XML 데이타베이스 시스템을 이용한 RDF 데이타의 저장 및 검색

(A Storage and Retrieval of RDF Data using an XML Database System)

서 명 희 † 정 진 완 †† 민 준 기 † 안 재 용 †
(Myoung-Hee Seo) (Chin-Wan Chung) (Jun-Ki Min) (Jae-Yong Ahn)

요 약 최근 차세대 웹으로 시멘틱 웹이 부각되고 있다. 시멘틱 웹상에서는 정보 리소스들이 서로 의미적으로 연결되어, 이를 컴퓨터가 처리할 수 있다. Resource Description Framework (RDF)는 이런 의미적 연결성을 제공한다. RDF는 웹 리소스들의 메타 데이타를 표현하기 위한 데이타 모델이다. 시멘틱 웹이 발전하기 위해서는 RDF 데이타를 효율적으로 관리하기 위한 방법이 가장 중요하다 할 수 있다. 본 논문에서는 RDF 데이타를 XML 데이타베이스 시스템에 저장하고 이를 검색하는 기법을 제안한다. XML 데이타베이스 시스템을 사용함으로써 XML 데이타와 RDF 데이타를 통합적이고 효율적으로 관리할 수 있다. 또한, 효율적인 검색 방법과 성능을 향상시킬 수 있는 방법들을 제안하고 있다. 논문에서 제안한 질의 처리 기법은 기존의 연구보다 나은 성능을 보여준다.

키워드 : 시멘틱 웹, 데이타베이스

Abstract The Semantic Web is proposed as the next generation Web technology. In the environment of the Semantic Web, resources are related with each other semantically and computers can process this information easily. The Resource Description Framework (RDF) supports this semantic relationship. RDF is the data model for describing metadata of the Web resources. To establish and develop the Semantic Web, methods for managing RDF data efficiently are the most important. So, in this research, we propose methods for storing and querying RDF data using an XML database system. Using an XML database system, XML data, main data of the Semantic Web, and RDF data, the metadata of XML data, can be managed in the same storage and by the same mechanism efficiently. In addition, we propose an efficient data retrieval method and several techniques to improve the system performance. Our query processing technique performs better than an existing system.

Key words : RDF, XML, Database, RQL

1. 서 론

최근 차세대 웹으로 시멘틱 웹(Semantic Web)이 부각되고 있다. 기존의 월드 와이드 웹과는 달리 시멘틱 웹상에서는 정보 리소스들의 의미가 정의되어 있고, 이들간의 의미적 연결을 지원한다. 시멘틱 웹에서 이런 의미적 연결성을 지원하기 위해 Resource Description Framework(RDF)[1]를 사용한다. RDF는 웹 리소스들의 메타 데이타를 표현하기 위한 데이타 모델이다. RDF를 이용해서 정의된 데이타를 시멘틱 웹상에서 자유롭게 접근할 수 있고, 이런 정보를 컴퓨터가 처리하여 원하는 정보를 얻을 수 있다.

시멘틱 웹이 차세대 웹으로 자리 잡기 위해서는 RDF 데이타를 잘 다루기 위한 기술들이 선행 연구되어야 한다. 특히, 방대한 양의 RDF 데이타를 효율적으로 저장하고 검색하는 기법이 중요하다고 할 수 있다. RDF 데이타를 저장하고 검색하기 위한 연구로 RDF 데이타를 객체 지향형 데이타베이스 시스템(ORDBMS)

· 본 연구는 정보통신부의 대학 IT연구센터(ITRC) 지원을 받아 수행되었습니다.

† 비 회 원 : 한국과학기술원 전산학과
mhseo@islab.kaist.ac.kr
jkmin@islab.kaist.ac.kr
jyahn@islab.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전산학과 교수
chungcw@cs.kaist.ac.kr

논문접수 : 2003년 8월 12일
심사완료 : 2003년 12월 26일

에 저장하고 검색하는 연구가 있다[3,4]. 하지만, RDF 데이타 모델 및 표현 방식과 객체 관계형 데이타베이스 시스템의 데이타 모델이 상이하기 때문에, 저장 및 검색 작업이 비효율적이다. 본 연구에서는 RDF 데이타를 XML 데이타베이스 시스템에 저장하고, RDF 질의 언어인 RQL을 이용하여 검색하는 방법을 제안하고 있다. 기존의 연구와 달리, RDF 데이타를 XML 데이타베이스 시스템에 저장함으로써 XML 데이타와 이의 메타 데이타인 RDF 데이타를 같이 저장할 수 있고, 따라서 이들을 통합적이고 효율적으로 관리할 수 있는 방법을 제공한다. 또한 RDF 데이타를 XML을 사용하여 표현할 수 있어, XML 데이타베이스 시스템에 RDF 데이타를 별도의 처리 과정 없이 XML 형태 그대로 저장할 수 있다.

또한, 본 연구에서는 RDF 데이타에 대한 RQL 질의를 XML 질의 언어인 XPath[8] 질의로 변환하여 처리하는 효율적인 방법을 제안하고 있다. RDF 데이타 모델과 XML 데이타 모델이 다르기 때문에 RDF 데이타를 XPath 질의 언어로 검색하기에 비효율적이다. 따라서, 본 논문에서는 RDF 질의 언어인 RQL을 XPath 질의로 변환하여 XML 데이타베이스 시스템에서 검색할 수 있는 방법을 제공한다. 또한, 질의 처리 과정에서 검색 성능을 향상시킬 수 있는 몇 가지 방법들을 사용하였다.

본 논문의 구성은 다음과 같다. 2장 관련 연구에서는 기존에 연구된 RDF 데이타 저장 및 검색 시스템에 대해서 살펴본다. 3장에서는 RDF 데이타를 데이타베이스에 저장하는 기법을 설명하고, 4장에서는 저장된 RDF 데이타 검색 기법에 대해 설명한다. 5장에서는 간단한 실험에 대한 결과에 대해 논의하고, 끝으로 6장에서는 결론을 맺는다.

2. 관련 연구

2.1 RDF 데이타 저장 및 검색 시스템

현재 세계 여러 곳에서 시멘틱 웹 연구의 일환으로 RDF 데이타 저장 및 검색에 대한 연구들이 진행 중에 있으나, 아직까지는 미비한 실정이다. 진행중인 연구 중에 RQL 질의 언어를 지원하는 것으로 ICS-FORTH의 RDFSuite[3]와 On-to-Knowledge 프로젝트의 일부인 Sesame[4] 시스템 등이 있다. 이 두 시스템 모두 RDF 데이타를 객체 관계형 데이타베이스 시스템에 저장하고 RQL 질의가 들어오면 이를 SQL3로 변환하여 결과를 가져온다. 또한, 저장할 때, RDF Schema 정보를 이용하여 데이타베이스 스키마를 구성하고 스키마에 맞게 RDF 데이타를 조작하여 저장한다. 특히, Sesame 시스템은 객체 관계형 데이타베이스 시스템에 국한되지 않고, 데이타베이스 시스템에 독립적으로 구성되어 있다. 하지만, 이 경우 질의 처리 작업의 대부분이 질의 처리 시스템에서 이루어지기 때문에 데이타베이스 시스템의 최적화된 질의 처리기를 사용하는 다른 시스템에 비해 그만큼 성능이 나쁠 수 있다. 또한, 이 두 시스템은 기반 데이타베이스 시스템으로 객체 관계형 데이타베이스를 사용하였으므로, 저장할 때 데이타베이스 스키마에 맞게 데이타를 구성하여 저장하여야 하고, 검색할 때에도 여러 테이블 간의 조인 등의 오버헤드가 있다.

본 논문에서 제안하는 시스템은 기반 데이타베이스 시스템으로 XML 데이타베이스 시스템을 사용하여, 저장할 때, 별도의 처리 없이 데이타 그대로 저장할 수 있다. 또한, 웹상의 주된 데이타인 XML 데이타와 이의 메타 데이타인 RDF 데이타를 같은 시스템에 저장하고 관리함으로써 두 데이타 간의 공간적 차이를 줄여, 통합적이고 효율적인 관리가 가능하다. 그리고, Sesame 시스템과는 달리 RDF 데이타에 대한 질의 처리를 가능한 한 데이타베이스 시스템에서 수행하도록 하였고, 질의 처리 시 성능을 향상시킬 수 있는 방법들을 적용하여 검색의 효율을 높였다.

3. RDF/RDF Schema 데이타 저장 기법

3.1 RDF 데이타 저장 과정

RDF 데이타 저장 과정은 그림 1과 같다. RDF description 데이타와 RDF schema 데이타가 들어오면 이를 Validator에서 구문과 의미에 대한 검증을 수행한다. 그 다음으로, Standardizer에서 여러 형태의 RDF/XML 구문을 하나의 구문으로 통합하는 작업을 수행한다. 이 때, 저장 공간의 효율성과 검색의 용이성을 고려한 하나의 RDF/XML 형태로 통일한다. 통일된 형태의 standard RDF/XML 데이타가 XML 데이타베이스 시스템에 저장된다. 이때, class/property hierarchy extractor에서 RDF schema 데이타의 클래스와 속성의 계층 구조 및 속성의 domain, range 제약에 대한 정보를 뽑아서 별도의 XML 문서로 저장한다. 이렇게 별도로 저장된 정보들은 검색의 성능을 향상시키기 위해 사용된다.

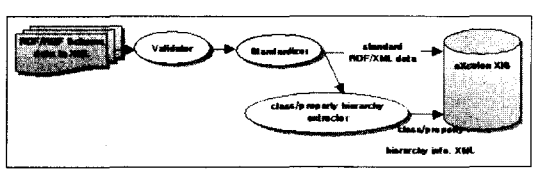

그림 1 RDF 데이타 저장 과정

3.2 통일된 RDF/XML 형태

RDF description 데이타 및 RDF schema 데이타는 XML로 표현될 수 있다. 즉, RDF/XML 구문[9]을 이용하여 표현할 수 있다. 따라서, XML 데이타베이스 시스템에 별도의 데이타 처리과정 없이 저장할 수 있다. 하지만, 하나의 RDF 그래프 데이타가 여러 형태의 XML 데이타로 변환될 수 있고, XML 질의 언어인 XPath 질의는 XML 구문에 의존적이다. 따라서, 입력된 RDF/XML 데이타의 형태가 어떻든지 간에 RQL 질의를 일정한 형태의 XPath 질의로 변환하기 위해서는 입력된 RDF/XML 데이타를 통일된 형태의 RDF/XML 데이타로 변환할 필요가 있다. 이 작업은 검증작업과 병행되어 수행되어 큰 오버헤드 없이 수행될 수 있다.

본 연구에서는 검색의 용이성과 저장의 효율성을 위해 RDF/XML 데이타의 여러 단축(abbreviation) 구문[9] 중에서 두 가지 구문만을 적용한 형태로 모든 RDF 데이타를 통일하였다. 즉, 원래의 RDF/XML 구문보다 단축된 형태로 훨씬 적은 저장 공간이 필요하도록 하였고, 리소스에 대한 속성과 속성값을 하나의 형태로 표현하여, 검색할 때 고정된 형태의 XPath 질의로 변환 가능하도록 하였다.

3.3 데이타 구체화

이 절에서는 검색에 자주 사용되고, 결과 도출이 어려운 RDF schema 클래스와 속성 데이타를 미리 뽑아 저장하는 과정에 대해서 알아본다. 이 과정을 데이타 구체화(materialization)라 한다. RQL은 RDF schema 클래스 및 속성의 트랜지티브 클로저(transitive closure)를 지원한다. 또한, 속성의 domain, range 제약에 대한 질의도 지원한다. 이런 질의들을 RDF/XML 형태의 RDF schema 데이타에서 검색하는 작업은 간단하지가 않다. 그림 2는 클래스들의 계층 구조만을 명시하고 있는 RDF schema를 RDF/XML 형태로 표현한 것이다. 이 RDF schema 데이타에 대해 MotorVehicle의 모든 트랜지티브 하위 클래스를 구하라는 RQL 질의가 들어온 경우, 이를 XPath 질의로 변환하면 다음과 같다.

```
R={/rdf:Description[/rdfs:subClassOf/@rdfs:resource
="MotorVehicle"]/@rdf:ID}
∃r∈R, /rdf:Description[rdf:ID =r]/rdfs:subClassOf/
@rdfs:resource
```

즉, MotorVehicle의 하위 클래스들을 구하고, 각각의 하위 클래스들의 하위 클래스를 검색하는 재귀적 과정을 통해 결과를 가져올 수 있다. 그러므로, 클래스의 계층 관계의 깊이가 깊어질수록 더 많은 XPath 질의가 필요하게 된다. 속성의 계층 관계를 구하는 경우도 마찬가지이다. 따라서, 클래스 및 속성의 계층관계 데이타를 미리 추출하여 관리하면, 이런 종류의 데이타를 요하는 질의의 검색 속도를 향상시킬 수 있다. 클래스, 속성의 계층관계에 대한 질의 외에, RQL 질의 언어로 검색 가능한 데이타로 속성의 *domain, range* 제약 사항이 있다. RDF schema 데이타에서 이 정보를 검색하는 작업은 그리 어렵지는 않지만, RDF schema 데이타의 양이 많은 경우 domain, range 속성 데이타만을 미리 뽑아두면 이들을 검색할 때, 검색 속도를 향상시킬 수 있다.

```
<rdf:Description rdf:ID="MotorVehicle">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
   schema#Class"/>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-
   schema#Resource"/>
</rdf:Description>
<rdf:Description rdf:ID="PassengerVehicle">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
   schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
<rdf:Description rdf:ID="Van">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
   schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
<rdf:Description rdf:ID="MiniVan">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
   schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Van"/>
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdf:Description>
```

그림 2 RDF/XML 형태의 RDF schema 데이타의 예

그림 3은 그림 2의 RDF schema에서 클래스 계층 구조 정보를 뽑아 이를 XML의 부모 엘레먼트와 자식 엘레먼트의 관계로 표현한 것이다. RDF 데이타 모델은 노드와 간선에 모두 레이블이 있는 그래프 형태이고, XML 데이타 모델은 노드에만 레이블이 있는 트리 형태이다. 따라서, RDF schema 데이타 중에서 클래스 및 속성의 subClassOf 속성으로 연결된 데이타만을 뽑아 간선의 레이블을 없애고, 다중 상속 받는 클래스를 나누어 상위 클래스에 대한 하위 클래스로 각각 속하도

```
/eg:MotorVehicle//*
```

```
<eg:MotorVehicle>
  <eg:PassengerVehicle>
  <eg:MiniVan/>
 </eg:PassengerVehicle>
  <eg:Van>
   <eg:MiniVan/>
  </eg:Van>
</eg:MotorVehicle>
```

그림 3 그림 2의 RDF schema 클래스 계층 구조에 대한 XML 데이타

록 하였다. 이렇게 XML 데이타로 표현하여 저장함으로써, 계층 구조를 검색하는 RQL 질의를 간단한 하나의 XPath 질의로 변환하여 결과를 가져올 수 있다. 앞서 살펴본 MotorVehicle의 모든 트랜지티브 하위 클래스들을 검색하는 RQL 질의는 다음과 같은 XPath 질의로 변환할 수 있다.

속성에 대한 계층 구조도 유사한 방법으로 XML 데이타로 표현되어 저장된다. 또한, 속성에 대한 domain, range 제약 사항에 대한 데이타도 같이 저장하는데, 이 데이타들은 해당 속성을 표현하는 XML 엘레먼트의 domain, range 어트리뷰트로 표현될 수 있다. 이렇게 RDF schema 데이타로부터 추출되어 구체화된 데이타를 이용해서 해당 데이타를 검색하는 RQL 질의를 보다 효율적으로 처리할 수 있다.

4. RDF/RDF Schema 데이타 검색 기법

4.1 질의 처리 과정

RQL 질의를 처리하는 과정은 그림 4와 같다. 사용자로부터 RQL 질의가 들어오면, 이를 Parser에서 파싱한 후 질의의 분류에 따라 해당 질의 처리기가 RQL 질의를 XPath 질의로 변환하여 XML 데이타베이스 시스템인 eXcelon XIS 시스템에 보내어 결과를 가져온 후, 결과 생성기가 결과를 테이블 형태로 변환하여 사용자에게 보내주게 된다. 질의 처리 시 RDF 데이타를 저장할 때 뽑아둔 RDF schema 클래스와 속성의 계층 정보를 사용하여 질의를 효율적으로 처리한다.

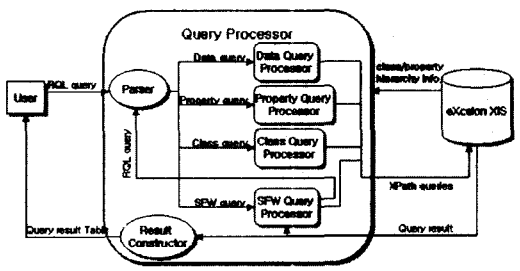

그림 4 RQL 질의 처리 과정

4.2 RQL 질의의 분류

RQL 질의는 형태상 크게 두 가지 - Select-From-Where(SFW) Query, Non SFW Query - 로 분류할 수 있다. SFW query는 select-from-where 구문으로 이루어진 질의이다. Non SFW Query를 질의 결과 타입과 질의의 형태에 따라 다시 분류해보면 세가지로 나눌 수 있는데, Class query, Property query, Data query로 분류할 수 있다. SFW 질의의 결과 타입은 RDF description 데이타, RDF schema 클래스 및 속성 어느 것이든 될 수 있다. 표 1은 RQL 질의의 분류를 정리한 것이다.

표 1 RQL 질의의 분류

Non-SFW Query	Class Query	TypeOf
		SuperClassOf[^]
		SubClassOf[^]
		Domain
		Range
		URI
	Property Query	SuperPropertyOf[^]
		SubPropertyOf[^]
		URI
	Data Query	InstanceOf
		URI
SFW Query		

4.3 Non Select-From-Where(Non SFW) Query 변환 기법

표 2는 Non-SFW query의 분류에 대해 각각에 해당되는 RQL 질의의 예와 각각을 XPath 질의로 변환한 예를 보여주고 있다. 표에서 각각의 질의에 대해 위 부분은 질의의 예이고, 아래 부분은 이를 XPath 질의로 변환한 형태를 보여 준다. 변환된 질의에서 rdf_data.xml은 RDF 데이타가 저장되어 있는 XML 문서의 이름이고, classHierarchy.xml은 데이타 저장 시 구체화된 클래스 계층 구조 정보가 저장되어 있는 XML 문서

표 2 Non-SFW Query의 XPath로의 변환

Class Query	TypeOf	document("rdf_data.xml")//rdf:Description[@rdf:about="www.culture.net#picasso132"]/rdf:type/@rdf:resource
	Super-ClassOf	document("classHierarchy.xml")//NS:Painter/ancestor::*
	Sub-ClassOf	document("classHierarchy.xml")//NS:Artist/descendant::*
	Domain	document("propertyHierarchy.xml")//NS:creates/@rdf:domain
	Range	document("propertyHierarchy.xml")//NS:creates/@rdf:range
	URI	document("classHierarchy.xml")//*
Property Query	Super-PropertyOf	document("propertyHierarchy.xml")//NS:sculpts/ancestor::*
	Sub-PropertyOf	document("propertyHierarchy.xml")//NS:creates/descendant::*
	URI	document("propertyHierarchy.xml")//*
Data Query	Instance Of	document("rdf_data.xml")//rdf:Description[rdf:type/@rdf:resource="http://www.icom.com/schema1.rdf#Painter"]/@rdf:about
	URI	∃c∈SubClassOf(http://www.icom.com/schema1.rdf#Artist), document("rdf_data.xml")//rdf:Description[rdf:type/@rdf:resource=c]/@rdf:about

의 이름이다. 마찬가지로 propertyHierarchy.xml은 속성 계층 구조 정보가 저장되어 있는 XML 문서의 이름이다. 클래스 계층 관계를 검색하는 SuperClassOf, SubClassOf, 그리고 속성의 계층 관계를 검색하는 SuperPropertyOf, SubPropertyOf 등의 질의와 속성의 Domain, Range 클래스를 검색하는 질의, 그리고 모든 클래스와 모든 속성을 검색하는 URI 질의 등은 3.3절에서 설명한 데이타 저장 시 미리 추출된 클래스 계층 구조 정보와 속성 계층 구조 정보 및 domain, range 속성 정보가 저장되어 있는 문서를 이용하면 간단한 XPath 질의로 변환될 수 있다. 클래스의 트랜지티브 하위 클래스들을 검색하는 SubClassOf() 질의를 예를 들면, 표 2에서 SubClassOf(http://www.icom.com/ schemal.rdf#Artist) 질의는 Artist 클래스의 트랜지티브 하위 클래스들을 검색하기 위해 클래스 계층 정보가 저장되어 있는 classHierarchy.xml 문서에서 Artist 엘레먼트의 자손(descendant) 엘레먼트들을 검색하는 XPath 질의로 변환된다. 그 외의 질의들은 RDF description 데이타에 대한 간단한 XPath 질의로 변환된다.

4.4 Select-From-Where(SFW) Query 처리 기법

4.4.1 Select-From-Where (SFW) Query의 특징

Select-From-Where(SFW) Query를 통해 RDF description 데이타나 RDF schema 데이타 어느 것이나 검색할 수 있다. SFW Query는 RDF 데이타 모델을 N-Triples[11] 형태로 보고 검색할 수 있는 방법을 제공한다. 또한, from절에 경로 표현식(path expression)을 사용하여 RDF 데이타 모델의 경로에 대한 검색을 할 수 있다. 그림 5는 RQL SFW 질의의 예이다.

```
select X, $W, Y, Z , $V, U
from {X:$W} creates{Y}.exhibited{Z:$V},
   {X}fname{U}
wherenot$W <= Painteror Z like louvre)
```

그림 5 RQL SFW Query의 예

그림 5의 질의는 RDF description 데이타와 RDF schema 데이타를 모두 접근해서 결과를 가져오는 질의이다. From 절의 경로 표현식에서 X, Y등의 변수는 RDF description 데이타 변수이고, '$'가 붙은 $W 등의 변수는 RDF schema 클래스 변수이다. From 절의 첫번째 경로 표현식인 {X:$W}creates{Y}는 속성이 creates인 트리플(triple)의 서브젝트 리소스 X와 그의 클래스 $W, 그리고 오브젝트 Y를 의미한다. 또한, 두 개의 경로를 '.'으로 연결하여 암시적으로 조인을 표현하고 있다. Where절에는 변수에 대한 조건을 명시함으로써 만족하는 데이타만을 검색할 수가 있다.

4.4.2 SFW Query 처리 알고리즘

SFW Query를 처리하는 알고리즘은 그림 6과 같다.

```
function process_SFWQuery(query q)
begin
   //select part, from part, 그리고 where part로 나눈다
                    :
   pathExprs := separateFromPart(fromPart)
   conjunctiveForm := getConjunctiveForm(wherePart)
   atomicConds := getAtomicConditions(conjunctiveForm)
   for every path_expr e which belongs to pathExprs do
          pathQueryResults.add(getPathQueryResults(e,
          atomicConds))

   return joinResults(stack(pathQueryResults))
end

function getPathQueryResults(pathExpr e, atomicConds a)
begin
   for every var v in e do
      xpathQuery := 표4.3에서 v에 대응되는 xpath query
      if some condition c exist in a for v then
         condString = getCondString(v, c)
         if condString is not null then
            xpathQuery := xpathQuery + condString
      results.add(getResults(xpathQuery))

   if some condition c' exist in a for e and not processed then
      results := checkCondition(results, c')

   return results
end

function getCondString(var v, condition c)
begin
   if operator o in c is like operator then
      condString := contains(node, o.arg)
   else if operator o is =, !=, not operator then
      condString := node o o.arg

   return condString
end

function checkCondition(results r, condition c)
begin
   if operator o in c is or operator then
      checkCondition(r, o.arg1) U checkCondition(r, o.arg2)
   else if o is and operator then
      checkCondition(r, o.arg1) ^ checkCondition(r, o.arg2)
   else if o is not operator then
      r - checkCondition(r, o.arg)
   else if o is (</<=/>/>=) operator then
      if var v in c is schema var then
         for every result r' in r do
            if r' is subSchema/superSchema of o.arg then
                  results.add(r')
      else if var v in c is data var then
         for every result r' in r do
            if r' (<|<=|>|>=) o.arg then
                  results.add(r')

   return results
end

function joinResults(pathQueryResultsStack r)
begin
   r1 := r.pop()
   if r is empty then return r1
   r2 := r.pop()

   if r1 and r2 have same variable then
      joinResult := sortMergeJoin(r1, r2)
   else joinResult := product(r1, r2)

   r.push(joinResult)

   return joinResults(r)
end
```

그림 6 SFW Query 처리 알고리즘

전체적인 흐름은 process_SFWQuery 함수를 통해 알 수 있다. 처리 과정을 간단하게 살펴보면, 우선 SFW query를 select, from, where부분으로 나눈다. 그 후에 from절의 경로 표현식을 단순 경로식들로 분리한다. 그 다음으로 where절에 있는 조건(condition)을 and로 연결된 형태(conjunctive form)로 바꾼 후, 이를 and를 기준으로 분리한다. 이를 애토믹 컨디션(atomic condition)이라 표현하고 있다. 그 다음 작업으로는 from절의 각각의 경로식을 XPath 질의로 변경한다. 이때, where절의 조건 중에서 같이 처리할 수 있는 것들은 XPath 질의의 프리디킷(predicate)으로 넣어, XPath 질의에 대한 결과를 가져올 때, 조건에 맞는 데이타만을 가져오게 한다. 조건 중에서 같이 처리할 수 없는 것들은 해당 경로식의 결과를 가져온 후, 조건에 맞는 데이타만을 가려낸다. 마지막 작업으로 각각의 경로식에 대한 결과값들을 조인한다. 각각의 단계별 자세한 처리 과정은 다음절에서부터 다룬다.

4.4.3 복잡한 경로식 처리

그림 5의 SFW query에서 from절의 첫번째 경로 표현식은 두 개의 경로를 '.'으로 연결하여 암시적 조인을 나타내고 있다. 이런 경로 표현식을 처리하기 위해 우선 하나의 경로로 이루어진 경로 표현식으로 변환한다. 그림 5의 from절의 경로 표현식들을 단순 경로 표현식으로 바꾸면 다음과 같다.

```
{X:$W} creates {Y}
{Y} exhibited {Z:$V}
{X} fname {U}
```

4.4.4 조건 분리

Where절의 조건들의 연산을 데이타베이스 시스템으로 내리기(push down) 위해 조건을 우선 and로 연결된 형태(conjunctive form)로 변환한다. 단, not으로 시작하지 않는 형태로 가정한다. 이는 가장 외곽의 연산자가 and가 되어 이를 기준으로 나누기 위해서 이다. 조건을 and로 연결된 형태로 바꾸는 방법은 다음의 몇 가지 법칙을 적절히 적용하여 사용할 수 있다.

```
(A and (B or C)) = ((A and B) or (A and C))
(A or (B and C)) = ((A or B) and (A or C))
not(not A) = A
not(A and B) = not(A) or not(B)
not(A or B) = not(A) and not(B)
```

그림 5의 예에서 where절의 조건을 아래의 마지막 법칙을 적용하면 다음과 같이 변환된다.

not($W <= Painter) and not(Z like "louvre")

이를 and를 기준으로 나누면 $W > Painter, not(Z like "louver") 두개의 애토믹 컨디션을 구할 수 있다.

4.4.5 경로 표현식의 분류와 처리

표 3은 본 연구에서 다루고 있는 단순 경로 표현식의 분류와 각각에 해당되는 XPath 질의를 보여준다. 단순 경로 표현식은 RDF description 데이타의 경로를 표현하는 Data path와 RDF schema 데이타의 경로를 표현하는 Schema path, 그리고 두 가지 경로 모두를 표현하는 Mixed path가 있다. 경로 표현식에서 *c*는 질의에서 주어진 RDF schema 데이타의 어느 한 클래스이고, *p*는 RDF schema 데이타의 어느 한 속성을 의미한다. 또한, *Class*는 RDF Schema에 정의되어 있는 Class 클래스를 의미하고, *Property*는 RDF Schema에 정의되어 있는 Property 클래스를 의미한다. 변환된 XPath 질의에서 *C*는 RDF schema 데이타의 모든 클래스들의 집합이고, *P*는 모든 속성들의 집합니다. 변환된 XPath 질의에서 나타나고 있는 SubClassOf, SubPropertyOf, Domain, Range등은 앞 절에서 설명한 방식으로 얻을 수 있는 값들이므로 여기서는 자세한 설명은 생략한다.

Data path에는 클래스와 그의 인스턴스를 나타내는 *c*{X}, $X{Y} 경로 표현식과 RDF description 데이타의 트리플을 나타내는 {X}*p*{Y}, {X}@P{Y}등의 경로 표현식이 있다. c는 주어진 클래스이고, $X는 RDF schema의 임의의 클래스를 의미한다. 마찬가지로 p는 주어진 속성이고, @P는 RDF schema 임의의 속성을 의미한다. 클래스의 인스턴스를 나타내는 경로를 XPath 질의로 변환하면 그 클래스를 type으로 갖는 모든 리소스들을 검색하는 질의로 변환할 수 있다. 트리플을 표현하는 경로를 XPath 질의로 변환하면 속성을 갖는 리소스와 속성값을 구하는 XPath질의로 변환할 수 있다.

Schema path에는 RDF schema의 모든 클래스를 구하는 *Class*{X}와 모든 속성을 구하는 *Property*{P}, {}@P{}등의 경로 표현식이 있고, 클래스의 트랜지티브 서브 클래스를 구하는 *c*{:$C}, $X{:$Y}와 schema의 트리플을 표현하는 {:$X}*p*{:$Y}, {:$X}@P{:$Y} 등의 경로 표현식이 있다. Schema path들은 모두 앞 절에서 설명한 클래스, 속성의 계층 구조 정보를 이용하여 쉽게 구할 수 있다. 특히 RDF schema 트리플을 표현하는 경로식의 경우, 속성의 domain, range 제약 사항 정보를 이용하여 쉽게 구할 수 있다.

Mixed path에는 클래스의 인스턴스와 트랜지티브 서브 클래스를 표현하는 *c*{X:$C}와 RDF description 데이타의 트리플, 그리고 RDF schema의 트리플을 표현하는 {X:$Z}*p*{Y:$W}, {X:$Z}@P{Y:$W} 등이 있다. 이들 경로 표현식은 앞서 설명한 data path와 schema path를 XPath로 변환한 방식을 모두 사용하여 구할 수 있다.

표 3 RQL SFW Query의 경로 표현식의 분류 및 해당 XPath 질의

경로 유형	경로 표현식	변환된 XPath 질의
Data path	c{X}	∃c'∈SubClassOf(c), <//rdf:Description[type/@resource= c']/@about>
	$X{Y}	∃c∈C, <c, //rdf:Description[rdf:type/@rdf:resource = c]/@rdf:about>
	{X}p{Y}	∃p'∈SubPropertyOf(p), V1={//rdf:Description[p']/@rdf:about}, ∃v1∈V1,<v1, //rdf:Description[@rdf:about=v1]/p'/@rdf:resource>
	{X}@P{Y}	∃p∈P, V1={//rdf:Description[p]/@rdf:about}, ∃v1∈V1, <v1, p, //rdf:Description[@rdf:about=v1]/p/@rdf:resource>
Schema path	Class{X}	∃c∈C, < c >
	Property{P}	∃p∈P, < p >
	{}@P{}	∃p∈P, < p >
	c{:$C}	∃c'∈SubClassOf(c), < c' >
	$X{:$Y}	∃c∈C, < c, SubClassOf(c) >
	{:$X}p{:$Y}	∃c1∈SubClassOf(Domain(p)), ∃c2∈SubClassOf(Range(p)), <c1, c2>
	{:$X}@P{:$Y}	∃p∈P, ∃c1∈SubClassOf(Domain(p)), ∃c2∈SubClassOf(Range(p)), < c1, p, c2 >
Mixed path	c{X:$C}	∃c'∈SubClassOf(c), <//rdf:Description[rdf:type/@rdf:resource= c']/@rdf:about, c'>
	{X:$Z}p{Y:$W}	∃p'∈SubPropertyOf(p), V1={//rdf:Description[p']/@rdf:about}, ∃v1∈V1, C1={//rdf:Description[@rdf:about=v1]/rdf:type/@rdf:resource}, V2={//rdf:Description[@rdf:about=v1]/p'/@rdf:resource}, ∃v2∈V2, C2={//rdf:Description[@rdf:about=v2]/rdf:type/@rdf:resource}, ∃c1∈C1, c1(SubClassOf(Domain(p')), ∃c2∈C2, c2(SubClassOf(Range(p')), < v1, c1, v2, c2 >
	{X:$Z}@P{Y:$W}	∃p∈P, V1={//rdf:Description[p]/@rdf:about}, ∃v1∈V1, C1={//rdf:Description[@rdf:about=v1]/rdf:type/@rdf:resource}, V2={//rdf:Description[@rdf:about=v1]/p/@rdf:resource}, ∃v2∈V2, C2={//rdf:Description[@rdf:about=v2]/rdf:type/@rdf:resource}, ∃c1∈C1, c1(SubClassOf(Domain(p')), ∃c2∈C2, c2(SubClassOf(Range(p')), < v1, c1, p, v2, c2 >

4.4.6 Where절의 조건 처리 방법

조건은 boolean 연산과 비교(comparative) 연산, 그리고 not 연산으로 구성되어 있다. Boolean 연산에는 and, or가 있고, 비교 연산에는 <, <=, >, >=, =, != 연산자와 문자열 비교를 위한 like 연산자가 있다. Boolean 연산자인 and와 or는 4.4.4에서 설명한 바와 같이 and로 연결된 형태로 바꾼 후, 이를 and를 기준으로 분리하여 atomic 조건들로 만든다. Atomic 조건들 중에서 각각의 경로 표현식을 XPath 질의로 변환할 때, 해당되는 조건들 중 XPath질의의 프리디킷으로 넣을 수 있는 것은 XPath 프리디킷으로 표현한다. 표 4는 조건의 비교 연산과 not연산에 대해 XPath 프리디킷으로 변환가능한지 그리고 가능하다면 어떻게 변환할 수 있는지 보여주고 있다. 비교 연산 중에서 연산자가 <, <=, >, >= 인 조건은 XPath 프리디킷으로 넣을 수 없다. 그러나 비교 연산 중에서 =, != 와 not 연산자를 가지고 있는 조건은 XPath 프리디킷으로 넣을 수 있고, 프리디킷으로 들어갈 때, 원래 연산자 그대로 프리디킷으로 넣을 수 있다. 비교 연산 중에서 like 연산자를 가지고 있는 조건은 contains() 라는 XPath 함수를 써서 XPath 프리디킷으로 넣을 수 있다. 그림 5의 조건 중에서 $W > Painter는 해당 경로 표현식인 {X:$W} creates {Y}를 XPath 질의로 변환할 때, XPath 질의의 프리디킷으로 넣을 수 없으나 조건 not(Z like "louver")는 해당 경로 표현식인 {Y} exhibited {Z:$V}를 XPath 질의로 변환할 때, 다음과 같은 XPath 프리디킷으로 변환하여 넣을 수 있다.

not(**contains**(*exhibited/*@resource, "*louver*"))

표 4 조건 연산의 분류에 따른 XPath 질의의 프리디킷으로의 변환

조건 연산자	<, <=, >, >=	=, !=, not	like
XPath 프리디킷	X	=, !=, not	contains()

경로 표현식 {Y} exhibited {Z:$V}를 조건을 고려하여 XPath 질의로 변환하면 다음과 같다.

```
∃p∈SubPropertyOf(exhibited),
Y={//rdf:Description[p]/@rdf:about}, ∃y∈Y,
Z={//rdf:Description[@rdf:about=y and
    not(contains(exhibited/@resource, "louvre"))]
      /p/@rdf:resource}, ∃z∈Z,
$V={//rdf:Description[@rdf:about=z]/rdf:type/@rdf:resource},
    ∃v∈$V, <y, z, v >
```

표 3의 변환 표에 따른 해당 경로의 XPath 질의와 경로의 변수에 대한 조건의 XPath 프리디킷이 같이 표현된 XPath 질의로 변환됐음을 볼 수 있다. 이렇게 가능한 조건을 XPath 질의의 프리디킷으로 표현하여 조건에 대한 연산을 가능한 데이타베이스 시스템에서 처리하도록 함으로써, 데이타베이스 시스템의 질의 처리기를 최대한 활용할 수 있고, 데이타베이스에서 주고 받는 데이타의 양을 줄일 수 있어 RQL 질의 처리를 효율적으로 수행하도록 하였다.

위의 비교 연산 중에서 <, <=, >, >= 연사자를 가진 조건이 XPath 질의의 프리디킷으로 표현되지 못하는 이유는 조건이 데이타 변수에 대한 조건일 경우, 이 연산자들에 대한 XPath 프리디킷을 eXcelon XIS 시스템에서 지원하고 있지 않기 때문이다. 조건이 스키마 변수에 대한 조건일 경우, 연산자들의 의미가 클래스 또는

속성의 계층 구조에 대한 관계를 의미하기 때문에, 이를 XPath 질의의 프리디킷으로 표현할 수 없다. 이외에 atomic 조건 중에서 and 와 or가 혼합된 형태의 조건도 XPath 프리디킷으로 표현할 수 없어, 해당 경로 표현식의 결과에 대해 조건을 검사한다.

그림 7은 그림 5의 SFW query에 대한 질의 처리 플랜(plan)을 표현하고 있다. 질의 트리의 말단은 경로 표현식으로 이루어져 있고, 중간 노드들은 relational algebra의 연산자들로 이루어져 있다. 트리의 말단에 타원으로 묶여져 있는 부분은 데이타베이스 시스템에서 처리되는 부분이고, 그 외의 부분은 RQL 질의 처리 시스템에서 처리되는 부분이다. 이처럼 가능한 연산을 데이타베이스 시스템에 내려 수행하였고, 그렇지 않은 선택 연산들도 최대한 먼저 수행함으로써, 조인 연산을 위한 데이타의 양을 최대한 줄여 조인 연산이 효율적으로 수행되도록 하였다.

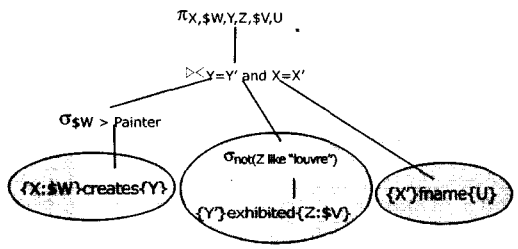

그림 7 그림 5의 SFW query에 대한 질의 처리 플랜(plan)

4.4.7 조인 연산 처리

그림 6에서 joinResults함수가 조인을 처리하는 함수이다. 각각의 경로 표현식의 결과에 대해 스택을 이용하여 순서대로 둘씩 조인을 수행한다. 두개의 결과가 조인이 필요 없는 경우는 두 결과를 프로덕트하고, 조인이 필요한 경우는 공통의 변수에 대해 sort-merge 조인을 수행한다. 본 연구에서 취하고 있는 조인 방법은 각각의 경로 표현식의 결과를 가져온 후, 이를 조인하는 방식이다. 다른 조인 방법으로는 하나의 경로 표현식의 결과를 가져온 후, 각각의 결과에 대해 반복적으로 다른 경로의 결과를 데이타베이스에서 가져오는 방법이 있고, 실제로 Sesame 시스템 등에서 이런 방식을 취하고 있다. 하지만, 이 방법은 한 경로의 결과 사이즈 만큼 데이타베이스에 접근해야 하고, 또한 조인의 횟수가 많아짐에 따라 그만큼 더 데이타베이스 접근 횟수가 많아지는 단점이 있다. 그만큼 조인 연산 수행 속도가 느려지게 된다. 본 연구에서는 앞 절에서 설명하였듯이, 선택 연산을 최대한 아래로 내려 수행함으로써 각각의 경로에 대한 결과가 그다지 크지 않을 것이고, 따라서 앞서 설명한 방법보다 효율적으로 조인을 수행할 수 있다.

5. 실 험

실험은 Pentium III 856MHz CPU, 256MB memory, Windows 2000 operating system, 그리고 XML 데이타베이스 시스템 eXcelon XIS 3.1에서 수행하였다. 실험에서 표 5의 4개의 질의를 사용하였다. 첫번째 질의는 RDF schema 데이타를 검색하는 schema query 중에서 하위 클래스들을 검색하는 질의이고, 두 번째 질의는 RDF description 데이타를 검색하는 data query 중에서 주어진 클래스 URI의 인스턴스를 검색하는 질의이다. 세 번째와 네 번째 질의는 SFW query이다. 세 번째 질의는 RDF description 데이타에 대한 경로를 포함하는 질의이고, 마지막 질의는 RDF schema 데이타 경로를 포함하는 질의이다.

표 5 실험에 쓰인 RQL 질의

Schema query(Q1)		subClassOf(http://139.91.183.30:9090/RDF/VRP/Examples/culture.rdfs#Artist)
Data query (Q2)		http://139.91.183.30:9090/RDF/VRP/Examples/culture.rdfs#Painter
SFW query	Q3	select X, Y from {X}http://139.91.183.30:9090/RDF/VRP/Examples/culture.rdfs#paints {Y}
	Q4	select $W, $Z from {: $W} http://139.91.183.30:9090/RDF/VRP/Examples/culture.rdfs#sculpts {: $Z}

5.1 질의 처리 성능

본 실험에서는 표 5.1의 질의를 RDF description 데이타의 사이즈가 500K, 1M, 2M인 데이타베이스에 대해 수행하였다. 각각의 데이타의 RDF schema는 동일하다. 표 6은 실험 결과를 보여준다. 실험 결과에서 Q1과 Q4에 대한 수행 시간은 데이타의 사이즈와 상관없이 일정함을 볼 수 있다. 이는 Q1과 Q4가 RDF schema 데이타에 대한 질의이고, 실험에 쓰인 데이타의 schema 데이타가 일정하기 때문이다. Q2와 Q3의 경우 RDF description 데이타의 사이즈가 클수록, 질의 처리 시간이 길어짐을 볼 수 있다. 특히, Q3의 경우 데이타의 사이즈가 커짐에 따라 처리 시간이 4배정도씩 길어짐을 볼 수 있는데, 이는 RDF description 데이타에 대한 SFW 질의를 XPath 질의로 변환할 때, 여러 XPath 질의들로 변환되고, 이에 따라 조인이 많이 필요하기 때문에 그만큼 시간이 길어진다. 하지만, 본 연구에서 제안한 조인 방법이 다른 조인 방법보다 좀더 나은 성능을 보여준다. 이는 다음 절에서 살펴본다.

표 6 RQL에 질의에 대한 처리 시간 (단위: 초)

	500K	1M	2M
Q1	0.137	0.134	0.133
Q2	1.802	2.874	4.447
Q3	44.384	154.863	610.918
Q4	0.438	0.441	0.437

5.2 RDFSuite와의 성능 비교

그림 8은 Museum RDF 데이타에 대한 표 5의 질의 처리 시간을 RDFSuite 시스템과 비교한 결과이다. Q2의 경우 RDFSuite 시스템과 질의 처리 시간이 거의 비슷하지만, 다른 질의의 경우 본 연구에서 구현한 시스템이 좀더 나은 성능을 보임을 볼 수 있다. 특히, Q3의 SFW 질의의 경우, 앞 절에서 데이타 크기가 커짐에 따라 수행 성능이 나빠짐을 볼 수 있지만, 다른 시스템과 비교했을 경우, 나쁘지 않은 성능을 보여 주고 있다. 이는 본 연구에서 보다 효율적으로 조인 연산을 수행하고 있기 때문이다.

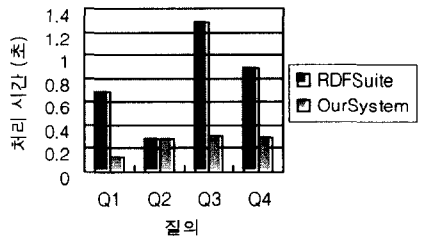

그림 8 RDFSuite 시스템과의 질의 처리 성능 비교

6. 결 론

본 연구에서는 RDF description 데이타와 RDF schema 데이타를 XML 데이타베이스 시스템에 저장하고 이를 검색하는 방법을 제안하였다. 시멘틱 웹이 차세대 웹으로 자리잡기 위해서는 가장 먼저 RDF 기반 기술들이 정립되어야 하고, 기반 기술들 중에서 가장 중요하고 시급한 문제는 RDF 데이타를 저장하고 검색하는 기술이다. 본 연구에서는 RDF 데이타를 XML 데이타베이스 시스템에 저장함으로써, 시멘틱 웹의 주된 데이타인 XML 데이타와 이에 대한 메타 데이타인 RDF 데이타를 통합적이고 효율적으로 다룰 수 있는 방법을 제공한다. 또한, RDF 질의 언어인 RQL 질의 언어를 통해 RDF description 데이타 뿐만 아니라, RDF schema 데이타도 검색할 수 있도록 함으로써, RDF 데이타 모델에 맞게 원하는 데이타를 쉽게 검색할 수 있는 방법을 제공한다. 또한, 저장할 때 RDF schema의 클래스와 속성의 계층 구조를 저장함으로써, 보다 쉽고 빠르게 이 데이타에 대한 RQL 질의를 처리하도록 하였다. 검색할 때는 최대한 데이타베이스 시스템에서 질의를 처리하도록 하였고, 또한 보다 효율적인 조인 기법을 제안하고 있다. 본 연구에서 제안한 여러 기법들의 성능 평가에서도 RDFSuite 시스템보다 질의 처리 성능이 좋음을 확인할 수 있었다.

본 연구에서 제안하고 있는 저장 및 검색 기법들은 지속적인 연구를 통해 보다 효율적인 방법으로 발전할 수 있을 것이다. 하지만, 아직 정립되지 않은 RDF 데이타 저장 및 검색 방법을 XML 데이타베이스 시스템을 사용하여 제안함으로써 앞으로의 RDF 데이타를 다루는 여러 연구들의 새로운 방향을 제시한 데에 본 연구의 의의가 있다고 할 수 있다.

참 고 문 헌

[1] O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, 1999.

[2] T. Berners-Lee, et. al. Uniform Resource Identifiers (URI): Generic Syntax.

[3] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, K. Tolle. The RDFSuite: Managing Voluminous RDF Description Bases, Technical Report, ICS-FORTH.

[4] J. Broekstra and A. Kampman. Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. International Semantic Web Conference 2002.

[5] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl. RQL:A Declarative Query Language for RDF. WWW2002.

[6] D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft, 2002.

[7] T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensible markup language (XML) 1.0. W3C Recommendation, February 1998.

[8] J. Clark, S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation 1999.

[9] D. Beckett. RDF/XML Syntax Specification (Revised). W3C Working Draft 2002.

[10] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. XQuery: A Query language for XML. W3C Working Draft, 2001.

[11] http://www.w3.org/TR/rdf-testcases/#ntriples

서 명 희

2001년 고려대학교 컴퓨터학과(학사). 2003년 한국과학기술원 전산학과(석사). 관심분야는 XML, 시맨틱웹

정 진 완

정보과학회논문지 : 데이타베이스 제 31 권 제 1 호 참조

민 준 기

정보과학회논문지 : 데이타베이스 제 31 권 제 1 호 참조

안 재 용

1999년 고려대학교 컴퓨터학과(학사). 2001년 한국과학기술원 전산학과(석사). 2001년~현재 한국과학기술원 박사과정 재학중. 관심분야는 공간데이타베이스, GIS, 버퍼관리, XML