XML 데이터를 위한 객체지향 데이타베이스 스키마 및 질의 처리

(Object-Oriented Database Schemas and Query Processing for XML Data)

요약

XML이 웹상에서 정보 교환의 표준으로 채택되면서 XML을 데이터베이스의 데이터로 보고 정보를 추출하는 분야가 주목받고 있다. 특히 최근에는 기존의 DBMS 시스템에 XML 데이터를 저장하여 기존의 DB 엔진으로 XML 질의를 처리하는 분야가 많이 연구되고 있다. 이때 주로 관계형 DBMS를 사용하는 연구가 많이 시도되고 있다. 본 연구에서는 객체 지향 데이터베이스 시스템이 XML 데이터를 저장하고 질의를 처리하는 또 하나의 해법이 될 수 있음을 보인다. 제안하는 방법은 XML의 DTD로부터 OODB의 클래스를 생성하고 이 클래스에 대하여 XML 질의를 수행한다. 특히, XML 데이터의 비정형적인 성질이 OO 모델의 계승(inheritance)을 이용하여 표현되어 질의 처리시에 유용하게 사용될 수 있음을 보인다.

Abstract

As XML has become an emerging standard for information exchange on the World Wide Web it has gained attention in database communities to extract information from XML seen as a database model. Recently, many researchers have addressed the problem of storing XML data and processing XML queries using traditional database engines. Here, most of them have used relational database systems. In this paper, we show that OODBSs can be another solution. Our technique generates an OODB schema from DTDs and processes XML queries. Especially, we show that the semi-structural part of XML data can be represented by the 'inheritance' and that this can be used to improve query processing.

1 서론

최근에 XML[1]이 웹상에서 정보 교환의 표준으로 채택되면서 XML을 데이터베이스의 데이터로 보고 정보를 추출하는 분야가 주목받고 있다. 즉, XML은 HTML과는 달리 그 자체에 데이터의 의미적인 정보를 가지고 있으므로 우리는 네트워크 상에 분산된 이질의(heterogeneous) 정보 근원으로부터 질의를 수행하고 필요한 정보를 추출해 낼 수 있다.

이러한 XML 데이터를 데이터베이스로 보고 질의를 수행하는 방법은 크게 두가지로 나눌 수 있다. 첫째는, XML 데이터는 그래프 기반의 비정형 데이터 모델의 한 인스턴스로 간주될 수 있으므로

비정형 데이터의 질의 언어와 질의 처리 기법을 이용하여 XML 데이터에 대한 질의 처리를 하는 방법이다.

둘째는, 기존의 데이터베이스 시스템에 XML 데이터를 저장하고, XML 질의를 기존 데이터베이스의 질의 언어로 변환하는 방법이다. 특히, 최근에는 XML 데이터를 RDBMS에 매핑하여 비정형 데이터 질의를 SQL 형태의 질의로 바꾸어 줌으로써 XML 질의를 처리하는 기법이 많이 제안되었다. 그런데 바꾸어 줌으로써 XML 질의를 처리하는 기법이 많이 제안되었다. 그런데 XML 테이타를 OODBMS에 저장하는 연구는 XML 언어의 모태라고 할 수 있는 SGML 테이타를 OODB에 매핑하는 연구[2] 이외에는 특별히 제안된 바가 없다.

본 논문에서는 XML을 OODB에 매핑하여 질의를 수행하는 방법을 제안한다. [2]의 방법에 비하여 독창적인 점은 OODBMS의 큰 장점인 계승(inheritance)을 이용한다는 점이다. 즉, [2]에서는 DTD의 각 엘리먼트에 대하여 클래스를 각각 생성하지만, 제안하는 기법은 DTD로부터 계승 정보를 뽑아내어 스키마를 생성한다. 이 방법은 클래스의 계승 관계를 이용하여 클래스를 구분함으로써 특정 클래스에 대한 질의를 수행할 때 객체 탐색 범위를 줄일 수 있는 장점을 가진다.

예를 들어 다음과 같은 DTD가 주어졌다고 하자. DTD(Document Type Descriptor)[3]는 XML 문서의 구조를 묘사하는 것으로 구체적인 내용은 3절에서 다룬다.

```
<!ELEMENT person (name, address, vehicle*,(school|company))>
<!ELEMENT name (firstname?, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT vehicle (model, company, gear?)>
<!ELEMENT model (#PCDATA)>
<!ELEMENT gear (#PCDATA)>
<!ELEMENT school (name, baseball-team?, person+,url?)>
<!ATTLIST school name CDATA #REQUIRED>
<!ELEMENT baseball-team (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT company (name, person+, url?)>
<!ATTLIST company name CDATA #REQUIRED>
<!ELEMENT alumni (name, year, school)>
<!ATTLIST alumni name CDATA #REQUIRED>
<!ELEMENT year (#PCDATA)>
```

그림 1: 예제 DTD

여기서 person의 경우를 보면 DTD로부터 우리는 person이 1.자가용을 가진 학생, 2.자가용을 가진 회사원, 3.자가용을 가지지 않는 학생, 4.자가용을 가지지 않는 회사원인 경우의 4가지 종류로 구분될 수 있음을 알 수 있다. 이러한 정보는 객체 지향 테이타베이스의 계승 개념으로 표현될 수 있다. 즉 person-A, person-B, person-C, person-D라는 클래스가 person이라는 일반 클래스로부터 계

승 받는 형태로 정의될 수 있다. 이렇게 스키마를 정의하면 OODB에서는 같은 타입의 객체는 클러스터링되는 효과를 가지므로 질의 처리시에 유용하게 사용될 수 있다. 예를 들어 질의가 자가용을 가지는 학생에 대한 질의라면 질의 처리기가 person-A 의 extents 만을 탐색할 수 있다.

본 논문에서는 이렇게 XML의 DTD로부터 계승을 이용한 OODB 스키마를 추출하여 질의를 처리하는 알고리즘을 보인다. 논문의 구성은 다음과 같다. 2절에서는 관련 연구를 다루고, 3절에서는 DTD로부터 OODB의 클래스를 추출하는 기법을 제시한다. 4절에서는 질의 언어를 다루고 5절에서 결론을 내린다.

2 관련연구

최근에 XML이 인터넷 상에서 데이타를 표현하는 표준으로 인식되면서 XML 데이타에 대한 저장, 질의 언어, 질의 처리 등의 분야가 활발히 연구되고 있다. 앞서 언급했듯이 XML 데이타 처리 방법은 1.XML을 비정형 데이타 모델[4,5]의 한 인스턴스로 보고 비정형 데이타 모델의 질의 엔진을 이용하는 방법[6,7]과 2.기존의 데이타베이스 시스템에 XML을 저장하여 기존 시스템의 질의 엔진을 이용하는 방법[8–10,2]의 두 가지 방법으로 구분할 수 있다.

비정형 데이타의 질의 엔진을 이용하는 방법은 데이타 모델이 명확하여 질의 언어의 의미가 전달되기 쉽다는 장점을 가지지만 오랜동안 연구되어온 DBMS의 여러 기능 들을 사용 할 수 없다는 단점을 가진다. 이러한 연구 중에 [11–15]는 비정형 데이타로부터 스키마 정보를 뽑아내어 질의 처리에 이용하는 방법으로 본 논문에서 다루는 기법이 DTD로부터 OODB에 매핑되는 스키마 정보를 뽑아내는 연구이기 때문에 밀접한 관련이 있다. 이중 [11,12]에서는 그래프 스키마라는 개념을 제시하였는데 이것은 비정형 데이타의 그래프 구조에 대한 부분적인 정보를 담고 있는 그래프 스키마를 이용하여 질의 탐색 범위를 잘라내거나(query pruning) 객체 지향 데이타베이스의 클래스 extents와 같은 상태 extents(state extents)를 이용하여 질의를 재구성(query rewriting) 하는 방법이다. 이 방법은 정적으로 그래프 스키마를 정의하고 여기에 대응하는(conform) 데이타에 대한 질의를 처리하는 성격을 가진다.

반면에 Dataguide[13,14]는 데이타를 중심으로 동적으로 존재하는 모든 경로에 대하여 인덱스를 거는 방법을 제안하였는데 스키마에 대한 정보를 미리 알 수 없거나, 관리할 수 없는 경우의 데이타에 대하여 유용하게 쓰일 수 있다. 이와 같은 방법들은 비정형 데이타를 위한 질의 엔진을 사용하는 방법이므로 본 논문의 기법과는 접근 방법이 다르다고 할 수 있다.

기존 데이타베이스 시스템을 이용하는 방법은 주로 RDBMS에 저장하는 방법[8–10]이 연구되었다. 이 중에서 [8,10]의 방법은 DTD로부터 스키마를 뽑아내는 기법으로 본 논문의 기법과 밀접하게 연관되어있다. 특히 우리는 [10]의 중복되는 엘리먼트를 그의 부모 관계 엘리먼트의 애트리뷰트로 하

는 inlining 기법을 적용하였다. 이때 관계형 모델보다 유연성(flexibility)을 가짐을 보인다. [9]의 방법은 그래프 형태를 DBMS에 저장하는 방법으로 제안하는 기법과는 다른 방법이다.

OODBMS에 저장하는 방법은 XML의 전신인 SGML을 저장하는 [2]의 방법이 제안되었다. 이 방법은 DTD로부터 각 엘리먼트에 대하여 클래스를 생성하는 방법을 보이고 기존 객체 지향 데이타베이스의 질의 언어의 확장된 형태를 제시하였다. 그렇지만 OO 모델의 큰 장점이라 할 수 있는 계승은 클래스 디자인 단계에서 이용하지 않고 있다.

3 DTD로부터 OODB 스키마로의 매핑

기존에 제안된 DTD로부터 OODB 스키마로의 매핑[2]은 DTD의 엘리먼트당 하나씩의 클래스를 생성한다. 이때 or 연산은 union 타입으로 모델링되고 '+'나 '*' 연산은 리스트로 표현된다. 그림 2는 그림 1의 DTD에 대한 OODB 스키마를 보인다.

```
class Person public type tuple(name:Name, address:Address,
       vehicle:list(Vehicle),union(school:School,company:Company))
class School public type tuple(name:string,
       baseball-team:Baseball-team,person:list(Person),url:Url)
class Company public type tuple(name:string,person:list(Person),
       url:Url)
class Vehicle public type tuple(model:Model,company:Company,
       gear:Gear)
class Alumni public type tuple(name:string,year:Year
       school:School)
class Name public type tuple(firstname:Firstname,lastname:Lastname)
class Firstname inherit Text
class Lastname inherit Text
class Address inherit Text
class Baseball-team inherit Text
class Url inherit Text
class Model inherit Text
class Gear inherit Text
class Year inherit Text
```

그림 2: OODB 스키마

이러한 방법은 다음과 같은 문제점을 가진다.

- DTD의 각 엘리먼트 당 하나의 클래스를 생성하므로 대개 많은 수의 클래스가 생성된다. 여기서 의미적으로 한 클래스에 포함될 수 있는 항목도 개개의 클래스로 나누어지는 경우가 많다. 예를 들어 클래스 Name과 클래스 Firstname, Lastname 등은 의미적으로 클래스 Person의 애트리뷰트로 들어갈 수 있다.

- 이 방법은 객체 지향 데이타베이스의 큰 장점이라고 할 수 있는 계승을 클래스 디자인시 이용하

고 있지 못하다. 예를 들어, Person 클래스는 계승을 이용하여 디자인 한다면 의미상으로 Person의 상위 클래스와 이로부터 계승받아 Student, Employee의 하위 클래스로 디자인될 수 있다. 이렇게 계승을 이용하여 클래스를 디자인 한다면 질의 처리시에 유용하다. 예를 들어, Person의 하위 클래스 Student에 대한 질의 처리시에 Person 타입의 객체 모두를 탐색하는 것이 아니라 Student 클래스의 인스턴스 만을 탐색할 수 있다.

- DTD의 '*' 연산이나 '?' 연산에 대하여 그 애트리뷰트 값이 없는 인스턴스의 경우에 null 값으로 채워지게 된다. 따라서 데이타의 분포에 따라 null 값을 가지는 데이타가 많아질 수 있다. 이것은 메모리의 효율성 면에서 불리하다.
- '|' 연산에 대하여 기존의 연구[2]에서는 union 타입으로 매핑되었으나 ODMG 모델에서는 union 타입을 직접 제공하지 않는다. 따라서 ODMG-compliant한 OODB에 그대로 적용할 수 없다.

따라서 제안하는 알고리즘은 기존 알고리즘을 다음과 같이 개선하여 이와 같은 문제점을 해결하였다.

- 각 엘리먼트에 대하여 각각 클래스를 만들지 않고 RDB의 inlining 기법을 적용하여 클래스의 개수를 줄인다.
- DTD로부터 각 엘리먼트를 구분하여 이 정보로부터 계승을 이용하여 클래스를 재구성한다.

3.1 Class inlining 기법

클래스 inlining 기법은 [10]에서 제안된 RDB의 inlining 기법을 ODB에 적용하는 기법이다. [10]에서는 DTD로부터 특정 엘리먼트의 자손 노드를 그 엘리먼트에 inlining 시킴으로써 릴레이션의 개수를 가능한 줄였다.

제안하는 기법이 [10]에서 제안된 기법과 다른 점은 두가지가 있다. 첫째는, 기존 관계 모델에서는 애트리뷰트의 값이 집합인 경우를 지원하지 않으므로 반복 연산자 '*'나 '+'가 있는 경우에 연산자의 엘리먼트에 대하여 각각 릴레이션을 생성한 후 외래키(foreign key)를 이용하여 릴레이션의 관계를 표현한다. 예를 들면, 그림 3에서 person 엘리먼트는 vehicle의 집합을 애트리뷰트로 가지므로 RDB의 방법에서는 vehicle과 person 엘리먼트에 대한 릴레이션을 만든 후 vehicle이 person을 가르키는 애트리뷰트를 인위적으로 생성한다. 그렇지만 ODB에서는 애트리뷰트의 값으로 list와 같은 집합 타입을 지원하므로 person이 그 애트리뷰트로 vehicle을 집합으로 가지도록 표현할 수 있다.

둘째는, [10]에서는 릴레이션 간의 관계를 표현하기 위하여 죠인 애트리뷰트를 시스템 내부에서 만들어 주었지만 ODB는 클래스 간의 관계를 직접 포인터로 표현할 수 있기 때문에 인위적인 죠인 애트리뷰트의 생성이 필요없다.

그러면 주어진 DTD로부터 class inlining 기법으로 클래스를 생성하는 방법을 살펴본다. 먼저,

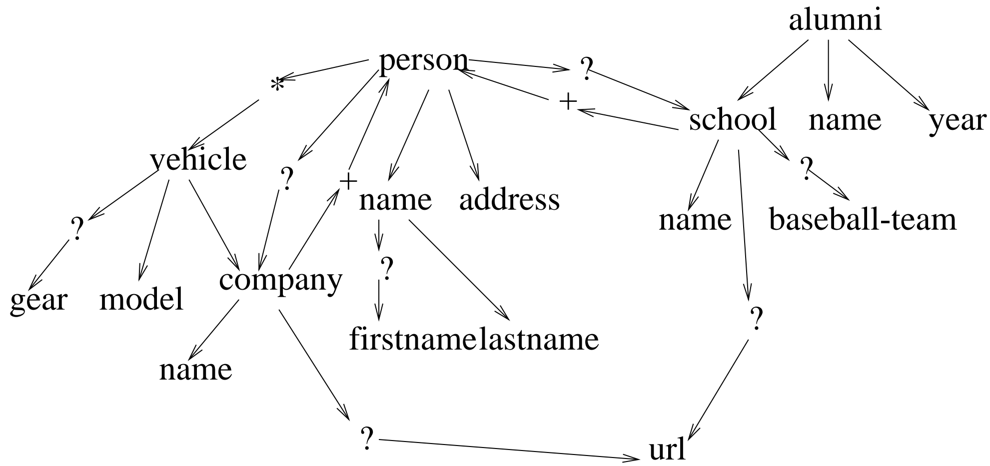


그림 3: A DTD graph

[10]에서와 같이 그림 1에 대하여 DTD 그래프를 그리면 그림 3과 같다. DTD 그래프의 노드는 DTD에 나타나는 엘리먼트, 애트리뷰트, 연산자로 구성되고, 에지는 그들간의 관계를 표현한다. 이때 각 엘리먼트는 그래프 상에 한 번씩만 나타나고, 애트리뷰트와 연산자는 DTD에 나타나는 만큼 나타난다. 이때 DTD는 단순화된 형태이다[10]. 즉, ","나 "|"와 같은 이진 연산자는 다른 연산자의 내부에 나타나지 않도록 변환된 형태이다. 예를 들어 $(e_1|e_2)$의 형태는 $e_1?, e_2?$의 형태로 변환된다.

DTD 그래프로부터 먼저 생성할 클래스를 결정하는데 다음의 규칙을 따른다.

1. in-degree 에지 수가 0이면 그 엘리먼트에 대하여 클래스를 생성한다. 이것은 클래스를 생성하지 않으면 이 엘리먼트를 표현할 수 없기 때문이다.
2. '*'나 '+'의 반복 연산자 밑의 노드에 대하여 클래스를 생성한다. 이것은 이 클래스를 애트리뷰트로 가지는 엘리먼트를 위하여 필요하다.
3. in-degree 에지 수가 1 인 경우에는 inline 시킨다.
4. in-degree 에지가 1인 상호 재귀적인 참조 관계가 있는 엘리먼트들은 그 중의 하나를 클래스로 만든다.
5. in-degree 에지가 2개 이상인 경우에는 클래스를 생성한다.

이렇게 클래스 inlining 기법을 적용하여 OODB의 스키마를 생성하면 다음과 같다.

3.2 계승을 이용한 클래스 분류

XML 데이타는 정규적인 스키마를 가지지 않고 비정규적일 수 있다는 특징을 가진다. 즉, 특정 애트리뷰트가 생략 가능할 수도 있고 같은 엘리먼트라도 다른 형태로 애트리뷰트를 가질 수 있다. 이 절에서는 이렇게 비정규적인 부분을 DTD로부터 뽑아내어 OODBMS의 계승(inheritance)를 이용하여 표현하는 방법을 다룬다.

```
class Person public type tuple(name.firstname:string,
      name.lastname:string,address:string,vehicle:list(Vehicle),
      union(school:School,company:Company))
class School public type tuple(name:string,
      baseball-team:string,person:list(Person),url:Url)
class Alumni public type tuple(name:string,
      year:String,school:School)
class Company public type tuple(name:string,person:list(Person),
      url:Url)
class Url inherit Text
class Vehicle public type tuple(model:string,company:Company,
      gear:string)
```

그림 4: OODB 스키마

3.2.1 DTD 오토마타

먼저 앞절에서 클래스 생성에 이용된 엘리먼트 e에 대한 DTD를 $(n\colon P)$로 추상화한다. 여기서 N을 엘리먼트 이름의 집합이라 할때, $n \in N$이고, P는 N에 대한 정규식 또는 PCDATA, 즉 문자열이다.

DTD의 한 엘리먼트 e의 DTD $(n : P)$에 대하여 정규식 P는 다음과 같이 다섯 가지 종류로 나눌 수 있다. r, r_1, r_2를 정규식이라고 하고 $L(r)$, $L(r_1)$, $L(r_2)$를 그 정규식이 나타내는 언어라고 가정한다.

1. $r = r_1, r_2$ 형태: r이 나타내는 언어 $L(r)$은 $L(r_1)$과 $L(r_2)$의 조합(concatenation)을 나타낸다.
2. $r = r_1|r_2$ 형태: $L(r)$은 $L(r_1)$ 또는 $L(r_2)$가 된다.
3. $r = r_1^+$ 형태: 같은 구조의 반복을 나타낸다.
4. $r = r_1^*$ 형태: 이 형태는 같은 구조가 하나도 없거나 반복을 나타낸다.
5. $r = r_1?$ 형태: 이 형태는 특정 구조가 있을 수도 있고 없을 수도 있음을 나타낸다.

이렇게 다섯 가지의 형태 중에서 엘리먼트를 그가 가지는 애트리뷰트에 따라 구분할 때 정보가 되는 형태는 2,4,5의 경우이다. 왜냐하면 1,3의 형태는 그 DTD에 대응하는 XML 테이타가 애트리뷰트를 같은 형태로 가지기 때문이다. 그러므로 본 논문에서는 DTD로부터 엘리먼트를 구분하는데 필요한 정보 만을 뽑아내기 위하여 다음과 같은 완화된 정규식을 정의한다.

정의 1 (완화된 정규식(relaxed regular expression)) *완화된 정규식은 다음의 다섯 가지 규칙에 의하여 구해진다.*

1. $r_1, r_2 \Rightarrow\ r_1, r_2$

2. $r_1|r_2 \Rightarrow\ r_1|r_2$

3. $r+ \Rightarrow\ r$

4. $r* \Rightarrow\ r+|\perp \Rightarrow\ r|\perp$ *(3에 의하여)*

5. $r? \Rightarrow \ r|\perp$

예제 1 *그림 1에서 person에 대한 항목은 (person: (name, address, vehicle*, (school|company)))로 추상화되고 완화된 정규식을 적용하면* $(person:\ (name, address, (vehicle|\perp), (school|company)))$*가 된다.*

알고리즘 1 DTD 오토마타의 구성

1: **입력:** 완화된 정규식 $r = n_iP_i^{'}$
2: **출력:** 오토마타 M_i
3: **procedure** Make_DTD_Automata(regular expression r)
4: **if** $r = a\ (a \in \sum)$ **then**
5: 오토마타 $M = (\{q_0, q_f\}, \sum, \delta, q_0, \{q_f\})$을 생성, 이때 δ는 다음과 같이 정의됨

1. $\delta(q_0, a) = q_f$; {그림 5}

6: return M;
7: **else if** $r = r_1|r_2$ **then**
8: $M_1 = (Q_1, \sum_1, \delta_1, q_1, F_1) \leftarrow$ Make_DTD_Automata(r_1);
9: $M_2 = (Q_2, \sum_2, \delta_2, q_2, F_2) \leftarrow$ Make_DTD_Automata(r_2);
10: 오토마타 M_1과 M_2로부터 새로운 오토마타 $M_i = (Q_1 - \{q_1\} \cup Q_2 - \{q_2\}, \sum_1 \cup \sum_2, \delta, [q_1, q_2], F_1 \cup F_2)$를 생성, 이때 δ는 다음과 같이 정의됨

1. $\delta(q, a) = \delta_1(q, a)$ for $q \in Q_1 - \{q_1\}$ and $a \in \sum_1$,
2. $\delta(q, a) = \delta_2(q, a)$ for $q \in Q_2 - \{q_2\}$ and $a \in \sum_2$,
3. $\delta([q_1, q_2], a) = \delta_1(q_1, a)$ where $a \in \sum_1$,
4. $\delta([q_1, q_2], a) = \delta_2(q_2, a)$ where $a \in \sum_2$;

11: **else** { $r = r_1, r_2$ }
12: $M_1 = (Q_1, \sum_1, \delta_1, q_1, F_1) \leftarrow$ Make_DTD_Automata(r_1);
13: $M_2 = (Q_2, \sum_2, \delta_2, q_2, F_2) \leftarrow$ Make_DTD_Automata(r_2);
14: M_1의 종료 상태 F_1을 $f_1, f_2, ..., f_m\ (m \geq 1)$이라 하자. 오토마타 M_1과 M_2로부터 새로운 오토마타 $M_i = (Q_1 - F_1 \cup Q_2 - \{q_2\} \cup \{[f_1, q_2], [f_2, q_2], ..., [f_m, q_2]\}, \sum_1 \cup \sum_2, \delta, q_1, F_2)$를 생성, 이때 δ는 다음과 같이 정의됨

1. $\delta(q, a) = \delta_1(q, a)$ for $q \in Q_1 - F_1$, $\delta_1(q, a) \neq f_k$ (where $1 \leq k \leq m$), and $a \in \sum_1$,
2. $\delta(q, a) = \delta_2(q, a)$ for $q \in Q_2 - q_2$ and $a \in \sum_2$,
3. $\delta([f_k, q_2], a) = \delta_2(q_2, a)$ for all k(where $k = 1, 2, ..., m$) and $a \in \sum_2$,
4. $\delta(q_f, a) = [f_k, q_2]$ for all q_f which satisfies $\delta_1(q_f, a) = f_k$(where $1 \leq k \leq m$) and $a \in \sum_1$;

15: **end if**
16: return M_i

이 완화된 정규식으로 부터 DTD 오토마타를 다음과 같은 방법으로 생성한다. DTD의 각 엘리먼트 (n_i, P_i)에 대하여 P_i에 완화된 정규식을 적용한 결과를 $(n_i, P_i^{'})$이라 한면, 새로운 정규식 $n_iP_i^{'}$[1]

[1] $n_iP_i^{'}$는 $n_i, P_i^{'}$와 같이 조합(concatenation)을 나타낸다.

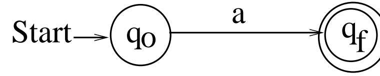


그림 5: $r = a$ 형태

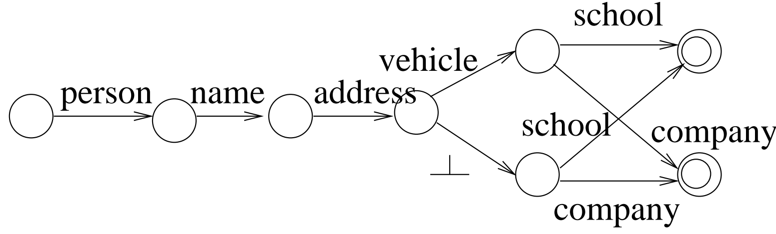


그림 6: DTD 오토마타

에 대하여 오토마타 A_i를 알고리즘 1에 따라서 생성한다. 여기서 생성한 오토마타는 각 엘리먼트를 그 각각이 가지는 애트리뷰트에 따라 구분하는데 쓰인다. 오토마타를 위한 기호는 [16]을 따르는데 오토마타는 5개의 항목을 가지는 $M = (Q, \sum, \delta, q_o, F)$으로 나타내지고 각 항목은 다음을 의미한다.

- Q: 유한개의 상태(state)의 집합을 나타낸다.
- $\sum$: 유한개의 입력 알파벳, 여기서는 DTD에 나타나는 엘리먼트의 이름과 $\perp$을 나타낸다.
- q_o: $q_o \in Q$를 만족하는 시작 상태(start state)를 의미한다.
- F: $F \subseteq Q$를 만족하는 종료 상태(final state)의 집합을 의미한다.
- δ: $Q \times \sum \rightarrow Q$의 전이 함수(transition function)를 나타낸다.

정리 1 *알고리즘 1에 의해 생성되는 오토마타 M은 항상 존재하고, M이 받아들이는 언어를 $L(M)$이라고 하고, 해당 정규식 r이 나타내는 언어를 $L(r)$이라고 하면 $L(M) = L(r)$이다.*

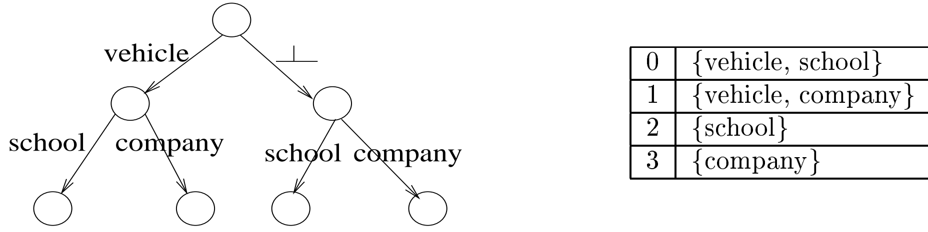
증명은 생략한다.

예제 2 *앞의 person에 대한 DTD에서 완화된 정규식을 적용한 결과인 $(person : (name, address, (vehicle|\perp), (school|company)))$에 대하여 알고리즘 1을 수행하면 그림 6과 같다.*

3.2.2 DTD 오토마타를 이용한 DTD 엘리먼트의 구분

DTD 오토마타는 완화된 정규식에서 생성되었으므로 DTD 정규식의 조합(concatenation)과 OR 조건만을 나타내게 된다. 이중에서 OR 조건의 분기점을 나타내는 부분이 엘리먼트를 구분하는 분기점이 되므로 이러한 분기점의 레이블을 기억하면 이를 통하여 엘리먼트를 구분할 수 있다.

알고리즘 2는 이러한 레이블을 저장하는 트리를 구성하는 알고리즘을 나타낸다. 알고리즘 2는 클래스 생성에 이용된 엘리먼트에 대한 DTD 오토마타를 입력으로 받아 분류 트리를 생성한다. 즉,



0	{vehicle, school}
1	{vehicle, company}
2	{school}
3	{company}

그림 7: A classification tree and a classification table

알고리즘 2는 주어진 오토마타의 시작 상태(start state)에서 종료 상태(final state)까지 재귀적으로 탐색하면서 트리를 생성한다. 알고리즘에서 transition(state)는 $\delta(q,a) = p$의 전이 함수가 있을때 $transition(q) = p$를 리턴하는 함수를 나타낸다. No_effect_label는 주어진 엘리먼트에 대하여 그가 가지는 레이블이 엘리먼트를 구분하는데 영향을 미치지 않는 레이블의 집합을 가지는 변수이다. 예를 들어 person에 대한 No_effect_label은 {person, name, address}이 되는데 이것은 person의 애트리뷰트 중에서 구조적인 부분을 나타낸다. 즉 모든 person 엘리먼트는 이 애트리뷰트들은 가진다.

알고리즘 2 DTD 엘리먼트 구분 트리 생성

1: **입력:** 상태(state) s, 오토마타 $M = (Q, \sum, \delta, q, F)$
2: **출력:** 분류 트리 T
3: **procedure** Make_classification_tree(state s, automaton M)
4: **if** $s \in F$ **then**
5: 상태 s에 대응되는 노드 s'을 생성;
6: return s';
7: **else**
8: **if** $transition(s)$가 둘 이상의 상태를 가짐 **then**
9: 상태 s에 대응되는 노드 s'을 생성;
10: s'의 루트를 가지고, 그 자식 노드로 $w \in transition(s)$인 모든 노드 w에 대하여, 에지로는 transition(s)에 있는 입력 알파벳 a를 가지고, Make_classification_tree(w,M)를 부 트리(sub-tree)로 가지는 트리 T를 생성;
11: return T;
12: **else**
13: $s \leftarrow \delta(s,a)$;
14: $No_effect_label = No_effect_label \cup a$;
15: Make_classification_tree(s,M);
16: **end if**
17: **end if**

그림 7은 알고리즘 2에 따라 생성한 DTD 오토마타의 person에 대한 분류 트리와 해당 분류 테이블을 나타낸다. 예를 들면 DTD의 한 엘리먼트인 person은 그가 가지는 레이블에 따라 {vehicle, school}, {vehicle, company}, {school}, {company}의 4그룹으로 나누어진다. 이렇게 엘리먼트를 구분하면 이것은 객체 지향 데이타 모델에서 계승(inheritance)으로 표현될 수 있다. 예를 들어 그림 7에서 person은 No_effect_label에 속하는 name, address를 애트리뷰트로 가지는 상위 클래스(superclass) Person과 이로부터 계승받아 그 애트리뷰트로 각각 {vehicle, school}, {vehicle, company}, {school},

{company}을 가지는 하위 클래스(subclass) Person0, Person1, Person2, Person3로 모델링 될 수 있다. 이와같이 school, company, vehicle에 대해서도 적용하여 클래스를 생성하면 다음과 같다.

```
class Person public type tuple(name.firstname:string,
       name.lastname:string,address:string)
class Person0 inherit Person type tuple(vehicle:list(Vehicle),
       school:School)
class Person1 inherit Person type tuple(vehicle:list(Vehicle),
       company:Company)
class Person2 inherit Person type tuple(school:School)
class Person3 inherit Person type tuple(company:Company)
class School public type tuple(name:string,person:list(Person))
class School0 inherit School type tuple(baseball-team:string,
       url:Url)
class School1 inherit School type tuple(baseball-team:string)
class School2 inherit School type tuple(url:Url)
class Company public type tuple(name:string,person:list(Person))
class Company0 inherit Company type tuple(url:Url)
class Vehicle public type tuple(model:string,company:Company)
class Vehicle0 inherit Vehicle type tuple(gear:string)
class Alumni public type tuple(name:string,
       year:String,school:School)
class Url inherit Text
```

그림 8: OODB 스키마

여기서 기반 클래스 Person과 School를 보면 그 애트리뷰트로 No_effect_label에 속하는 애트리뷰트 집합을 가지는데 DTD 구조상 Person 클래스는 인스턴스를 가지지 않는 반면에 School 클래스는 그 인스턴스를 가지게 된다.

4 질의 언어

XML 데이타에 대한 질의 언어로는 여러 가지가 제안되었는데 예를 들면, XML-QL[17], UnQL[7], Lorel[18] 등이 있다. 이러한 XML 질의 언어는 OODB의 질의 언어인 XSQL[19], OQL[20] 등의 경로식(path expression)을 확장한 형태라고 할 수 있는 정규 경로식(regular path expression)을 기반으로 한다.

정의 2 *(정규 경로식(Regular Path Expression)) 정규 경로식은 $H.P$ 형태로*

1. *H는 객체 이름 또는 객체를 표시하는 변수,*
2. *P는 XML 문서의 엘리먼트나 애트리뷰트의 레이블에 대한 정규식이다. 즉, $P = label|(P|P)|(P.P)|P*$이다.*

그런데 이러한 정규 경로식을 경로식 기반의 객체 지향 데이타베이스 질의 언어로 바꾸기 위해서는 다음의 두가지 문제를 해결하여야 한다. 첫째는, 정규 경로식의 $(P|P)$에서 기인하는 or 연산 문제이다. OODB의 경로식(path expression)[19]에서는 기본적으로 or 연산을 지원하지 않으므로 이 연산을 처리하는 루틴이 필요하다. 이것은 [21]에서 제안된 '|' 연산 제거 기법(alternation elimination)으로 해결 가능하다. 예를 들면 $person.(school|company).name$은 $(person.school.name) \cup (person.company.name)$으로 바뀔 수 있다.

둘째 문제는 정규 경로식의 $P*$에서 기인하는 재귀 질의의 문제이다. 이 문제는 [21]에서 데이타베이스의 구조적인 요약 정보를 가진 DataGuide[13]를 이용하여 컴파일시에 실제 가능한 경로식으로 질의를 변환하는 기법이 제안되었다. 우리도 이와 비슷한 방법으로 Dataguide 대신 DTD를 이용한 기법을 제안한다. 본 논문에서는 다음과 같이 단순하면서도 일반적인 XML 질의를 대부분 포함할 수 있는 단순 정규 경로식(simple regular path expression)을 정의하여 이 단순 정규 경로식이 기존의 OODB의 질의로 바꾸어지는 기법을 보인다. 제안하는 방법은 비슷하게 정규 경로식에도 적용될 수 있을 것이다.

정의 3 *(단순 정규 경로식(Simple Regular Path Expression)) 단순 정규 경로식은* $H.p_1.p_2.....p_n$*의 형태로*

1. *H는 객체 이름 또는 객체를 표시하는 변수,*
2. p_i $(1 \leq i \leq n)$*는 XML 문서의 엘리먼트나 애트리뷰트의 레이블, 또는 임의 길이의 레이블 연속을 나타내는* $*$*이다.*

4.1 '*' 연산이 없는 단순 경로 질의의 변환

이 절에서는 앞서 정의한 단순 정규 경로식에서 '*' 연산이 없는 경우를 다룬다. 이 경우는 쉽게 ODB의 질의로 바뀌어 질 수 있다. 예를 들어 다음과 같은 lorel-like한 XML 질의를 처리한다고 하자.

```
select X.name.firstname, X.name.lastname
from person X, X.vehicle Y
where X.address = "Seoul", Y.model = "EF-Sonata", Y.gear="auto"
```

이 질의는 어떤 사람이 서울에 살면서 그 사람이 가진 자동차의 모델이 "EF-Sonata"이고, 그 자동차가 자동 변속기인 경우 그 사람의 firstname과 lastname을 출력하는 질의이다. 주어진 질의에 대하여 질의 처리기는 알고리즘 3을 이용하여 질의 처리에 필요한 부클래스(subclass)를 결정한다.

예를 들어, 클래스 Person에 대하여 $S = \{name, address, vehicle\}$, $S2 = \{vehicle\}$ 이고, 그림 7에서 $T[person][0] = \{vehicle, school\}$이 된다. 따라서 클래스 Person0에 해당하는 인덱스 0이 $result_index_set$에 추가된다. 이런 방법으로 주어진 질의는 다음과 같은 객체 지향 데이타베이스 질의 언어로 변환된다.

알고리즘 3 부클래스(subclass) 결정

1: **입력:** from 절에 언급된 클래스 C, 질의 q, 분류 테이블 $T[class][index]$
2: **출력:** 질의 처리시 필요한 부클래스에 해당하는 인덱스의 집합
3: **procedure** Decide_subclasses(a class C, a query q, a classification table $T[class][index]$)
4: $result_index_set \leftarrow \phi$;
5: $S \leftarrow$ the set of child labels of C;
6: $S2 \leftarrow S - No_effect_label$ of C;
7: **for all** $T[C][index]$ where $1 \leq index \leq$ number of groups corresponding to C **do**
8: **if** $S2 - T[C][index] = \phi$ **then**
9: $index$를 $result_index_set$에 추가;
10: **end if**
11: **end for**
12: return $result_index_set$;

```
select tuple(f:p."name.firstname",l:p."name.lastname")
from p in (Person0 or Person1),y in (p.Vehicle and Vehicle0)
where p.address = "Seoul", y.model = "EF-Sonata", y.gear="auto"
```

질의 결과를 다시 XML 문서로 바꾸는 방법은 [10]의 방법을 그대로 적용할 수 있다. 예제의 경우에 질의 결과가 $\{(Serge, Abiteboul), (Victor, Vianu)\}$라면 적당한 태그를 첨가하여 다음과 같이 XML 문서를 만들 수 있다.

```
<person>
    <name>
        <firstname> Serge </firstname>
        <lastname> Abiteboul </lastname>
    </name>
    <name>
        <firstname> Victor </firstname>
        <lastname> Vianu </lastname>
    </name>
</person>
```

좀더 복잡한 경우, 즉 예를 들면 태그에 변수가 있는 경우 등에도 [10]의 방법으로 XML 문서를 만들 수 있다.

4.2 '*' 연산을 포함한 질의의 변환

다음은 단순 정규 경로식에서 '*'가 존재하는 경우를 다룬다. 임의의 경로에 매칭될 수 있음을 나타내는 '*'는 스키마 구조를 알지 못하는 사용자가 유용하게 사용할 수 있는 연산자로 XML 질의에서 많이 나타난다. 예를 들어, 다음과 같은 질의를 처리한다고 하자.

```
select u
from  person.*.url u
```

이 질의는 주어진 DB에서 person에서 시작하여 임의의 경로에 의하여 도달되는 url을 구하는 질의이다. [10]에서는 재귀(recursion)을 이용한 최소 고정점 질의(least fix-point query)로 변환하는 기법을 제안하였지만 질의 변환 과정을 사용자의 몫으로 남겨두었고 변환된 질의도 매우 복잡하다.

반면에 제안하는 기법은 DTD 그래프를 이용하여 '*'에 매핑될 수 있는 모든 경우를 찾은 후 이것을 이용한다. 이때 사용하는 DTD 그래프는 연산자를 뺀 형태이다. 먼저, '*'에 매핑되는 경로를 찾는 알고리즘이 알고리즘 4와 같다. '*' 연산을 포함한 질의는 '*' 앞부분의 단순 정규 경로식 R_1과 위의 예제 질의에서는 $R_1 = person$, $R_2 = url$이 된다. 따라서 알고리즘 4은 입력 질의 $R_1.*.R_2$에 대하여 DTD 그래프 G_D로부터 '*'에 매핑되는 가능한 경로를 찾는다.

즉, $R_1 = \phi$일때는 DTD의 각 엘리먼트로부터 모든 가능한 경로를 찾고, $R_1 \neq \phi$인 경우에는 R_1의 마지막 레이블로부터 R_2의 첫째 레이블 사이의 모든 가능한 경로를 찾는다. G_D가 사이클을 가지면 알고리즘은 모든 가능한 최단 경로를 찾는다.알고리즘 4는 15 행에서 방문되었어도(mark) 현재 경로와 사이클을 이루지 않는 경우에 그 노드를 탐색하는데 이것은 DTD 그래프 상에 다이아몬드 계층 구조가 있을 경우를 위해서이다.

알고리즘 4에 의하여 위의 질의의 '*' 연산은 school, company, vehicle.company의 경로로 치환될 수 있다. 따라서 다음의 lorel-like 질의로 변환된다.

```
select u
from  (person.school.url|person.company.url|person.vehicle.company.url) u
```

이 질의는 다음의 OODB의 질의로 바뀐다. 여기서 적당한 부클래스로 변환되는 과정은 생략되었다.

```
select u
from  p in Person,s in p.school,c in p.company,v in p.vehicle
         v2 in v.company, u in (s.url,c.url,v2.url)
```

5 결론

본 논문에서는 객체 지향 데이터베이스가 XML 데이터를 저장하고, 질의를 수행하는 또 하나의 해법이 될 수 있음을 보였다. 우리는 XML 문서의 구조를 나타내는 DTD로부터 OODB의 스키마를 생성하고 XML 질의를 OODB의 질의로 바꾸어서 처리하는 기법을 제안하였다. 특히, 제안하는 방법은 XML 데이터가 기존 DBMS의 스키마와는 달리 고정적인 애트리뷰트를 가지지 않기 때문에 생기는 DTD로부터 OODB 클래스 매핑 시에 생기는 문제를 OO의 계승을 이용하여 해결하였다. 즉, XML 데이터의 비정규적인 부분을 클래스의 계승을 이용하여 표현함으로써 특정 애트리뷰트의 값이 없어서 생기는 Null 값 문제를 해결하고 질의 처리 시에도 유용하게 쓰인다.

알고리즘 4 '$*$' 연산 제거

```
1: 입력: DTD 그래프 G_D, 질의 R = R_1.*.R_2
2: 출력: '*' 연산이 없는 질의
3: Procedure Star-Flattening(G_D, R)
4: if R_1 = φ then
5:    for DTD상의 각 요소(element) e do
6:       Star-Flattening(G_D,e.*.R_2)
7:    end for
8: else
9:    head ← R_1의 마지막 레이블;
10:   tail ← R_2의 첫째 레이블;
11:   큐 Q를 비어있는 상태로 초기화;
12:   head에 방문 표시 하고 Q에 삽입;
13:   while Q가 비어있지 않음 do
14:      x ← GetFront(Q);
15:      for x에 인접한 방문 표시 안된 노드와 방문 표시된 노드 중 사이클을 만들지 않는 노드 w
         do
16:         if w = tail then
17:            head 에서 tail까지의 경로를 출력;
18:            w에 방문 표시;
19:         else
20:            w를 방문 표시하고 Q에 삽입;
21:         end if
22:      end for
23:   end while
24: end if
```

XML 질의 처리는 단순 정규 경로식에 대하여 XML 질의가 OODB의 질의로 바뀌고 다시 질의의 결과가 XML 질의로 변환될 수 있음을 보였는데 임의의 경로가 매핑될 수 있음을 나타내는 '$*$'와 같은 연산은 DTD로부터 가능한 경로를 실제로 찾은 후 질의를 처리하는 기법을 제안하였다.

참조 서적

[1] T. Bray, J. Paoli, and C. Sperberg-McQueen. Extensible markup language (XML) 1.0. Technical report, W3C Recommendation, 1998.

[2] V. Christophides, S. Abiteboul, S. Cluet and M. Scholl. From Structured Documents to Novel Query Facilities. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1994.

[3] J. Bosak, T. Bray, D. Connolly, E. Maler, G. Nicol, C. M. Sperberg-McQueen, L. Wood, and J. Clark. W3C XML Specification DTD. Technical report, W3C Recommendation, 1998.

[4] Peter Buneman. Semistructured data. In *Proceedings of ACM Symposium on Principles of Database Systems*, 1997.

[5] Serge Abiteboul. Querying semi-structured data. In *Proceedings of the International Conference on Database Theory*, 1997.

[6] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database management system for semistructured data. *SIGMOD Record*, 1997.

[7] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1996.

[8] Alin Deutsch, Mari Fernandez, and Dan Suciu. Storing semistructed data with STORED. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1999.

[9] Daniela Florescu and Donald Kossmann. Storing and querying XML data using an RDBMS. *IEEE Data Engineering Bulletin*, 1999.

[10] Jayavel Shanmugasundaram, H. Gang, Kristin Tufte, Chun Zhang, David DeWitt, and Jeffrey F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of the Conference on Very Large Data Bases*, 1999.

[11] Dan Suciu, Mary Fernandez, Susan Davidson, and Peter Buneman. Adding structure to unstructured data . In *Proceedings of the International Conference on Database Theory*, 1997.

[12] Mary Fernandez and Dan Suciu. Optimizing regular path expressions using graph schemas. In *IEEE International Conference on Data Engineering*, 1998.

[13] Roy Goldman and Jennifer Widom. DataGuides: enabling query formulation and optimization in semistructured databases. In *Proceedings of the Conference on Very Large Data Bases*, 1997.

[14] Svetlozar Nestorov, Jeffrey Ullman, Janet Wiener, and Sudarshan Chawathe. Representative objects: concise representations of semistructured, hierarchical data. In *IEEE International Conference on Data Engineering*, 1997.

[15] Tova Milo and Dan Suciu. Index structures for path expressions. In *Proceedings of the International Conference on Database Theory*, 1999.

[16] John E. Hopcroft, Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley Publishing Company, 1979.

[17] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Query language for XML. In *Proceedings of Eighth International World Wide Web Conference*, 1999.

[18] S. Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, Janet Wiener. The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1996.

[19] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1992.

[20] R.G.G. Cattell. *The object database standard: ODMG-93.* Morgan Kaufmann Publishers, 1994.

[21] J. McHugh and J. Widom. Compile-Time Path Expansion in Lore. In *Proceedings the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, 1999.