존재하면 ending을 삭제할 수 없다.

5.2.2 ending을 가리키는 관계성 타입의 시맨틱

3 절에서 제시한 4가지 요소를 ending을 가리키는 관계성 타입에서 어떻게 표현할 수 있는지 알아보자. 다양성은 하나의 링크가 여러 개의 목적지를 가지는 것으로 이것은 XLink를 정의할 때 arc type의 XLink 엘리먼트의 from과 to 애트리뷰트에 해당하는 label을 가진 엘리먼트를 arc-type XLink 엘리먼트의 부모 엘리먼트가 몇 개 씩 가지고 있느냐에 따라 결정된다. 직접성은 role 카탈로그에서 role 타입에 의해 표현할 수 있다. 배타성과 종속성은 role 카탈로그에서 endoption을 사용해서 나타낸다. 이 endoption은 여섯 종류가 있고 이 endoption에 대한 의미를 알아보면 다음과 같다.

• ED(Exclusive Deletion)

*starting*과 *ending*의 관계가 1:n, 즉 *ending*이 단 하나의 링크에만 참여하고 starting이 삭제되거나 링크 정의가 삭제되면 이에 해당하는 ending도 함께 삭제된다. 그림 6에서 (16, 19)가 그 예로 16이나 xmlitem.xml 삭제되면 19도 따라서 삭제된다.

• SD(Shared Deletion)

ED와는 달리 starting과 ending의 관계가 m:n, 즉 다른 링크에 참여하고 있는 ending을 ending으로 가리킬 수 있다. 따라서 ending이 SD나 SN이나 SB인 role이나 arcrole에 의해 링크에 참여하고 있더라도 다른 링크에서 ending으로 참조할 수 있다. 하지만 ED, EN이

Algorithm 1 boolean delete(object *o*)

```
1: if o is starting resource with XLink l then
2:     if l.role.endoption is ED then
3:         if l is extended link then
4:             if l.role.type is arcrole then
5:                 delete l.ending refernce
6:                 modify l
7:             else if l.role.type is role then
8:                 modify l.arctype
9:             end if
10:        end if
11:            delete l.ending
12:    else if l.role.endoption is SD then
13:        if l is simple link or there exist one l then
14:            same as EN
15:        else if there exist other link l' and
               l'.ending == l.ending then
16:            modify l.arctype
17:        end if
18:    else if l.role.endoption is EN or SN then
19:        if l is extended link then
20:            modify l.arctype
21:        end if
22:    else if l.role.endoption is EB or SB then
23:        if l.ending is exists then
24:            return false
25         end if
26:    end if
27: else if o is ending resource with XLink l then
28:    if l.role.startoption is DT then
29:        if l is extended link then
30:            if l.role.type is arcrole then
31:                delete l.startingreference
32:                modify l
33:            else if l.role.type is role then
34:                modify l.arctype
35:            end if
36:        end if
37:        delete l.starting
38:    else if l.role.startoption is NF then
39:        if l is simple link then
40:            modify l.type → "none"
41:        else if l is extended link then
42:            modify l.arctype
43:        end if
44:    else if l.role.startoption is BK then
45:        if l.starting is exists then
46:            return false
47:        end if
48:    end if
49: end if
50: delete o
51: return true
```

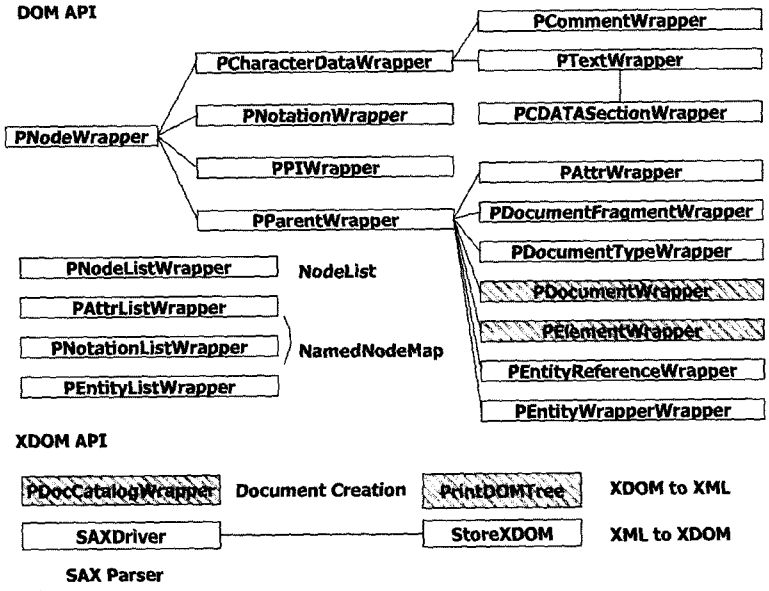

그림 8 DOM API + XDOM API

나 EB에 의해서 링크에 참여하고 있을 경우에는 참조할 수 없다. starting이나 링크가 삭제될 때 다른 링크가 없으면 ending도 함께 삭제된다.

• EN(Exclusive Nullify)

ending과 starting이 1:n이고 ending이 starting에 대해 독립적이기에 starting의 존재 여부가 ending의 존재 여부에는 영향을 미치지 않는다. 따라서 starting이나 링크가 삭제될 때 해당하는 XLink 엘리먼트만 수정해 준다.

• SN(Shared Nullify)

ending과 starting이 m:n이고 ending이 starting에 대해 독립적이기에 starting의 존재 여부가 ending의 존재 여부에는 영향을 미치지 않는다. 따라서 starting이나 링크가 삭제될 때 EN과 같이 XLink 엘리먼트만 수정해준다. locator-type 엘리먼트와 그 엘리먼트가 가리키는 객체 사이에 존재하는 ending을 가리키는 관계성이 이에 해당된다.

• EB(Exclusive Blocking)

starting과 ending이 1:n이고 starting을 삭제하려고 할 때 ending이 있으면 starting이나 링크를 삭제할 수 없다.

• SB(Shared Blocking)

ending와 starting이 m:n이고 starting을 삭제하려고 할 때 ending이 있으면 starting이나 링크를 삭제할 수 없다.

XML 리소스가 삭제될 때 *endoption*과 *startoption*에 따른 행동 양식은 Algorithm 1과 같다. 링크를 가리키는 객체는 staring이거나 ending 객체이다. 각각에 대하여 그 객체의 *role.endoption* 타입에 따라 그에 해당하는 동작을 수행한다. 1번째 줄을 보면 삭제되는 객체가 starting일 때 이것의 *role.endoption*이 각각 ED, SD, EN/SN, EB/SB일 경우에 대한 부분이다. 이것은 5.2.2절에 설명되어 있다. 또한 27번째 줄은 삭제되는 객차가 ending일 때 이것의 role.startoption이 각각 DT, NF, BK인 경우이며 이것은 5.2.1절에 설명되어 있다.

5.3 확장 링크와 XLink를 위한 DOM 트리 확장

본 논문이 구현되는 환경은 XDOM 저장장치이고 XDOM은 4장에서 설명한 바와 같이 DOM API[15]와 XDOM에서 제공하는 확장된 API(그림 8)로 응용 프로그래머가 XDOM에 XML 데이타를 저장하게 한다.

XDOM에서 XML 엘리먼트를 구현한 클래스는 그림 8의 API인 PElementWrapper와 실제 엘리먼트가 저장되는 클래스인 PElement 클래스가 있다. XLink를 엘리먼트를 저장하기 위하여 따로 별도의 클래스는 두지 않고 PElement와 PElementWrapper를 사용하여 XLink 엘리먼트를 저장한다. XLink 엘리먼트가 XLink 구문[2]에 유효한지, 어떤 엘리먼트가 XLink 엘리먼트인지를 알기 위해 XLink 클래스를 사용한다. XLink 엘리먼트를 등록하기 위하여 PDocumentWrapper 클래스에 createXLink() 함수를 추가하였다. createXLink()는 XLink 엘리먼트의 링크 정보를 XLink 카탈로그에 저장하는 역할을 한다.

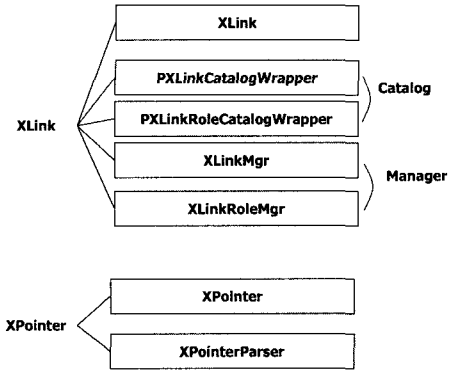

그림 9 XLink API

DOM 트리 확장은 starting을 DOM 트리로 따라갈 때 starting이 속한 링크가 존재하면 ending도 하나의 DOM 트리 일부분인 것처럼 따라갈 수 있게 하는 기능이다. xmlitem.xml을 DOM 파싱하면 그림 6의 xmlitem.xml에서 실선 부분에 해당하는 DOM 트리만 나타나지만 XLink에 의해서 DOM 트리 확장을 하면 굵은 점선에 해당하는 부분까지 같이 나타난다.

이 기능을 위해서는 먼저 XPointer 형태로 저장되어 있는 ending의 주소를 해석할 수 있어야 한다. 따라서 XPointer를 위한 클래스를 디자인하고 PrintDOMTree 클래스에 expand()를 추가하였다. expand()는 주어진 XML 문서의 엘리먼트들이 starting으로 참여하는 링크가 존재할 때 *ending*을 *starting*의 자식으로 만들어서 확장된 XML 문서를 만들어서 출력해준다.

XLink를 위해 XDOM API에서 변경되는 부분은 그림 8의 빗금 친 부분이고 추가하는 API는 그림 9와 같다.

6. 시스템 구현

전체 시스템 구조는 그림 10과 같다. XLinkMgr 클래스는 XLink가 생성되면 그 정보를 XLink 카탈로그에 저장하고 XLink를 유지시키고 XLink 참조 관계성을 유지하는 역할을 한다. 응용 프로그래머는 이 XLinkMgr를 통해서 XLink 메타 정보를 얻을 수 있고 생성한 XLink 정보를 카탈로그에 저장할 수 있다.

롤 메니저(role manager)는 role을 롤 카탈로그에 저장하고 role을 조작하는 역할을 하며, XLinkRoleMgr 클래스로 구현하였다.

XDOM 안에 저장된 XML 문서에 XLink를 추가하기 위해서는 다음과 같은 과정이 필요하다.

role 생성 → role에 endoption과 startoption 부여 → XLink를 추가할 XML 문서에 XLink 엘리먼트 생성 → XLink 엘리먼트에 등록된 role 부여 → XLink 생성

XDOM에서 XLink API를 이용하여 문서를 연결하는 XLink를 생성하는 예제는 6.2절에서 보인다.

6.1 XLink API

XDOM에서 XLink를 지원하기 위하여 사용하는 주요 클래스에 대한 API를 설명한다.

6.1.1 XLinkRoleMgr

XLinkRoleMgr 클래스는 role을 등록 및 삭제하고 초기 role을 등록하는 역할을 한다. 주요한 API는 다음과 같다.

- searchCat : role 카탈로그에서 특정 role이 어느 카탈로그의 어느 인덱스에 있는지 검색한다.

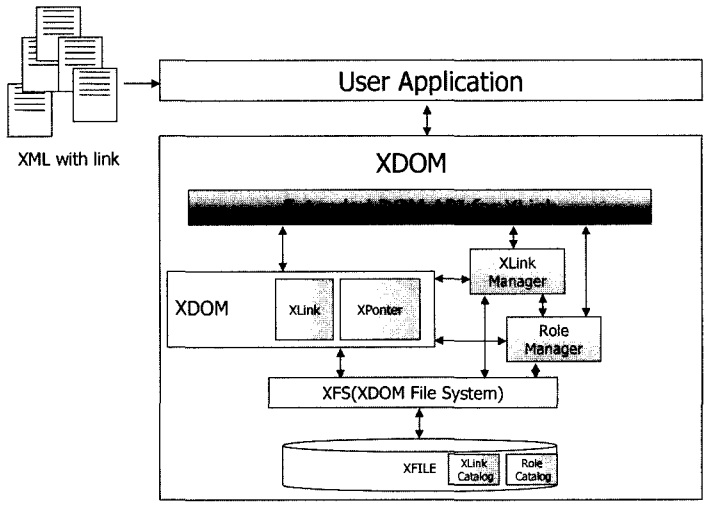

그림 10 XDOM에서의 XLink 지원 시스템 구조

- registerRole : role 카탈로그에 role을 등록한다.
- modifyRole : role 카탈로그에서 role을 수정한다.
- deleteRole : role 카탈로그에서 role을 삭제한다.
- getRole : role 이름에 해당하는role을 반환한다.

6.1.2 PXLinkRole

XLink를 정의하는 엘리먼트에는 role과 arcrole이라는 XLink의 시멘틱 정보를 나타내는 애트리뷰트가 있다. 사용자는 XLink를 만들기 전에 XLink에 부여할 role을 시스템에 등록하고 사용한다. PXLinkRole은 이런 role을 저장하는 클래스이다. 주요 API는 다음과 같다.

- setToRel : from} → to로의 관계성을 설정한다.
- setFromRel : to → from으로의 관계성을 설정한다.

6.2 XLink API를 사용한 응용 프로그램 예제

그림 2 relateditems.xml에서 XLink를 생성하고 xmlitem.xml이 삭제될 때 그림 11의 role이 표현하는 관계성에 맞게 유지시켜주는 응용 프로그램을 살펴보자. arcrole에 따라 xmlitem.xml이 삭제되면 htmlitem.xml이나 wwwitem.xml은 변화 없고 relateditems.xml에서는 링크를 정의하는 엘리먼트의 타입이 none으로 바뀌고 xmlitem.xml을 가리키는 locator-type의 엘리먼트가 삭제되어야 한다. 예제 6.1은 XLink API를 이용하여 프로그래밍한 예제이고 예제 6.2는 XLink API를 이용하지 않고 프로그래밍 한 예이다.

예제 6.1에서 XLink를 생성할 때는 DOM API를 이용해서 XLink에 관련된 엘리먼트들(relateditems, relitem, relitemlist)을 생성해 주고 이 중 가장 상위 엘리먼트인 relateditems만 createXLink()함수를 이용하여 등록(줄 26)해 주면 된다. xmlitem.xml이 삭제될 때 줄 29번 한 줄만 추가하면 시스템에서 자동으로 참조 관계성에 맞게 종속된 객체는 삭제하고 링크도 고쳐준다. 하지만 XLink API를 사용하지 않을 때는 예제 6.2처럼 XLink를 생성할 때 줄 29-35의 추가적인 코드가 들어가고 삭제할 때도 마찬가지로 줄 41-49를 추가해 주어야 한다. 위의 예제에서 XLink 라이브러리를 사용해서 훨씬 간단하게 XLink 응용 프로그램을 작성할 수 있음을 살펴볼 수 있다. XLink 사용의 자세한 예는 [16]에 나타나있다.

7. 결 론

본 논문에서는 XML 문서를 저장하는 객체 정보 저장소(object repository)에서 XLink를 지원해 주기 위한 시스템을 제안했다. 이 시스템은 XML 문서 저장을 위한 객체 저장소 시스템인 XDOM 상에 구현하였고 XLink를 지원해주기 위해서 본 시스템에서 제공해주는 기능은 참조 관계성, 확장 링크 및 XLink를 위한 DOM 트리 확장을 위한 마운트이다.

referitem Role	relateditemlist Role
name:referitem	name:relateditemlist
type:role	type:arcrole
endoption:SN	endoption:SN
startoption:DT	startoption:NF
referexam Role	showexam Role
name:referexam	name:showeaxm
type:role	type:arcrole
endoption:SN	endoption:ED
startoption:DT	startoption:NF

그림 11 예제 role

예제 6.1 *XLink API를 사용할 경우*

```
 1: CreateXLink
 2:
 3: // role 등록
 4: XLinkRoleMgr rmgr =
    XLinkRoleMgr.getXLinkRoleMgr();
 5: rmgr.register(role, referitem, SN, DT);
 6: // relateditemlist와 referexam, showexam도 마찬가지
    로 등록
 7:
 8: // 확장 링크를 저장하는 relateditems.xml 생성
 9: PDocumentWrapper source =
    PDocCatalogWrapper.insert(relateditems.xml);
10: PElementWrapper element =
    source.createElement(relateditemslink);
11: source.setDocumentElement(element);
12
13: PElementWrapper childElement =
    source.createElement(relateditems);
14: element.appendChild( childElement );
15: childElement.setAttribute( xlink:type, extended );
16: element = childElement;
17:
18: // 첫번째 relitem 생성
19: childElement = source.createElement(relitem);
20: element.appendChild(childElement);
21: childElement.setattribute( xlink:type, locator );
22: // 나머지 애트리뷰트도 21과 같은 방법으로 등록
23: // 두번째, 세번째 relitem 엘리먼트와 relitemlist도 위와
    같은 방법으로 생성
24:
25: // xlink 등록
26: source.createXLink(element);
27: end CreateXLink
28:
29: Delete xmlitem.xml
29: PDocCatalogWrapper.delete(students.xml);
30: end Delete
```

XLink가 링크하는 문서가 동일한 사이트내에 있다고 가정하고 그 때 일어날 수 있는 XLink 참조 문제(현수 참조, 쓰레기 문제)를 객체 정보 저장소 단계에서 해결

예제 6.2 *XLink API를 사용하지 않을 경우*

```
 1: CreateXLink
 2:
 3: Hashtable roleTbl = new Hashtable();
 4: Hashtable xlinkTbl = new Hashtable();
 5:
 6: roleTbl.add(studentlist, new role(role, referitem,
    SN, DT));
 7: // 나머지 role 등록
 8:
 9: // 확장 링크를 저장하는 relateditems.xml 생성
10: PDocumentWrapper source =
    PDocCatalogWrapper.insert(relateditems.xml);
11: PElementWrapper element =
    source.createElement(relateditemslink);
12: source.setDocumentElement(element);
13
14: PElementWrapper childElement =
    source.createElement(relateditems);
15: element.appendChild( childElement );
16: childElement.setAttribute( xlink:type, extended );
17: element = childElement;
18:
19: // 첫번째 relitem 생성
20: childElement = source.createElement(relitem);
21: element.appendChild(childElement);
22: childElement.setattribute( xlink:type, locator );
23: // 나머지 애트리뷰트도 22와 같은 방법으로 등록
24: storeXLink(childElement);
25: // 두번째,세번째 relitem 엘리먼트와 relitemlist도
    위와 같은 방법으로 생성
26: storeXLink(element);
27: end CreateXLink
28:
29: StoreXLink(Element element)
30: if(element .type = locator type) {
31:     // locator type 일 경우 실행 코드
32:     xlinkTbl.add(); }
33: // type이 locator를 제외한 다른 것일 경우 실행 코드
34: .....
35: end StoreXLink
36:
37: Delete xmlitem.xml
38: DeleteObj( xmlitem.xml );
39: end Delete
40:
41: DeleteObj(String docName)
42: long docid =
    PDocCatalogWrapper.searchDocOidbyDocName(docName);
43:
44: if(docid가 xlinkTbl에 존재할 경우) {
45:   if(docid.linkType = source) {
46:       // relationship option에 따른 실행 코드 }
47:   else if
48:   .....
49: end DeleteObj
```

하기 위하여 참조 관계성 모델을 디자인하고 구현했다. XLink에 관계된 리소스간의 starting-ending 관계를 Exclusiveness와 Multiplicity, Dependency, Directiveness 네 가지 측면에 따라 starting과 ending, ending과 starting간의 관계를 나누어서 관계성 모델을 디자인하고 이런 관계성 모델에 관련된 시맨틱 옵션을 XLink의 시맨틱 애트리뷰트인 role과 arcrole에 부여하도록 하였다. 이렇게 시맨틱 옵션이 부여된 role과 arcrole을 XLink 엘리먼트에서 사용함으로 XLink에 참여하는 리소스 간의 참조 무결성을 보장하였다. 응용 프로그래머로 하여금 role과 arcrole을 등록하게 하는데 role과 arcrole을 등록할 때 해당하는 starting과 ending이 어떤 관계 시맨틱을 갖는지를 같이 등록해주고 ending과 starting, 링크가 삭제될 때 role이나 arcrole에 해당하는 시맨틱 옵션에 따라 행동하게 한다.

그 외의 기능 확장 링크와 XLink를 위한 DOM 트리 확장을 위해 XLink와 Xpointer 클래스를 디자인하였고 구현에 있어서는 XLink 카탈로그와 role 카탈로그를 이용하여 XLink 리소스들 간의 관계를 저장하고 XDOM의 설계 방침에 따라 Wrapper 클래스와 Persistent 클래스로 나누어 구현을 하였다.

XLink 시맨틱 지원 시스템에 대한 향후 연구 방향은 다음과 같다. 첫째, 한 사이트에만 국한된 리소스들 사이의 참조 관계성이 아닌 웹상에서의 리소스들 사이에서의 참조 관계성 시맨틱을 디자인한다. 둘째, XLink 시맨틱이 제공될 경우 저하될 수 있는 시스템 성능을 향상시키기 위한 연구를 수행할 것이다.

참 고 문 헌

[1] Neil Bradley, *The XML Companion*, Addison-Wesley, 2 edition, 1999.

[2] Steve DeRose, Eve Maler, and David Orchard, editors, *XML Linking Language(XLink) Version 1.0*, http://www.w3.org/TR/xlink, 2001.

[3] Hugh C. Davis, Hypertext link integrity, *ACM Computing Survey*, 31, December 1999.

[4] 오동일, 최일환, 박상원, 김형주, XDOM: Java 기반 XML 객체 정보 저장소의 설계 및 구현, *http://oopsla.snu.ac.kr/RnD/2001/xdom.html*, 10 2001.

[5] Hugh C. Davis, Referential Integrity of Links in Open Hypermedia Systems, *Hypertext and hypermedia*, 1998.

[6] A. Rizk and L. Sauter, Multicard: An Open Hypermedia System, *ACM Conference on Hypertext*, 1992.

[7] Steven DeRose, Eve Maler, and Ron Daniel Jr., editors, *XML Pointer Language (XPointer) Version 1.0*, http://www.w3.org/TR/xptr, 2001.

[8] E. F. Codd, A Relational Model from Large

Shared Data Banks, *Communication of ACM*, 13(6), 1970.
[9] R. G. G. Catte and Douglas K.Barry, editors, *The Object Database Standard : ODMG 2.0*, Morgan Kaufmann, 1997.
[10] V. C. Storey, Understainding Semantic Relationships, *The VLDB Journal*, 2(4), 1993.
[11] Kendall Scott, *UML Explained*, Addison-Wesley, 2001.
[12] H. V. Jagadish, Integrity Maintenance in an Object-Oriented Database, *VLDB*, 1992.
[13] W. Kim, Cpmposite Object Support in an Object-Oriented Database, *OOPSLA*, October 1987.
[14] 이현주, 송하주, 이상원, 김형주, 객체지향 데이타베이스 시스템에서 확장된 관계성의 설계와 구현, *한국정보과학회논문지: 데이타베이스*, 27(3), 9 2000.
[15] Document Object Level (DOM) Level 1 Specification, October 1998.
[16] 박희경, 박상원, 김형주, XLink 라이브러리 사용자 매뉴얼, *http://oopsla.snu.ac.kr/RnD/2001/xlink/manual.doc*, 2001.

박 희 경
1995년 경북대학교 수학교육과(학사)
1998년 경북대학교 컴퓨터공학과(학사)
2002년 서울대학교 컴퓨터공학부(석사)
관심분야는 XML, 데이타베이스, 하이퍼미디어, 전자상거래 등

박 상 원
1994년 서울대학교 컴퓨터공학과(학사)
1997년 서울대학교 컴퓨터공학과(석사)
2002년 서울대학교 컴퓨터공학과(박사)
2002년 2월~2003년 2월 세종사이버대학교(전임강사). 2003년 3월~현재 한국외국어대학교 컴퓨터및정보통신공학부 전임강사. 관심분야는 XML, 객체지향시스템, 객체관계형 데이타베이스 등

김 형 주
정보과학회논문지 : 컴퓨팅의 실제
제 9 권 제 3 호 참조