Journal of Digital Contents Society Vol. 13 No. 1 Mar. 2012(pp. 43-52)

OWL 문서의 변경 탐지 및 관리 기법

김연희*, 김지현**

요 약

정보 자원에 대한 정확한 검색을 위해서는 점진적으로 변화하는 온톨로지의 변경 내용을 효율적으로 관리할 필요가 있다. 최근에는 OWL을 이용해 온톨로지를 기술하는 경우가 많기 때문에 OWL 문서로 작성된 온톨로지의 변경 내용을 관리할 수 있는 기법이 필요하다. 이러한 요구에 따라 본 논문에서는 OWL로 기술된 온톨로지의 변화를 탐지할 수 있도록 변경 가능한 요소를 분류하고 각 요소의 특성에 맞게 변경 내용을 관리할 수 있는 저장 스키마를 제안한다. 그리고 온톨로지 버전별로 클래스나 프로퍼티에 대한 정보를 제공하는 뷰를 이용하여 질의 처리 성능을 개선할 수 있는 가능성을 제시한다. 본 논문에서 제안한 저장 스키마는 각 온톨로지 버전과 관련된 메타데이터의 변경 내용도 함께 저장한다. 또한 온톨로지가 변경되면 자동적으로 추가 또는 삭제되어야 하는 메타데이터의 내용도 추론을 통해 관리할 수 있다. 따라서 본 논문에서 제안한 저장 스키마를 이용하면 온톨로지의 변경 이력에 대한 검색이 가능하고 사용자가 선택한 온톨로지 버전에 맞는 정확하고 유효한 메타데이터를 제공할 수 있다.

Change Detection and Management Scheme of OWL Documents

Youn-Hee Kim*, Jee-Hyun Kim**

Abstract

For accurate search on information resources, it is needed to manage gradual changes in ontology efficiently. Recently, because ontology is often written using OWL, techniques that can manage changes in OWL documents are required. To meet these needs, in this paper, we classify changeable elements to detect changes in OWL ontology and propose a storage schema that can manage the changes according to the characteristics of each element. And we suggest the possibility of improving performance of query processing using views that provide information about classes or properties in each ontology version. The proposed storage schema stores changes in metadata associated with each ontology version. In addition, it can manage metadata that must be added or deleted through reasoning when ontology changes. So, the proposed storage schema can support queries about history of changes in ontology and provide accurate and valid metadata that is suitable for user-selected ontology version.

Keywords : OWL Document, Ontology Change Management, Storage Schema

1. 서론

정보 시스템에서 의미에 기반을 둔 지능화된 검색을 제공하기 위해서는 정보 자원의 의미를 기술할 수 있는 메타데이터와 온톨로지의 역할이 중요하다. 메타데이터는 온톨로지에 정의된 개념을 이용하여 정보 자원의 의미와 정보 자원 간의 의미적 관련성을 자세하게 기술한 것이다. 그리고 온톨로지는 메타데이터를 기술할 때 사용되는 개념을 정의하고 개념간의 관계를 명시한 것으로 메타데이터에 직접적으로 기술되어 있지 않는 새로운 지식을 추론할 수 있는 논리적 근거를 제공한다.

일반적으로 온톨로지는 실세계에 존재하는 개념에 대한 모델링에 기반을 두고 있기 때문에

※ 제일저자(First Author) : 김연희
접수일:2012년 01월 31일, 수정일:2012년 02월 18일
완료일:2012년 03월 15일
* 부천대학교 e-비즈니스과
yhkim@bc.ac.kr
** 서일대학교 컴퓨터소프트웨어과
▣ 본 논문은 2010년도 서일대학 학술연구비에 의해 연구되었음.

시간이 지나면 실세계의 변화와 함께 계속적인 변경이 발생하게 된다[1, 2]. 또한 사용자의 요구에 따라 변경이 발생할 수도 있다. 이러한 온톨로지의 계속적인 변화를 정확하게 관리하지 못하면 변경된 온톨로지와 기존 온톨로지간의 충돌이 발생하여 온톨로지에 대한 신뢰성이 낮아지게 되고 정보 자원의 과거 메타데이터에 대한 검색이나 온톨로지의 변경 이력에 대한 검색 등을 올바르게 지원하지 못하게 된다. 그리고 온톨로지는 점진적으로 내용의 일부가 변화하는 경우가 많기 때문에 새로운 버전의 온톨로지가 배포되면 온톨로지의 전체 내용을 저장하지 않고 이전 버전의 온톨로지에 비해 새로 변경된 부분만을 저장하는 것이 효율적이다[3, 4]. 따라서 점진적인 온톨로지의 변화를 탐지하고 변경 정보를 효율적으로 관리하기 위한 연구가 필요하다.

메타데이터와 온톨로지를 정형화된 형태로 기술하는 다양한 언어들 중에서 풍부한 표현력과 다양한 모델링 요소를 제공하고 W3C의 권고안으로 채택된 OWL(Web Ontology Language)이 표준 언어로 인식되어 관심과 활용도가 높다[5]. OWL은 이전에 많이 활용되던 RDF(Resource Description Framework)와 RDF 스키마에 비해 다양한 추론적 요소들을 포함하고 있으므로 RDF와 RDF 스키마에 대한 변경 관리와는 다른 접근 방법으로 OWL의 변경 내용을 관리할 필요가 있다. 또한 온톨로지의 변경에 따라 관련된 메타데이터의 변경도 함께 발생하기 때문에 그에 대한 고려가 필요하다.

따라서 본 논문에서는 OWL 문서로 작성된 온톨로지 버전의 변화를 탐지할 수 있도록 변경 유형을 분류하고 각 유형별 특성에 맞게 변경 내용을 관리할 수 있는 저장 스키마를 제안한다. 특히 본 논문에서 제안한 저장 스키마는 온톨로지의 버전별 변경 내용뿐만 아니라 그와 관련된 메타데이터의 변경 내용도 함께 관리한다. 그리고 본 논문에서는 온톨로지 버전에 기반을 둔 질의의 처리 성능을 개선하기 위해 뷰를 이용하는 방법을 제안한다.

2. 관련연구

온톨로지의 변경 내용을 관리하려면 변경된 내용이 무엇인지를 탐지하고 분류하는 작업이 먼저 수행되어야 한다. 온톨로지 변경을 탐지하는 기법을 제안한 최근의 연구들은 RDF와 RDF 스키마로 기술된 온톨로지를 주어(subject), 술어(predicate), 목적어(object)로 구성된 트리플의 집합으로 변환한 후 차집합 계산을 이용해 변경을 탐지하는데 초점을 두고 있다[3, 6]. [3]에서는 공노드의 사용 패턴을 이용하여 RDF 문서의 변경을 탐지하는 기법을 제안하였고 [6]에서는 대용량 데이터를 지원하기 위해 관계형 데이터베이스를 이용한 RDF 문서의 변경 탐지 기법을 제안하였다. 온톨로지 변경 탐지에 대한 기존 연구들은 RDF와 RDF 스키마를 대상으로 하는 경우가 많지만 RDF와 RDF 스키마에 비해 OWL이 개념을 명확하게 정의하는데 필요한 추론적 요소를 더 많이 포함하고 있기 때문에 OWL을 대상으로 하는 연구가 필요하다. 특히, 단순히 트리플 집합의 차집합 계산으로 변경을 탐지하기 보다는 OWL이 포함하고 있는 다양한 요소에 대한 변경을 세부적으로 탐지할 수 있도록 유형을 분류하고 각 유형에 적합한 변경 관리를 할 필요가 있다.

시간의 흐름에 따라 제공되는 버전별 온톨로지의 변경 내용을 관리하기 위한 초기의 연구들은 온톨로지의 모든 버전을 저장하거나 여러 가지 기준에 따라 주기적으로 온톨로지의 버전과 변경 집합을 파일의 형태로 저장하는 방법을 주로 사용한다[4]. 그러나 버전에 따라 온톨로지가 변경되면 관련된 메타데이터의 내용이 함께 변경되고 정보 시스템에서는 메타데이터에 대한 관리와 검색도 중요하기 때문에 온톨로지의 변경 내용뿐만 아니라 그와 관련된 메타데이터도 함께 관리되어야 한다. 이러한 요구에 따라 버전별 온톨로지의 변경 내용과 특정 온톨로지 버전에 기반을 둔 메타데이터의 내용을 함께 관리하는 저장 기법에 대한 연구도 진행되었다[7]. [7]에서는 OWL로 기술된 온톨로지의 변경 내용과 그에 관련된 메타데이터를 관계형 데이터베이스를 이용하여 저장하기 때문에 대용량 데이터를 지원할 수 있다. 그리고 소수 레이블 기법[8]을 이용하여 온톨로지 내에 정의된 클래스 또는 프로퍼티의 계층 관계를 쉽게 판별할 수 있다. 그러나 [7]에서는 이행적 특성이나 대칭적 특성, 역관계 등 프로퍼티가 가지고 있는 다양한 추론

적 특징을 고려하지 않기 때문에 그것을 통해 새롭게 유도되는 메타데이터에 대한 검색이 어렵다. 그리고 온톨로지의 핵심 요소인 클래스나 프로퍼티가 새로 추가된 버전과 삭제된 버전은 저장하지만 저장된 메타데이터가 유효성을 가지는 온톨로지의 버전이 무엇인지에 대한 정보는 저장하지 않기 때문에 메타데이터의 변경 이력을 알 수 없다는 문제가 있다.

본 논문에서는 기존 연구의 한계를 극복하고 효율적인 온톨로지의 변경 관리가 가능하도록 OWL 문서로 작성된 온톨로지가 포함하고 있는 변경 요소를 탐지하여 분류하고 온톨로지의 변경 내용은 물론 관련된 메타데이터의 변경 내용도 함께 저장할 수 있는 저장 스키마를 관계형 데이터베이스를 이용하여 개발하는데 목표을 두고 있다.

3. OWL의 변경 요소 분류

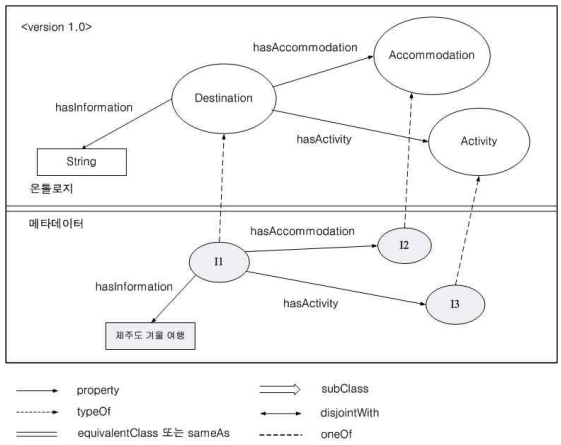
OWL의 변경은 요소의 추가, 삭제, 수정 연산에 의해 발생된다. 일반적으로 OWL 요소의 수정 연산은 요소의 삭제 연산 후 새롭게 추가 연산을 수행한 것과 결과가 같다. 본 논문에서는 OWL의 변경 관리를 보다 단순화하기 위해 변경을 위한 추가와 삭제 연산만을 고려한다. 새로운 버전의 OWL 문서가 배포되면 이전 버전의 OWL 문서와의 차이를 탐지하여 변경된 내용을 추출해야 한다. 이를 위해 본 논문에서는 OWL 문서로 기술된 온톨로지에 대해 <표 1>에서 분류한 유형에 따라 변경을 탐지한다.

<표 1> OWL 문서의 변경 탐지를 위한 요소 분류

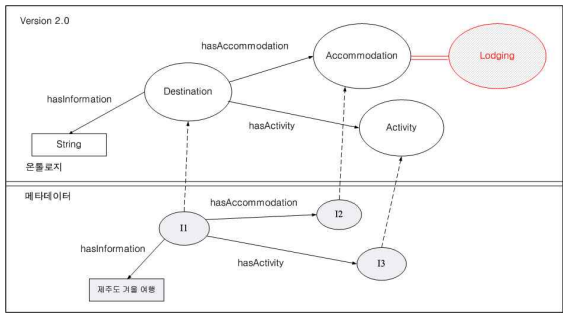
요소	내용
클래스 (class)	클래스 정의의 추가 및 삭제
클래스 제한 사항	클래스 제한 사항의 추가 및 삭제 (oneOf, hasValue, cardinality 등)
클래스 관계	클래스간 관계의 추가 및 삭제 (subClass, unionOf, disjointWith 등)
프로퍼티 (property)	프로퍼티 정의의 추가 및 삭제
프로퍼티 특성	프로퍼티 특성의 추가 및 삭제 (symmetric, transitive 등)
프로퍼티 관계	프로퍼티간 관계의 추가 및 삭제 (subProperty, inverseOf 등)

개념을 표현하는 클래스와 클래스의 속성이나 클래스간의 의미적 관계를 표현하는 프로퍼티가 OWL의 핵심 요소이기 때문에 본 논문에서는 이 두 가지를 변경 탐지의 대상으로 한다. <표 1>에서 제시한 대로 새로 배포된 온톨로지에서 클래스에 대해 정의, 제한 사항, 관계의 추가와 삭제가 탐지되거나 프로퍼티에 대해 정의, 특성, 관계의 추가와 삭제가 탐지되면 변화가 발생한 것으로 판단한다.

버전별로 변경이 발생한 온톨로지와 그것과 관련된 메타데이터의 간단한 예를 살펴보자.



(그림 1) 여행 온톨로지 버전 1.0과 메타데이터

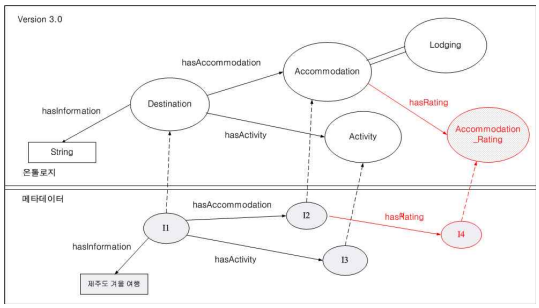


(그림 2) 여행 온톨로지 버전 2.0과 메타데이터

(그림 1)은 여행이라는 주제와 관련하여 OWL 문서로 기술된 온톨로지의 초기 버전과 그것에 기반을 두고 함께 기술된 메타데이터를 그래프 형태로 표현한 것이다. (그림 1)에서 제시한 여행 온톨로지 버전 1.0에는 세 개의 클래스와 세 개의 프로퍼티가 정의되어 있다. 메타데이터에는 "I1"이 "Destination" 클래스 타입의 인스턴스인

개체이고 “I2”는 “Accommodation” 클래스 타입의 개체, “I3”는 “Activity” 클래스 타입의 개체임이 기술되어 있다. 그리고 온톨로지 버전 1.0에 정의된 프로퍼티를 이용하여 개체간의 의미적 관계도 메타데이터에 기술되어 있다.

(그림 2)는 여행 온톨로지 버전 2.0과 메타데이터를 그래프 형태로 표현한 것이다. 온톨로지 버전 2.0에는 버전 1.0에 비해 “Lodging” 클래스의 정의와 함께 “Accommodation” 클래스와 동등 클래스(equivalentClass) 관계임이 추가되었다. 메타데이터에는 직접적으로 추가된 내용이 없다.



(그림 3) 여행 온톨로지 버전 3.0과 메타데이터

(그림 3)은 여행 온톨로지 버전 3.0과 메타데이터를 그래프 형태로 표현한 것이다. 온톨로지 버전 3.0에는 “AccommodationRating” 클래스와 “hasRating” 프로퍼티가 추가되었다. 그리고 메타데이터에는 “AccommodationRating” 클래스 타입의 개체인 “I4”가 추가되고 “I2” 개체와 “hasRating” 프로퍼티 관계를 맺고 있음이 직접적으로 추가되었다.

온톨로지의 변화에 따라 메타데이터의 내용이 간접적으로 변화하는 경우도 있다. 만약 새로운 버전의 OWL 온톨로지에서 <표 1>에 제시한 변경 요소가 탐지되면 추론을 통해 관련된 메타데이터의 내용도 함께 변경되어야 한다.

본 논문에서는 온톨로지 변경에 따라 자동적으로 추론되어야 하는 메타데이터의 변경 내용을 다음과 같이 크게 세 가지로 분류하고 OWL 변경 관리에 고려한다. 첫째, 개체의 클래스 타입을 추론한다. 예를 들어, 하위 클래스(subclass)로 정의된 클래스가 삭제된 온톨로지 버전에서는 삭제된 클래스 타입으로 지정된 개체의 클래스 타입을 삭제된 클래스의 상위 클래스(superclass)로 변경한다. 둘째, 개체의 동질성과 이질성 관계를 추론한다. 예를 들어, 클래스 간의 이질성(disjointWith) 관계가 추가된 온톨로지 버전에서는 각 클래스의 타입으로 지정된 개체들도 이질성(differentFrom) 관계로 추론한다. 셋째, 개체간의 의미적 관계도 새로 추론한다. 예를 들어, 프로퍼티의 역관계(inverseOf), 프로퍼티의 대칭적(symmetric), 이행적(transitive) 특성이 추가된 온톨로지 버전에서는 개체간의 새로운 의미적 관계를 추론한다.

4. OWL의 변경 관리

본 논문에서는 점진적으로 변화하는 OWL의 버전과 변경 내용을 관리하기 위한 저장 스키마를 제안한다. OWL 문서로 기술된 온톨로지의 버전이 여러 개 배포될 수 있고 온톨로지와 관련한 메타데이터의 내용도 함께 관리해야 하기 때문에 대용량 데이터의 지원이 가능한 관계형 데이터베이스를 이용한다.

4.1 OWL 변경 관리를 위한 저장 스키마

본 논문에서는 OWL로 작성된 온톨로지의 버전별 변화를 탐지하여 변경된 내용만을 저장함으로써 저장 공간의 효율성을 높이고 관련된 메타데이터의 변경 사항을 쉽게 추론할 수 있도록 한다. OWL 문서의 변경 내용을 관리하기 위해 본 논문에서 제안한 저장 스키마는 10개의 테이블로 구성된다.

<표 2>는 OWL 문서로 작성된 온톨로지의 버전에 관한 정보를 저장하는 Version 테이블에 대해 설명한다.

<표 2> Version 테이블

필드명	내용
ID(PK)	온톨로지 버전을 식별하는 기본키
name	온톨로지 버전의 이름
creationDate	온톨로지 버전의 생성 또는 배포 날짜
addFactor	이전 버전에서 새로 추가된 요소의 개수
deleteFactor	이전 버전에서 삭제된 요소의 개수

본 논문에서 제안한 저장 스키마에는 버전별로 온톨로지에 정의된 클래스에 대한 정보를 저장하는 Class, ClassRelation, ClassRestriction, ClassMember 테이블이 존재한다. <표 3>은 클

래스의 기본 정보를 저장하는 Class 테이블에 대해 설명한다.

<표 3> Class 테이블

필드명	내용
ID(PK)	클래스를 식별하는 기본키
label	클래스에 부여된 레이블
name	클래스의 이름
addV(FK)	클래스가 추가된 버전 (Version 테이블의 ID 필드 참조)
deleteV(FK)	클래스가 삭제된 버전 (Version 테이블의 ID 필드 참조)

<표 4>는 클래스간의 관계 정보를 저장하는 ClassRelation 테이블에 대해 설명한다.

<표 4> ClassRelation 테이블

필드명	내용
ID(PK)	클래스간의 관계를 식별하는 기본키
CID1(FK)	관계를 맺고 있는 클래스의 ID (Class 테이블의 ID 필드 참조)
CID2(FK)	관계를 맺고 있는 클래스의 ID (Class 테이블의 ID 필드 참조)
type	클래스간의 관계가 무엇인지 구분 (union, intersection, disjointWith 등)
addV(FK)	클래스 관계가 추가된 버전
deleteV(FK)	클래스 관계가 삭제된 버전

<표 5>는 프로퍼티에 대한 제한 사항을 가지고 있는 클래스에 대한 정보를 저장하는 ClassRestriction 테이블에 대해 설명한다.

<표 5> ClassRestriction 테이블

필드명	내용
ID(PK)	클래스가 가지고 있는 프로퍼티의 제한 사항을 식별하는 기본키
CID(FK)	프로퍼티 제한 사항을 가진 클래스의 ID(Class 테이블의 ID 필드 참조)
PID(FK)	클래스와 연관된 프로퍼티의 ID (Property 테이블의 ID 필드 참조)
type	프로퍼티의 제한 사항이 무엇인지 구분 (hasValue, someValue, allValue 등)
value	프로퍼티의 제한 값
addV(FK)	제한 사항이 추가된 버전
deleteV(FK)	제한 사항이 삭제된 버전

<표 6>은 멤버 개체를 제한하고 있는 클래스에 대한 정보를 저장하는 ClassMember 테이블에 대해 설명한다.

<표 6> ClassMember 테이블

필드명	내용
ID(PK)	멤버 개체가 제한된 클래스를 식별하는 기본키
CID(FK)	멤버 개체가 제한된 클래스의 ID (Class 테이블의 ID 필드 참조)
IID(FK)	클래스에 소속된 멤버 개체의 ID (Individual 테이블의 ID 필드 참조)
addV(FK)	멤버 개체 제한이 추가된 버전
deleteV(FK)	멤버 개체 제한이 삭제된 버전

본 논문에서 제안한 저장 스키마에는 버전별로 온톨로지에 정의된 프로퍼티에 대한 정보를 저장하는 Property, InverseProperty 테이블이 존재한다. <표 7>은 프로퍼티의 기본 정보를 저장하는 Property 테이블에 대해 설명한다.

<표 7> Property 테이블

필드명	내용
ID(PK)	프로퍼티를 식별하는 기본키
label	프로퍼티에 부여된 레이블
name	프로퍼티의 이름
domain(FK)	프로퍼티의 도메인 클래스 ID (Class 테이블의 ID 필드 참조)
range(FK)	프로퍼티의 레인지 클래스 ID (Class 테이블의 ID 필드 참조)
type	프로퍼티의 특성을 구분 (symmetric, transitive, functional 등)
addV(FK)	프로퍼티가 추가된 버전
deleteV(FK)	프로퍼티가 삭제된 버전

<표 8>은 프로퍼티간의 역관계 정보를 저장하는 InverseProperty 테이블에 대해 설명한다.

<표 8> InverseProperty 테이블

필드명	내용
ID(PK)	프로퍼티간의 역관계를 식별하는 기본키
PID1(FK)	역관계를 맺고 있는 프로퍼티의 ID (Property 테이블의 ID 필드 참조)
PID2(FK)	역관계를 맺고 있는 프로퍼티의 ID (Property 테이블의 ID 필드 참조)
addV(FK)	프로퍼티의 역관계가 추가된 버전
deleteV(FK)	프로퍼티의 역관계가 삭제된 버전

본 논문에서 제안한 저장 스키마에는 버전별로 정의된 온톨로지와 관련된 메타데이터의 정보를 저장하는 Individual, IndividualRelation, Triple 테이블이 존재한다. 이 세 개의 테이블에는 OWL 문서에 직접적으로 기술되어 있는 메타

데이터뿐만 아니라 온톨로지의 변경에 따라 자동적으로 추론되는 메타데이터도 함께 저장된다. 그리고 메타데이터가 어떤 버전의 온톨로지에 따라 기술되거나 추론되었는지에 대한 정보도 저장하기 때문에 온톨로지 변경에 따른 메타데이터의 변경 이력에 대한 검색을 지원한다. <표 9>는 메타데이터에 기술된 개체에 대한 정보를 저장하는 Individual 테이블에 대해 설명한다.

<표 9> Individual 테이블

필드명	내용
ID(PK)	개체를 식별하는 기본키
name	개체의 이름
classType (FK)	개체의 타입으로 지정된 클래스의 ID (Class 테이블의 ID 필드 참조)
addV(FK)	개체가 추가될 때 참조된 버전
deleteV(FK)	개체가 삭제될 때 참조된 버전

<표 10>은 개체간의 이질성과 동질성 관계를 저장하는 IndividualRelation 테이블에 대해 설명한다.

<표 10> IndividualRelation 테이블

필드명	내용
ID(PK)	개체간의 관계를 식별하는 기본키
IID1(FK)	관계를 맺고 있는 개체의 ID (Individual 테이블의 ID 필드 참조)
IID2(FK)	관계를 맺고 있는 개체의 ID (Individual 테이블의 ID 필드 참조)
type	개체간의 관계가 무엇인지 구분 (sameAs, differentFrom)
addV(FK)	개체 관계가 추가될 때 참조된 버전
deleteV(FK)	개체 관계가 삭제될 때 참조된 버전

<표 11> Triple 테이블

필드명	내용
ID(PK)	트리플 구조의 메타데이터 정보를 식별하는 기본키
subject (FK)	기술된 메타데이터에서 주어인 개체의 ID(Individual 테이블의 ID 필드 참조)
predicate (FK)	기술된 메타데이터에서 술어인 프로퍼티의 ID(Property 테이블의 ID 필드 참조)
object (FK)	기술된 메타데이터에서 목적어인 개체의 ID(Individual 테이블의 ID 필드 참조)
addV(FK)	메타데이터가 추가될 때 참조된 버전
deleteV(FK)	메타데이터가 삭제될 때 참조된 버전

<표 11>은 개체간의 의미적 관계나 개체의 특성에 대해 기술한 메타데이터를 저장하는 Triple 테이블에 대해 설명한다.

4.2 OWL의 변경 관리 전략

본 논문에서는 제안한 저장 스키마를 이용해서 추가와 삭제 연산에 따라 변경된 온톨로지 내용을 관리하고 온톨로지의 변경에 따라 메타데이터 변경 내용을 추론하는 전략을 다음과 같이 제안한다.

4.2.1 추가 연산에 따른 변경 추론

(1) 클래스 추가

배포된 온톨로지 버전에서 새로운 클래스의 추가가 탐지되면 Class 테이블에 새로운 클래스 정보를 삽입하고 addV 필드에 추가가 탐지된 온톨로지 버전을 지정한다.

(2) 클래스의 제한 사항 추가

기존 클래스에 제한 사항의 추가가 탐지되면 유형에 따라 ClassMember나 ClassRestriction 테이블에 추가된 제한 사항에 대한 정보를 삽입하고 addV 필드에 추가가 탐지된 버전을 지정한다. 그리고 추가된 제한 사항에 따라 추론된 새로운 개체에 대한 정보는 Individual 테이블에 삽입하고 addV 필드에 버전을 지정한다.

(3) 클래스 관계 추가

하위 클래스의 추가가 탐지되면 Class 테이블에 추가된 클래스의 정보를 삽입하면서 상위 클래스와의 관계는 클래스에 부여된 레이블로 표현하고 addV 필드에 버전을 지정한다. 동일 클래스가 추가되면 Class 테이블에 추가된 클래스의 정보를 삽입하면서 같은 레이블을 부여한다. 그리고 여러 클래스들의 합집합과 교집합 관계로 정의된 클래스가 추가되면 Class 테이블에 클래스들의 정보를 삽입하면서 하위 클래스와 같은 방법으로 레이블을 부여하고 ClassRelation 테이블에 관계 정보를 삽입한다.

(4) 프로퍼티 추가

새로운 프로퍼티의 추가가 탐지되면 Property 테이블에 새로운 프로퍼티에 대한 정보를 삽입하고 addV 필드에 버전을 지정한다.

(5) 프로퍼티의 특성 추가

기존 프로퍼티에 대해 대칭적 또는 이행적 특성의 추가가 탐지되면 Property 테이블에서 특성이 추가된 프로퍼티의 deleteV 필드에 현재

버전을 지정한다. 그리고 특성이 추가된 프로퍼티의 모든 정보를 Property 테이블에 새로 삽입한 후 addV 필드에 현재 버전을 지정한다.

(6) 프로퍼티 관계 추가

기존 프로퍼티에 대해 하위 프로퍼티나 동일 프로퍼티의 추가가 탐지되면 클래스의 추가와 마찬가지 방법으로 Property 테이블에 추가된 프로퍼티에 관한 정보를 삽입하고 addV 필드에 버전을 지정한다. 만약 프로퍼티의 역관계가 추가되면 InverseProperty 테이블에 삽입한다.

4.2.2 삭제 연산에 따른 변경 추론

(1) 클래스 삭제

Class 테이블에서 해당 클래스의 deleteV 필드에 삭제 연산이 탐지된 온톨로지 버전을 지정한다. 만약 삭제된 클래스가 제한 사항을 가지고 있거나 다른 클래스와 관계를 맺고 있다면 ClassRestriction, ClassMember, ClassRelation 테이블에서 해당 클래스와 관련된 레코드의 deleteV 필드에도 삭제가 탐지된 버전을 지정한다. 그리고 Individual 테이블에서 삭제된 클래스의 타입으로 지정된 개체의 deleteV 필드에 버전을 지정한 후 같은 개체에 대해 클래스 타입을 표현하는 classType 필드를 널 값으로 하는 새로운 레코드를 추가하면서 addV 필드에 버전을 지정한다.

(2) 클래스의 제한 사항 삭제

삭제가 탐지된 제한 사항의 유형에 따라 ClassMember나 ClassRestriction 테이블에서 관련된 제한 사항의 deleteV 필드에 삭제 버전을 지정한다. 함께 삭제되어야 하는 개체가 있다면 Individual과 IndividualRelation 테이블에서 해당 개체의 deleteV 필드에 삭제 버전을 지정하고 Triple 테이블에서도 삭제된 개체에 대해 기술된 메타데이터의 deleteV 필드에 버전을 지정한다.

(3) 클래스 관계 삭제

하위 클래스의 삭제가 탐지되면 Class 테이블에서 해당 클래스의 deleteV 필드에 삭제 버전을 지정하고 Individual 테이블에서 삭제된 하위 클래스 타입으로 지정된 개체의 deleteV 필드에 삭제 버전을 지정한다. 그리고 같은 개체에 대해 클래스 타입을 삭제된 클래스의 상위 클래스로 지정한 새로운 레코드를 Individual 테이블에 삽입하고 addV 필드에 버전을 지정한다. 동일 클래스의 삭제도 하위 클래스와 마찬가지 방법으로 처리한다. 합집합 또는 교집합 관계를 맺고 있는 클래스가 삭제되면 하위 클래스의 삭제와 같은 방법으로 처리한 다음 ClassRelation 테이블에서 삭제된 관계에 대한 레코드의 deleteV 필드에 버전을 지정한다.

(4) 프로퍼티 삭제

프로퍼티의 삭제가 탐지되면 Property 테이블에서 해당 프로퍼티의 deleteV 필드에 삭제 연산이 탐지된 온톨로지 버전을 지정한다. 만약 삭제된 프로퍼티와 역관계를 맺고 있는 프로퍼티가 있다면 InverseProperty 테이블에서 해당 프로퍼티와 관련된 레코드의 deleteV 필드에도 삭제 연산이 발생한 온톨로지 버전을 지정한다. 그리고 Triple 테이블에서 삭제된 프로퍼티를 이용해 개체간의 의미적 관계를 기술한 메타데이터의 deleteV 필드에도 버전을 지정한다.

(5) 프로퍼티의 특성 삭제

프로퍼티의 특성이 삭제되면 Property 테이블에서 특성이 삭제된 프로퍼티의 deleteV 필드에 삭제가 탐지된 버전을 지정한다. 그리고 같은 프로퍼티에 대해 프로퍼티의 특성을 표현하는 type 필드를 널 값으로 하는 새로운 레코드를 Property 테이블에 다시 추가하면서 addV 필드에 버전을 지정한다. 또한 프로퍼티의 특성에 따라 추론된 메타데이터도 Triple 테이블에서 삭제하도록 처리한다.

(6) 프로퍼티 관계 삭제

하위 프로퍼티의 삭제가 탐지되면 Property 테이블에서 해당 프로퍼티의 deleteV 필드에 버전을 지정하고 Triple 테이블에서 삭제된 프로퍼티를 이용해 개체간의 의미적 관계를 기술한 메타데이터의 deleteV 필드에도 버전을 지정한다. 동일 프로퍼티의 삭제도 하위 프로퍼티와 같은 방법으로 처리한다. 프로퍼티간 역관계의 삭제가 탐지되면 InverseProperty 테이블에서 삭제하도록 처리하고 Triple 테이블에서 역관계에 근거를 두고 추론된 메타데이터의 deleteV 필드에도 삭제된 버전을 지정한다.

4.3 뷰를 이용한 성능 개선 방안

관계형 데이터베이스에 저장된 온톨로지에 대한 추론 관련 질의의 처리 성능을 향상시키기 위해 실체화 뷰(materialized view)를 이용할 수

있다[9]. 실체화 뷰는 오라클에서 제공하는 것으로 일반적인 뷰와는 달리 물리적으로 저장되는 특징을 가지고 있다. 이러한 특징을 가지고 있는 실체화 뷰를 이용하면 조인 등을 포함하고 있는 복잡한 질의의 처리 성능을 향상시킬 수 있다. 본 논문에서는 이러한 실체화 뷰의 특징을 OWL 문서의 변경 관리 성능을 향상시키기 위해 이용한다.

```
Create Materialized View Ver3Class
As
  Select Class.ID cno, Class.label label,
       Class.name name, Version.ID vno
  From Class, Version
  Where Version.name = 'Ver3.0'  and
        Class.addV <= Version.ID  and
        (Class.deleteV is null  or
         Class.deleteV > Version.ID);
```

(그림 4) 버전 관리를 위한 실체화 뷰

버전별로 온톨로지의 변경 내용을 관리하는 것은 사용자가 제시한 온톨로지 버전과 관련하여 유효한 정보만을 제공하기 위해서이다. 따라서 본 논문에서 제안한 저장 스키마와 관련한 모든 질의는 사용자가 제시한 버전의 유효성을 판단하기 위해 Version 테이블과의 조인이 반드시 필요하다. 특히, 각 버전의 온톨로지가 포함하고 있는 유효한 클래스와 프로퍼티에 대한 내용을 실체화 뷰를 이용해 미리 생성해두면 질의 처리 시 조인 연산을 줄일 수 있기 때문에 성능 개선 효과가 있다. (그림 4)는 여행 온톨로지 버전 3.0에서 유효한 클래스를 위한 실체화 뷰를 가장 기본적인 옵션을 설정하여 정의한 SQL 문이다.

5. 실험

본 논문에서 제안한 저장 스키마의 질의 처리 성능을 평가하기 위해 인텔 Core2 Duo 2.4GHz CPU와 2GB 메모리, Windows XP 운영체제의 컴퓨터 환경에 설치된 오라클 11g와 Visual Studio 2008을 이용해 C 언어로 OWL 변경 관리 시스템을 구현하여 실험을 수행하였다. 그리고 여행을 주제로 한 온톨로지에 대해 12개의 OWL 문서 버전을 직접 설계하고 생성한 후 실험 데이터로 활용하였다. <표 12>는 본 논문에서 실험 데이터로 사용한 12개 OWL 문서 버전의 특성을 보여준다. 예를 들어 버전 1.0에 비해 버전 2.0에서는 한 개의 클래스가 추가되었다. <표 12>에서 이전 버전에 비해 클래스와 프로퍼티의 개수가 변하지 않은 경우는 클래스간의 관계나 제약 사항, 프로퍼티간의 관계나 프로퍼티의 특성 등이 추가된 것이다. <표 12>에서 제시한 트리플의 개수는 각 버전의 온톨로지와 관련하여 직접적으로 기술된 메타데이터의 트리플 수를 의미한다.

<표 12> 실험 데이터의 특성

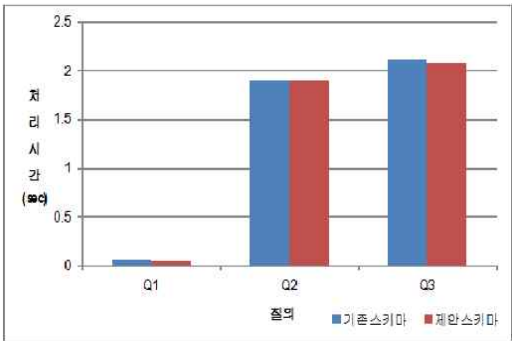
버전	클래스 수	프로퍼티 수	트리플 수
Ver1.0	8	8	30,000
Ver2.0	9	8	30,000
Ver3.0	10	9	40,000
Ver4.0	13	9	50,000
Ver5.0	13	9	60,000
Ver6.0	13	9	80,000
Ver7.0	13	11	100,000
Ver8.0	13	9	80,000
Ver9.0	13	9	60,000
Ver10.0	12	9	60,000
Ver11.0	11	8	50,000
Ver12.0	10	8	50,000

<표 13>은 실험에 사용한 6개의 질의를 보여준다. Q1은 온톨로지 버전 정보 자체를 검색하고 Q2와 Q3은 제시한 버전에서 유효한 온톨로지 정보를 검색한다. 그리고 Q4, Q5, Q6은 제시한 온톨로지 버전과 관련하여 유효한 메타데이터 정보를 검색한다.

<표 13> 실험에 사용된 질의 유형

질의 유형	질의 예
버전 질의	Q1) 추가 연산이 가장 많이 발생한 온톨로지 버전 검색
온톨로지 질의	Q2) Ver 1.0에서 모든 클래스의 이름 검색
	Q3) Ver 7.0에서 “Activity” 클래스의 하위 클래스 검색
메타데이터 질의	Q4) Ver 7.0에서 “Accommodation” 클래스 타입의 개체 검색
	Q5) Ver 7.0에서 “hasActivity” 관계를 맺고 있는 개체 쌍 검색
	Q6) Ver 7.0에서 “hasPart” 관계를 맺고 있는 개체 쌍 검색

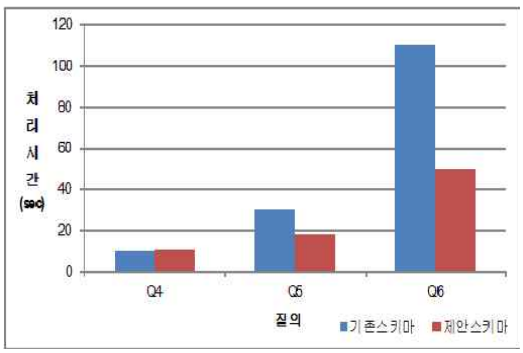
본 논문에서 제안한 저장 스키마의 처리 성능을 평가하기 위해 OWL 문서의 버전과 변경 내용을 관리하는 기존 연구 중 관계형 데이터베이스를 이용하는 [7]에서 제안한 저장 스키마를 비교 대상으로 선택하고 <표 13>에서 제시한 6개의 질의에 대한 처리 시간을 비교하였다. 모든 질의는 데이터베이스 엔진이 구동된 직후 바로 처리하고 10번씩 수행한 후 평균 처리 시간을 초 단위로 측정하였다.



(그림 5) Q1, Q2, Q3 질의 처리 시간 비교

(그림 5)는 본 논문에서 제안한 저장 스키마와 [7]에서 제안한 저장 스키마에 대해 <표 13>에서 제시한 Q1, Q2, Q3 질의의 처리 시간을 비교한 그래프이다. 제안 스키마와 기존 스키마 모두 온톨로지 버전에 대한 정보와 클래스와 프로퍼티가 어떤 버전의 온톨로지에서 추가되고 삭제되었는지에 대한 정보를 별도의 테이블에 저장하여 관리한다. 따라서 Q1, Q2, Q3 질의에 대해서는 제안 스키마와 기존 스키마의 처리 시간은 거의 같으며 같은 개수의 결과 데이터가 반환되었다. 하지만 기존 스키마에서는 온톨로지 버전별로 추가 연산의 횟수와 삭제 연산의 횟수를 별도로 관리하지 않기 때문에 이전 버전에 비해 추가 연산이 많이 발생한 온톨로지 버전을 검색하는 Q1 질의에서 정확한 결과가 반환되지 않았다.

(그림 6)은 <표 13>에서 제시한 Q4, Q5, Q6 질의의 처리 시간을 비교한 그래프이다. 제안 스키마는 기존 스키마와 달리 온톨로지 변경에 따른 메타데이터의 변경 이력을 함께 저장하기 때문에 질의에서 제시된 온톨로지 버전과 관련하여 유효한 메타데이터만을 검색하는 것이 가능하다. 또한 제안 스키마에서는 온톨로지 변경에 따라 자동적으로 추가되거나 삭제되어야 하는 메타데이터의 내용을 추론하여 관리하기 때문에 보다 정확한 검색 결과를 사용자에게 제공할 수 있다. 메타데이터에 대한 내용을 검색하는 Q4, Q5, Q6 질의에 대해 기존 스키마의 검색 결과에는 질의에서 제시한 온톨로지 버전에 맞지 않는 메타데이터의 내용이 포함되어 있었다. 특히, Q4의 경우 제안 스키마의 처리 시간이 약간 느린 것으로 평가되었는데 이것은 제안 스키마는 질의에서 제시한 버전과 관련하여 유효한 개체만을 검색하기 위해 메타데이터의 유효성을 판단하는 시간이 더 필요하기 때문이다. 하지만 시간의 차이가 크지 않으며 앞서 설명한 대로 기존 스키마의 경우 유효하지 않은 개체도 검색 결과로 반환하여 검색의 정확성이 낮아지는 문제가 있기 때문에 제안 스키마가 더 효율적이라고 판단할 수 있다. 그리고 역관계를 맺고 있는 프로퍼티와 관련된 Q5 질의와 이행적 특성을 가지고 있는 프로퍼티와 관련된 Q6 질의의 경우에 기존 스키마는 역관계와 이행적 특성에 따른 추론 기능을 기본적으로 지원하고 있지 않기 때문에 제안 스키마의 질의 처리 시간이 상대적으로 빠르다는 것을 확인할 수 있다. 또한 제안 스키마에서는 메타데이터의 유효성을 확인할 수 있기 때문에 기존 스키마에 비해 검색 결과의 정확도가 높다.



(그림 6) Q4, Q5, Q6 질의 처리 시간 비교

6. 결론

정보 시스템에서 관리하는 정보 자원에 대한

메타데이터를 기술하기 위해 필요한 개념과 개념간의 의미적 관계를 정의하는 온톨로지는 시간이 흐름에 따라 여러 가지 이유로 변경이 발생하게 되고 새로운 버전이 나타나게 된다. 특히 온톨로지는 내용의 일부가 점진적으로 변화하는 특성을 가지고 있다. 이러한 온톨로지의 계속적인 변화를 정확하게 관리하지 못하면 버전별 온톨로지의 충돌이 발생하여 검색의 신뢰성이 낮아지고 온톨로지의 변경 이력에 대한 검색이나 이전 버전의 온톨로지를 이용한 검색을 지원할 수 없는 문제가 발생하게 된다. 따라서 본 논문에서는 풍부한 표현력과 다양한 모델링 요소를 제공하는 OWL 문서로 기술된 온톨로지 버전의 변화를 탐지하기 위해 변경 유형을 분류하고 유형별 특성에 맞게 변경 내용을 관리할 수 있는 저장 스키마를 제안하였다. 그리고 뷰를 이용해 온톨로지 버전별로 유효한 클래스와 프로퍼티의 정보를 관리하여 제안한 저장 스키마의 질의 처리 성능을 향상시킬 수 있는 전략을 함께 제안하였다.

본 논문에서 제안한 저장 스키마는 온톨로지 버전별로 변경된 내용과 온톨로지에 기반을 두고 기술된 메타데이터의 변경 내용을 함께 관리한다. 그리고 온톨로지의 변경에 따라 자동적으로 추가되거나 삭제되어야 하는 메타데이터의 변경 내용도 추론을 통해 함께 관리한다. 따라서 본 논문에서 제안한 저장 스키마를 통해 온톨로지의 버전별 변경 이력은 물론 사용자가 제시한 버전의 온톨로지와 관련된 유효한 정보만을 검색하는 것이 가능하다.

참 고 문 헌

[1] Michel Klein, Atanas Kiryakov, Damyan Ognyanov and Dieter Fensel, "Ontology Versioning and Change Detection on the Web", In Proceedings of EKAW, pp.247-259, 2002.

[2] Giorgos Flouris, Dimitris Manakanatas, Haridimos Kondylakis, Dimitris Plexousakis and Grigoris Antoniou, "Ontology Change: classification and survey", The Knowledge Engineering Review, Vol. 23 No. 2, 2008.

[3] 이동희, 임동혁, 김형주, "내포된 공노드를 포함하는 RDF 문서의 변경 탐지 기법", 정보과학회논문지, 제34권 제6호, pp.518-527, 2007.

[4] Hongwon Yun, Junghwa Lee, and Jungwon Kim, "Ontology Versions Management Schemes using Change Set", Journal of Information Technology Applications & Management, Vol. 12 No. 3, pp.27-39, 2005.

[5] OWL 2 Web Ontology Language Primer, "http://www.w3.org/TR/2009/REC-owl2-primer- 20091027/", 2009.

[6] 임동혁, 이상원, 김형주, "관계형 데이터베이스 기반의 후방향 추론을 이용하는 확장 가능한 RDF 데이터 변경 탐지 기법", 정보과학회논문지, 제37권 제4호, pp.197-202, 2010.

[7] 오우진, "OWL 환경에서 온톨로지 버전관리", 홍익대학교 대학원 컴퓨터공학전공, 2010.

[8] Gang Wu, Kuo Zhang, Can Liu and Juanzi Li, "Adapting Prime Number Labeling Scheme for Directed Acyclic Graphs", DASFAA 2006, pp. 787-796, 2006.

[9] 이훈, 유상봉, "효율적인 온톨로지 저장과 처리를 위한 관계형 데이터베이스 설계", 한국정보기술학회논문지, 제8권 제9호, pp.143-151, 2010.

김 연 희

2000년 : 홍익대학교 컴퓨터공학과 (공학사)
2002년 : 홍익대학교 컴퓨터공학과 (공학석사)
2006년 : 홍익대학교 컴퓨터공학과 (공학박사)
2007년~현 재 : 부천대학교 e-비즈니스과 강의전담교수
관심분야 : 시맨틱 웹, 데이터베이스, 이러닝 시스템

김 지 현

1978년 : 이화여자대학교 수학과 (이학사)
1994년 : 단국대학교 전자정보전공 (경영학석사)
2004년 : 단국대학교 전산통계학과 (이학박사)
1997년 : 정보관리 기술사
1998년~현 재 : 서일대학교 컴퓨터소프트웨어과 부교수
관심분야 : 웹 공학, 데이터베이스, 품질 관리