온톨로지 버전 관리를 위한 변경 집합과 저장 기법

윤홍원*

Change Sets and Storage Polocies for Ontology Versions Management

Hong-won Yun*

요 약

온톨리지는 시맨틱 웹의 핵심 요소이며 시간이 흐름에 따라서 변화하는 특징을 가지고 있다. 온톨리지에서 발생하는 변화는 도메인 변화, 개념 변화, 메타데이터 변화 등이 있다. 이 논문에서는 이들 변화와 시간차원을 기반으로 하는 변경 집합을 제안한다. 온톨리지 버전은 대량의 정보를 저장하고 관리해야 하는데 이를 위해서 변경 집합을 이용한 버전 저장 방법들 제안한다. 제안하는 방법들 사이에 성능을 비교하고 평가한다.

ABSTRACT

In this paper, We describe a change set that is to manage historical information and to maintain ontologies during long term. We propose several storage policies of ontology using a change set. A change set includes a change operation set and temporal dimensions. A change operation set is composed with schema change operations and data change operations. We propose three storage policies that are storing all the change sets and storing the change sets and versions periodically, storing the change sets and the versions using sum of change operations. Also, we present the experimental results for evaluating the performance of different storage policies.

키워드

Semantic Web, Ontology, Versions Management, Storage Policy

I. 서 론

시맨틱 웹(Semantic Web)은 기존 웹(WWW)상의 정보에 잘 정의된 의미(Semantic)를 부여함으로써, 사람뿐만 아니라 컴퓨터도 쉽게 문서의 의미를 해석할 수 있도록 하여, 컴퓨터를 이용한 정보의 검색, 해석 및 통합 등의 업무를 자동화하기 위한 목적으로 제안되었다. 이러한 "잘 정의된 의미"를 다루고자 하는 것이 바로 시맨틱 웹에서 온톨로지의 역할이다[1]. 웹에 의미를 부여한다는 것은, 메타 데이터 개념을 통해 웹 문서에 시맨틱 정보를 덧붙여, 사람이 아닌 컴퓨터가 의미 정보를 자동으로 추출 할 수 있는 패러다임을 조성하는 것을 의미한다[2].

온톨로지는 특정 분야에 대한 재사용이 가능한 지식 조각을 제공함으로써, 시맨틱 웹의 기본 빌딩 블록처럼 여겨진다. 그러나 그 지식 조각들은 정적이지 않고, 시간이 흐를수록 계속 변한다. 따라서 온톨로지의 수정이 일어나게 되고, 그 수정된 온톨로지와 기존의 온톨로지간의 충돌이 발생하는 등

*신라대학교 컴퓨터정보공학부

접수일자 : 2004. 07. 01

의 효율적인 재사용을 방해하는 문제가 발생할 수 있다. 특히, 웹과 같이 분산되고 통제되지 않은 환경에서는 이러한 온톨로지의 변화를 다루는 지원이 반드시 필요하다[3].

또한, 사용자들은 흔히 특정 분야에 대해 과거 정보를 검색하거나, 수정된 온톨로지들 사이의 변화에 대하여 질의하기를 원하기 때문에, 온톨로지의 과거 정보들도 보관해 놓아야할 필요가 있다.

온톨로지 버전 관리는 이러한 온톨로지의 변화와 그 변경 이력의 보관과 관련 있다. 더 자세히 말하면, 온톨로지 버전 관리란 온톨로지의 수정본(즉, 새로운 버전)들을 만들고 관리함으로써 온톨로지의 변화를 다루는 방법을 말한다. 이를 위해서는, 버전들을 구별하여 인식하고, 온톨로지 변경 절차를 다루는 방법론들이 제기되어야 한다. 온톨로지 버전 관리는 대량의 데이터를 다루게 되는데, 대량의 데이터를 관리하기 위해서는 효율적인 저장 정책이 필요하다.

지금까지 다양한 데이터 모델에 대해서 버전을 관리하기 위한 여러 가지 방법들이 개발되어져 왔지만 온톨로지 버전 관리에는 적당하지 않다. 그리고 아직 우리나라에서는 온톨로지 버전 관리 기법에 관한 연구들이 미비한 실정이다. 이에 본 연구에서는 계속 변하는 온톨로지의 효율적인 재사용을 위해, 그리고 사용자들의 과거 정보 질의에 대한 원활한 정보 제공을 위해 온톨로지의 이력 정보를 저장·관리하는 온톨로지 버전 관리 기법에 대해 연구한다.

기존의 온톨로지를 저장하는 간단한 방법으로 온톨로지의 모든 버전을 저장하는 방법이 있다. 이 방법은 가장 단순한 방법으로, 빠른 응답 시간을 제공하지만, 많은 저장 공간을 필요로 한다는 문제점이 있다. 또 다른 방법으로는 버전을 저장하지 않고 변경된 요소들만 링크드 리스트(Linked list) 형태로 저장하는 방법이 있다. 이 방법은 저장 공간은 적게 차지하지만, 링크드 리스트를 따라가 과거 버전을 생성해야 하므로 정보를 검색하는 시간이 많이 걸리게 된다[4].

본 연구에는 온톨로지의 버전과 버전사이의 변화를 변경 집합(Change Set)으로 정의하여 나타내며, 이 변경 집합은 온톨로지의 구조와 내용을 변경하게 하는 연산들의 모음이다. 본 연구에서는 모든 변경 집합을 저장하는 방법, 버전과 변경 집합을 주기적으로 저장하는 방법, 변경 연산의 개수의 누적합이 임계값을 기준으로 변경집합과 버전을 저장하는 방법을 제안한다. 그리고 각 저장 기법의 공간 사용과 접근 시간 측면에서 성능을 평가한다.

II. 관련 연구

2.1 시맨틱 웹에서 온톨로지

온톨로지에 대한 정의는 여러 가지가 있지만, 시맨틱 웹에서의 온톨로지는 Gruber에 따르면, "공유된 개념화(shared conceptualization)에 대한 정형화되고 명시적인 명세(formal and explicit specification)"라고 정의하였다[6]. 여기서 '개념화'란 특정 분야에서 사람들이 사물에 대해 생각하는 바를 추상화한 모델을 말한다. '명시적 명세'란 개념 사용 유형과 사용된 유형의 제약조건들을 명시적으로 정의한 것을 말하여, '정형화된'이란 온톨로지는 프로그램(또는 기계)이 이해할 수 있는 형태이어야 한다는 것을 말한다. 그리고 '공유된'이란 온톨로지가 표현하는 개념이 어느 한 개인에게만 국한되는 것이 아닌 해당 그룹 구성원간의 합의된 지식에 바탕을 두고 있다는 것을 의미한다[2].

온톨로지는 간단히 표현하면 "개념과 개념들 간의 관계, 개념의 속성, 제약조건들로 구성된 사전"으로써, 어느 특정 분야에 관련된 단어들을 계층적 구조로 표현하고 추가적으로 이를 확장할 수 있는 추론 규칙을 포함한다.

시맨틱 웹에서 제공하는 온톨로지는 다음과 같은 특징을 갖는다. 첫째, 온톨로지 작성 주체로 국가나 대규모 단체와 같은 집단뿐만 아니라 개인이나 그룹 같은 소규모의 다수 집단을 지향한다. 둘째, 온톨로지는 확장과 수정을 전제로 작성된다. 시맨틱 웹에서는 이미 정의된 다른 온톨로지를 검색하여 이를 수정하고 확장 또는 결합하여 개별적 목적에 맞게 진화시키는 기능을 추구한다. 셋째, 사용자에게 맞게 특화되거나 전문 분야에 적합하게 세분화되고 경우에 따라서는 상충되는 온톨로지의 모임을 허용한다. 넷째, 시맨틱 웹의 목적과 부합되도록 기계에 의한 처리를 전제로 만들어진다. 다섯째, 온톨로지의 추론 기능을 제공한다. 온톨로지에 직접적으로 표현되지 않은 개념간의 관계를 추론을 통하여 제시하거나 이에 대한 질의에 답할 수 있게 된다[1].

시맨틱 웹에서는 자연어 위주의 기존 웹 문서와 달리 컴퓨터가 해석하기 쉽도록 의미를 부여한 계층을 가지고 있기 때문에 자동화된 에이전트나 정교한 검색 엔진들이 부여된 의미를 이용하여 고 수준의 자동화와 지능화를 이룰 수 있게 된다. 온톨

로지를 핵심으로 "의미"를 부여하고자 하는 시맨틱 웹은 현재의 "링크의 그물"인 웹을 "의미의 그물"로 변환시킬 것이다[7].

2.2 문서의 버전 관리

문서를 생성하고 개발하는 동안 문서 내의 구성요소는 변경되며, 이러한 변경의 작업으로 구성요소의 한 버전들을 다음 버전으로 변환시키는 이력 과정을 버전관리라 한다[8]. 즉, 한 문서의 버전관리는 문서를 생성한 후 사용자 요구사항 변경, 내용 수정, 문서의 기능 추가 또는 기능의 변경 등의 원인으로 시간이 지남에 따라 변화하는 문서의 내용들을 버전으로 관리한다[9].

기존의 버전 관리 시스템 중 SCCS는 소스 코드 파일의 모든 버전을 저장하고 갱신과정을 기록하고 언제 왜 변화가 일어났는지 관리한다. RCS는 가장 최근 파일과 역방향 편집 스크립트를 저장하며, 역방향 편집 스크립트를 적용하여 최신 파일을 제외한 다른 버전을 생성한다. SCCS와 RCS는 텍스트 라인 중심의 문서를 관리하기에 적합하지만 구조적인 문서나 온톨로지 버전을 관리하기에는 적합하지 않다.

XML 문서의 버전을 관리하는 방법으로 UBCC가 있다[10]. UBCC는 현재 유효한 객체들을 중심으로 군집화한다. 한 페이지 안에서 현재 유효한 객체가 임계값보다 작으면 유효한 객체를 다른 페이지에 복사되어진다. 특정 문서의 버전을 재구성하는 것은 버전을 포함하는 객체들을 찾아서 그것들을 정확한 논리적 순서로 생성하는 것과 같다. 그러나 이 방법은 XML 문서의 단기적인 버전 관리에 초점을 맞추고 있고, 내용 변경 중심으로 개발되었으며 구조 변경에 대한 버전 관리가 쉽지 않은 문제점을 가지고 있다.

XML 문서의 버전을 저장하는 가장 간단한 방법으로써 모든 버전을 저장하는 방법이 있다. 이 방법은 문서의 이력 정보를 요구하는 질의가 많은 응용에 적합하며 가장 단순한 저장 방법이다. 하지만 저장 공간을 많이 필요로 하는 문제점을 가지고 있다. 또한 가장 최근 버전과 역방향 델타를 저장하는 방법도 있는데, 이 방법은 저장 공간을 적게 차지하지만, 델타를 이용하여 과거 버전을 생성하므로 이력 정보를 검색하는데 시간이 많이 걸리게 된다[11].

본 연구에서는 위에서 언급한 각종 문서 관리 기법들을 응용하여 온톨로지 버전 관리 기법을 제안한다. 과거 어떤 시점에 유효했던 온톨로지 버전을 찾거나 온톨로지의 이력 정보 검색을 지원하기 위하여, 시간 차원 질의를 지원하는 온톨로지 버전의 변경집합을 정의한다. 변경 집합에서는 온톨로지의 내용 변경과 구조 변경 연산을 정의한다. 많은 양의 온톨로지 버전을 효율적으로 관리하기 위하여, 변경 집합을 이용한 온톨로지 버전 저장 방법을 제안한다. 제안하는 저장 방법에서는 장기적인 관점에서 문서의 버전을 유지할 수 있도록 한다.

III. 온톨로지 버전 관리 기법

3.1 변경 집합

본 연구에서는 온톨로지의 버전과 버전 사이의 변화를 변경집합(Change Set)으로 나타내며, 이 변경 집합은 온톨로지 구조 변경 연산과 온톨로지 내용 변경 연산을 모두 포함한다.

3.1.1 온톨로지 버전과 변경집합

온톨로지의 버전은 초기 온톨로지 형태(즉, V0)에서 시작하여 첫 번째 변화 뒤에 V1이 되고, 그 뒤를 이어 변화의 과정을 계속 거치면서 전체적으로 V1, V2, ..., Vn의 순서를 가지게 된다. 여기서 마지막 Vn은 현재 유효한 버전이거나 최종 버전이 된다. Cn은 n번째 변경 집합이고, Vn은 n번째 버전을 의미한다. Cn+1 변경 집합을 Vn에 적용시키면 Vn+1이 생성된다. 즉, Vn+1 = Vn ∘ Cn+1이다.

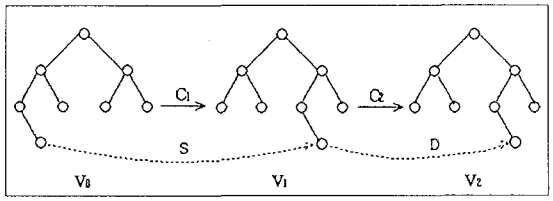

그림 1. 온톨로지 버전과 변경 집합의 관계
Fig. 1 Relationship of ontology versions and change sets

그림 1은 버전과 변경집합의 관계를 보여준다. 그림1에서 트리의 각 노드는 온톨로지의 엘리먼트이고, S와 D는 각각 구조 변경과 데이터 변경이 일어났다는 것을 뜻한다. C1, C2는 변경 집합을 의미하며, C1은 구조 변경에 관련된 구체적인 연산의 목록을 내포하고 있고, C2는 데이터 변경 연산의 구체적인 목록을 내포하고 있다.

온톨로지의 초기 버전인 V0에서 구조의 변경이 일어나서 V1이 생성되었다. 구조 변경 연산은 구조 변경 연산 S에 의하여 이루어 졌으며 이 구조 변경연산은 C1로 표현된다. V0에서 V1이 생성된

과정은, V0 ◦ C1 = V1로 나타낼 수 있다. 마찬가지로, V1에서 데이터 변경 연산 D가 발생하여 V2가 되었으며, 이것은 V1 ◦ C2 = V2로 나타낸다. 따라서, 어떤 버전 Vi에서 다음버전 Vi+1로 버전이 바뀐 것은 Vi ◦ Ci+1 = Vi+1이 된다.

그림 2에서와 같이 버전의 생성 순서가 V1, V2, ..., Vn이고, 변경 집합이 만들어진 순서가 C1, C2, ..., Cn이라고 하면, 전체 버전과 변경 집합 사이의 생성 순서는 다음과 같이 된다.

C1, V1, C2, V2, ..., Vn

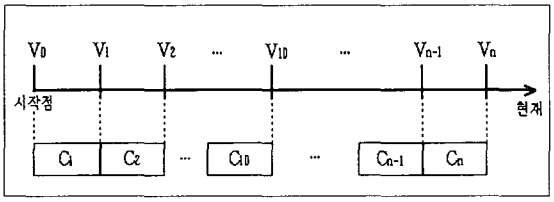

그림 2. 버전과 변경집합 사이의 생성 순서
Fig. 2 Creation sequence between versions and change sets

이때, 생성되는 모든 버전을 다 저장한다면 시간이 흐를수록 대량의 저장 공간이 필요할 것이고, 변경 집합만을 저장한다면 모든 버전을 저장하는 것보다 적은 저장 공간을 차지하지만, 과거 정보를 얻기 위해 변경집합을 적용시켜 버전을 생성하는데 많은 시간이 걸릴 것이다. 따라서, 온톨로지 버전 관리 기법에서는 생성되는 온톨로지 버전을 모두 저장하는 것은 아니다. 어떤 버전을 저장할 것이며, 변경집합은 어떻게 저장할 것인가에 대해서 3.2장에서 살펴본다.

3.1.2 변경 연산

변경 집합은 하나의 버전에서 발생한 변경 연산들의 집합을 말하며, 온톨로지 구조 변경 연산과 내용 변경 연산을 모두 포함한다. 변경 집합 C는 다음과 같이 표현한다.

C = (S ∪ D) ◦ T

여기서 S는 온톨로지 구조 변경 연산을, D는 온톨로지 데이터 변경 연산을, T는 시간 질의를 처리하기 위해서 사용하는 시간 차원을 의미한다. 표 1과 표 2에서 각각 구조 변경 연산과 데이터 변경 연산을 나타내고 있다.

모든 변경 집합 C는 시간 질의를 처리하기 위해 시간 차원을 가진다. 예를 들어 "1999년 12월 1일 이후에 추가된 상품은 어떤 것들이 있는가?", "2000년 상반기에 최고 유행한 아이템은 무엇인가?"와 같은 질의에 응답하기 위해서는, 온톨로지의 버전의 이력 정보를 검색할 수 있도록 지원해야 한다.

시간의 연속을 t1, t2, ···, tn이라고 하면, 어떤 시점 ti에서 생성된 버전의 정보가 실세계에서 유효한 시간을 유효시간으로 나타내고, 버전이 시스템에 기록된 시간을 트랜잭션 시간으로 나타낸다. 시간 차원을 나타내는 T는 유효시간과 트랜잭션 시간을 가지며 다음과 같다.

T = {vt, tt}

3.2 온톨로지 버전 저장 기법

본 장에서는 본 연구에서 제안하는 온톨로지 저장 기법에 대해서 살펴본다. 앞으로 다루게 될 세 가지 저장 기법들은 앞장에서 살펴본 변경 집합을 이용한다. 세 가지 저장 기법은 기본적으로 질의 빈도가 가장 높은 가장 최근 버전과, 특정 버전을 생성하는데 사용되는 최초 버전을 저장해 둔다.

앞으로 살펴볼 온톨로지의 세 가지 저장 기법은 다음과 같다. 첫째, 모든 변경 집합을 저장한다. 둘째, 버전과 변경 집합을 주기적으로 저장한다. 셋째, 변경 연산 누적합의 임계값을 기준으로 버전과 변경 집합으로 저장한다.

본 연구에서 제안하지는 않았지만, 온톨로지 버전 관리 기법 중 가장 간단한 방법으로 온톨로지의 모든 버전을 저장하는 방법이 있다. 이는 질의된 버전을 생성하는데 드는 부가적인 처리 오버헤드가 없으므로 빠른 응답 시간을 제공하지만, 모든 온톨로지의 버전이 저장되기 때문에 많은 저장 공간을 필요로 한다는 문제점이 있다. 이 기법은 본 논문에서 제안된 다른 기법들과 성능을 비교하기 위해 사용할 것이다.

3.2.1 기법 1

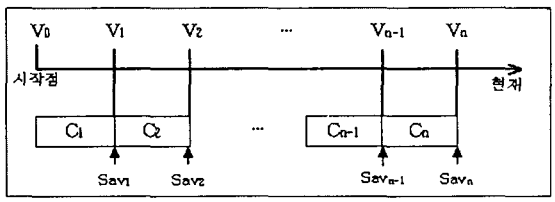

그림 3. 기법 1의 개념
Fig. 3 Concept of scheme 1

기법 1은 최초 버전과 가장 최근 버전, 그리고 변경 집합을 저장한다. 그림 3은 변경 집합을 저장하는 기법 1의 개념을 그림으로 나타낸 것이다.

이 저장 기법은 최초 버전과 가장 최근 버전만을 저장하고 나머지 버전은 생성되면 저장하지 않고 버전과 버전 사이의 변경 집합만 저장한다. 그림 3에서 V1, V2, ..., Vn은 버전을 나타내고, C1, C2, ..., Cn은 변경 집합을 의미한다. Sav1, Sav2, ..., Savn은 각 버전이 생성된 시점이고, 또한 각 변경 집합이 저장되는 시점이다. 그림 3에 의하면, Sav1시점에 V1이 생성되었으며, 변경 집합 C1이 저장된다. 기법 1에서는, 버전 V0과 Vn이 저장되고, C1에서부터 Cn까지의 변경 집합이 저장된다.

기법 1에서 저장되는 버전과 변경 집합을 그림 4에서 나타내고 있다. 기법 1에서 저장되는 버전과 변경 집합의 순서는 V0, C1, C2, ..., Vn과 같다. V0에 C1을 적용하여 V1을 재생성할 수 있다. 이것을 V1 = V0 ◦ C1로 나타낸다. 마찬가지로, V1에 C2를 적용하여 V2를 재생성할 수 있으며, V2 = V1 ◦ C2로 표현 할 수 있다.

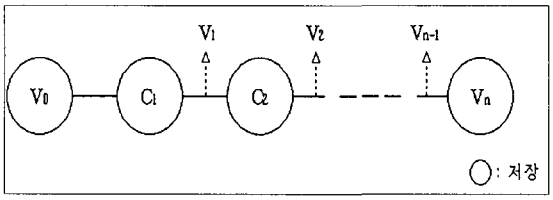

그림 4. 기법 1에서 저장되는 버전과 변경 집합
Fig. 4 Storing versions and change sets in Scheme 1

기법 1에서는 최초 버전과 가장 최근 버전을 제외한 모든 버전을 최초 버전으로부터 다시 만들어야 한다. 따라서 V2는 최초 버전 V0에서 시작하여 다시 만들어야 하며, 이 과정은 V2 = (V0 ◦ C1) ◦ C2가 된다. n-1번째 버전을 재생성하는 과정을 식으로 나타내면, Vn-1 = (((V0 ◦ C1) ◦ C2) ◦ C3)... ◦ Cn-1이 된다.

3.2.2 기법 2

기법 2는 버전과 변경집합의 모음을 주기적으로 저장한다. 저장 주기는 시간 단위가 된다. 이 기법에서는 일정 시간단위에 의해 그룹화 되는 변경 집합의 모음을 사용한다. 그림 5는 변경 집합의 모음과 버전을 주기적으로 저장하는 기법 2의 개념을 그림으로 나타낸 것이다.

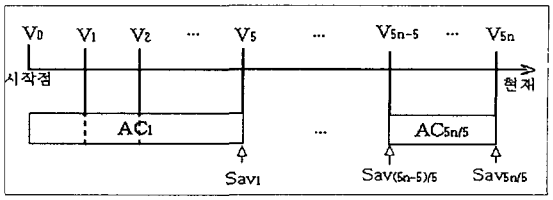

그림 5. 기법 2의 개념
Fig 5. Concept of scheme 2

이 그림에서 저장주기는 5로 가정하였으며, AC1, ..., AC5n/5각각은 저장 주기마다 변경 집합의 모음을 나타낸다. AC1은 첫 주기에 다섯 개의 변경 집합을 모은 것이며(AC1 = C1 + ... + C5), 첫 주기의 저장 시점인 Sav1에 AC1이 저장되고, 또한 V5가 저장된다. 이 기법에서 저장되는 버전은 V0, V5, ..., V5n이고, 저장되는 변경 집합은 AC1, ..., AC5n/5이 된다.

그림 6은 기법 2에서 저장되는 버전과 변경 집합을 나타내고 있다. 이 기법에서 버전과 변경 집합이 저장되는 일련의 순서는 V0, AC1, V5, ..., AC5n/5, V5n이 된다. 최초 버전 V0에 첫 번째 변경 집합의 모음 AC1을 적용하여 V1, ..., V4를 재생성할 수 있으며, 이것을 V1 ~ V4 = V0 ◦ AC1로 표현한다. V5에 변경 집합의 모음 AC2를 적용하여 V6, ..., V9를 다시 만들 수 있고, V6 ~ V9 = V1 ◦ AC2로 표현한다. 기법 2에서 어떤 버전을 재생성하는 과정을 식으로 나타내면, Vq×n+1 ~ Vq×n+(q-1) = Vq×n ◦ ACn+1(n≥0, q:저장주기)이 된다.

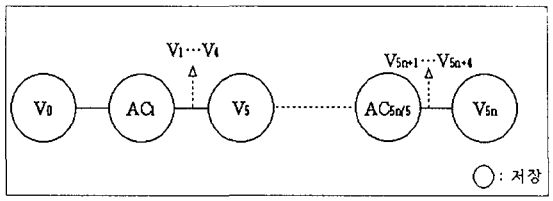

그림 6. 기법 2에서 저장되는 버전과 변경 집합
Fig 6. Storing versions and change sets in scheme 2

3.2.3 기법 3

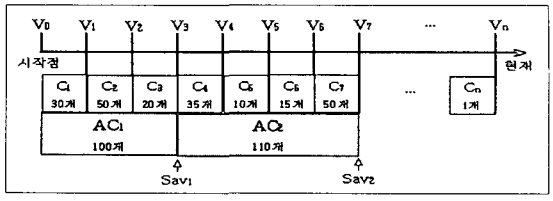

그림 7. 기법 3의 개념
Fig 7. Concept of scheme 3

기법 3은 변경 집합 안의 변경 연산 개수의 누적 합이 임계값을 넘으면 버전으로 저장하고, 저장하는 버전의 직전까지 변경 집합의 모음으로 저장한다. "Count_of_UpdateOperation(변경 집합)"함수를 이용해 각 변경 집합 안의 변경 연산의 개수를 세어, 임계값과 같거나 크게 될 때까지 합산한다. 그림 7은 기법 3의 개념을 도식화 한 것이다.

Count_of_UpdateOperation(C1)을 이용하여 변경 집합 C1의 변경 연산의 개수 30을 얻고, 같은 방법으로 다른 변경 집합 C2, ..., Cn의 변경 연산의 개수를 얻는다. 이를 차례로 합산하여 임계값보다 크거나 같게되면 그 시점을 저장 시점으로 보고 그때의 버전을 저장하고, 그 직전까지의 변경 집합은 변경 집합 모음으로 저장한다. 저장한 후에는 변경 연산의 합산을 다시 초기화한다. 그림 7에서의 변경 연산 합의 임계값은 100이다. 예를 들어 C1의 변경 연산의 개수는 30이고, C2는 50, C3은 20이다. 이를 차례로 더하면 100이 되어 임계값과 같으므로, 이 시점이 첫 저장 시점인 Sav1이 되고, 이때 변경 집합 C1과 C2, C3을 모은 변경 집합 모임 AC1이 저장되고, 또한 버전 V3이 저장된다. 그리고, 변경 연산의 합산은 다시 초기화된다. 같은 방법으로 C4, ..., C7의 변경 연산의 개수를 합산해 보니 110이 되어 임계값보다 크므로, 이 시점이 두 번째 저장 시점인 Sav2가 되고, 변경 집합 모음 AC2가 저장되고, 버전 V7이 저장된다.

그림 8은 기법 3에서 저장되는 버전과 변경 집합을 나타낸 것이다. 기법 3에서 저장되는 버전과 변경 집합의 순서는 V0, AC1, V3, AC2, V7, ..., Vn이 된다. 기법 2에서는 저장될 버전이 미리 결정되어 있었으나 기법 3에서는 저장 시점이 미리 결정되어 있지 않으며, 변경 연산의 합이 임계값과 같거나 크게 될 때 저장한다. 최초 버전 V0에 첫 번째 변경 집합의 모음 AC1을 적용하여 V1, ..., V2를 다시 만들 수 있으며, 이것을 V1 ~ V2 = V0 。AC1로 표현한다. 저장된 버전 V3을 AC2에 적용하여 AC2에 들어있는 변경 집합의 순서대로 버전을 재생성한다.

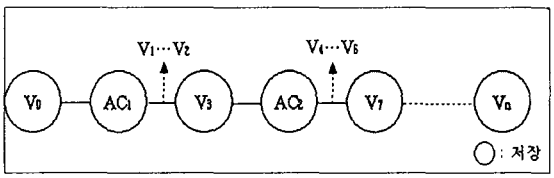

그림 8. 기법 3에서 저장되는 버전과 변경 집합
Fig. 8 Storing versions and change sets in scheme 3

IV. 성능 평가

본 장에서는 기존의 저장 기법과 세 가지 저장 기법들 사이의 성능을 평가한다. 성능 평가는 각 저장 기법이 차지하는 저장 공간과 사용자 질의에 대한 평균 응답시간을 측정하여 서로 비교하였다.

4.1 모의 실험 환경

본 실험에서는 실제 온톨로지를 사용하지 않고, 모의 실험을 통해 성능을 평가하였다. 표 1은 본 실험에서 사용한 모의 실험 환경을 기술하고 있다.

4.2 성능 평가 결과 분석

성능 평가에서 사용한 기존의 저장 방법은 모든 버전을 저장하는 방법을 채택하여 기법 0으로 두어 비교하였다. 모든 버전을 저장하는 방법은 최초 온톨로지로부터 새로운 버전이 생성될 때마다 생성된 버전을 모두 저장하는 방법이며, 각 버전의 변경 집합은 없다. 네 가지 기법에 대하여, 저장되는 버전과 변경 집합, 변경 집합의 크기 변화에 따른 저장 공간 사용, 사용자 질의 수의 변화에 따른 응답 시간을 측정하였다.

4.2.1 저장되는 버전과 변경 집합

실험 주기 365일 동안 발생한 변경 횟수에 따른 버전과 변경 집합의 수를 조사해본 결과는 표 2와 같다. 변경 횟수란 버전이 생성되는 횟수와 같은 의미이다. 본 실험에서 변경 횟수는 랜덤 함수를 이용하여 365일 내에서 변경이 임의로 일어나게 하였다. 변경이 일어날 때마다 각 기법의 저장 정책에 맞게 버전 또는 변경 집합이 저장된다. 생성된 버전들 중 저장되는 버전의 변경 집합은 따로 저장하지 않는다. 즉, 한 저장 시점에서 버전 또는 변경 집합 둘 중 하나만 저장한다.

표 2에는 총 8번의 실험 결과를 제시하고 있는데, 변경 횟수가 랜덤 하게 설정되어 있기 때문에, 실행할 때마다 다른 변경 횟수를 얻게 된다. 표 2에서는 365일 주기 안에서 변경 횟수가 평균적으로 170~200회 정도 일어나는 것을 알 수 있으며, 각각의 기법에서 그 때마다 버전 또는 변경 집합을 저장하게 된다.

기법 0은 변경이 일어날 때마다 모든 버전을 저장하므로, 변경 횟수와 버전수가 항상 같으며 변경 집합은 저장하지 않는다. 기법 1은 최초 버전과 최근 버전만 저장하고, 그 나머지는 변경 집합으로 저장하므로 버전 수는 1, 변경 집합 수는 변경 횟수에서 1을 뺀 수가된다. 참고로, 이 실험 결과에서 버전 수는 최초 버전(V0)을 포함하지 않은 수이다.

기법 2와 기법3은 비슷한 결과를 보이는데, 기법 2는 주기적으로 버전을 저장하고 그 나머지는 변경 집합으로 저장하는 방법으로 본 실험에서는 저장 주기를 30일로 설정하였으며, 저장 주기 내에서 대략 12~14개의 버전이 저장된다. 기법 3은 변경 집합 안의 변경 연산 개의 누적합이 임계값과 같거나 크게 되면 버전으로 저장하고, 그 나머지는 변경 집합으로 저장하는 방법으로, 본 실험에서는 임계값을 350개로 주었으며, 임계값이 350개 일 때 대략 12~14개의 버전이 저장된다. 참고로, 각 변경 집합에서의 변경 연산의 개수는 10~50개 사이의 임의의 값으로 설정하였다.

표 1. 본 실험에서 제시한 모의 실험 환경
Table. 1 Simulation environment

항 목	제시한 값	설 명
온톨로지 크기	100000byte	한 온톨로지 크기, 저장되는 하나의 버전도 이 크기와 동일함
변경 집합 크기	5000byte	버전으로 저장되지 않는 변경 집합의 크기
총 시뮬레이션 기간	365일	성능 평가를 위한 실험 기간
기법 2의 저장 주기	30일	기법 2에서 사용되는 저장 주기
기법 3의 변경 연산 개수 누적합의 임계값	350개	기법 3에서 사용되는 변경 연산 개수 누적합의 임계값
각 변경집합의 변경 연산 개수	10 ~ 50개	랜덤 함수를 이용하여 10~50 사이의 값을 할당
질의 처리 시간	10ms	사용자의 질의 받아 원하는 응답을 해주는데 걸리는 시간
버전 재생성 시간	100ms	변경 집합을 적용시켜 해당 버전을 재생성하는데 걸리는 시간
변경 집합 읽는 시간	10ms	변경 집합을 읽는데 걸리는 시간
총 질의 횟수	100번	실험에 사용되는 사용자 질의 횟수
최근 질의 비율	50%	가장 최근 버전에 저장되어 있는 정보에 대한 질의 비율
과거 질의 비율	50%	가장 최근 버전 이외의 다른 버전들에 저장되어 있는 정보에 대한 질의 비율

4.2.2 변경 집합 크기 변화에 따른 저장 공간

그림 9는 각 저장 기법들이 차지하는 저장 공간을 나타낸 그래프이다. 본 실험에서는 전체 온톨로지에서 변경 집합이 차지하는 비율을 5%, 15%, 30% 45%로 바꾸어가면서 각 저장 기법이 차지하는 저장 공간을 크기를 평가하였다. 본 실험에서는 온톨로지의 전체 크기를 100Mbyte로 설정하였으므로, 변경 집합의 크기는 각각 5Mbyte, 15Mbyte, 30Mbyte, 45Mbyte의 크기를 가진다.

그림 9에서 보는 것처럼, 기법 1이 저장 기법들 가운데 가장 적은 저장 공간을 차지하는 것으로 나타난다. 기법 1은 최초 버전과 가장 최근 버전만 버전으로 저장하고 나머지는 변경 집합만을 저장하기 때문에 저장 공간을 적게 차지한다. 기법 0은 모든 버전을 저장하므로 가장 많은 저장 공간을 차지하는 것으로 나타나며, 다른 저장 기법들보다 지나치게 많은 저장 공간이 필요함을 알 수 있다. 기법 2와 기법 3은 저장 공간을 차지하는 비율이 비슷하게 나타나며, 저장 공간이 가장 적게 필요한 기법 1과 큰 차이는 없음을 알 수 있다.

표 2. 변경 횟수에 따른 버전과 변경 집합의 수
Table. 2 Number of versions and change sets according to No. of update

	변경 횟수	기법	버전수 (단위:개)	변경 집합 수 (단위:개)
1	186	기법0	186	0
		기법1	1	185
		기법2	12	174
		기법3	12	174
2	191	기법0	191	0
		기법1	1	190
		기법2	12	179
		기법3	14	177
3	187	기법0	187	0
		기법1	1	186
		기법2	12	175
		기법3	14	173
4	192	기법0	192	0
		기법1	1	191
		기법2	12	180
		기법3	13	179
5	191	기법0	191	0
		기법1	1	190
		기법2	12	179
		기법3	14	177
6	185	기법0	185	0
		기법1	1	184
		기법2	12	173
		기법3	13	172
7	181	기법0	181	0
		기법1	1	180
		기법2	12	169
		기법3	12	169
8	195	기법0	195	0
		기법1	1	194
		기법2	12	183
		기법3	13	182

또한, 변경 집합의 크기가 커질수록 기법 1, 기법 2, 기법 3의 저장 공간은 조금씩 커지는 것을 알 수 있다. 기법 1, 기법 2, 기법 3은 변경 집합을 저장하기 때문에 변경 집합의 크기가 커지면 저장 공간도 늘어나게 된다. 그러나, 기법 0은 변경 집합을 저장하지 않기 때문에, 변경 집합의 크기가 늘어나는데 대한 영향을 받지 않는다.

4.2.3 질의 수 변화에 따른 평균 응답 시간

그림 10은 네 가지 저장 기법의 사용자 질의에 대한 평균 응답 시간을 나타낸다. 본 실험에서는 사용자 질의 횟수를 100번에서 400번 사이의 값으로 바꿔가면서 각 저장 기법들의 평균 응답 시간을 비교하였다. 본 실험에서는 질의 횟수의 50%는 가장 최근 버전에 대한 질의로, 나머지 50%는 최근 버전이 아닌 다른 버전들에 대한 질의로 설정하였으며, 표 1에서 설명하였듯이, 사용자의 질의 받아 원하는 응답을 해주는데 걸리는 시간은 10ms, 변경 집합을 적용시켜 해당 버전을 재생성하는데 걸리는 시간은 100ms, 변경 집합을 읽어 오는데 걸리는 시간은 10ms로 설정하였다.

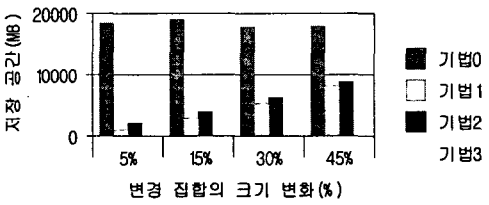

그림 9. 저장 기법들 사이의 저장 공간 비교
Fig. 9 Comparison of storage space among storage policies

그림 10에서 보는 것처럼, 기법 1이 매우 큰 응답 시간을 보이는데, 기법 1에서는 과거 버전을 저장해 놓지 않아, 과거 버전에 대한 질의가 발생하는 경우에, 버전을 재생성하는 시간이 많이 걸리기 때문이다. 기법 0은 가장 작은 응답 시간을 보이고 있는데, 과거 버전을 모두 저장하고 있는 기법 0은 과거 버전을 재생성하는 시간이 들지 않기 때문에 가장 적은 응답 시간을 보인다. 기법 2도 비교적 적은 응답 시간을 보이고 있으며, 기법 3은 기법 0보다 많은 응답 시간을 보이며, 기법 2보다 우수한 성능을 보이고 있다.

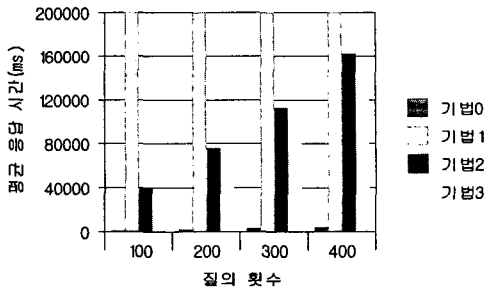

그림 10. 저장 기법들 사이의 평균 응답 시간 비교
Fig. 10 Comparison of response time among storage policies

질의 횟수가 많아질수록 각 기법들의 평균 응답 시간도 늘어나는데, 기법 1은 수치가 너무 높아 그림 10에 전부 나타내지는 못했으나, 질의 횟수가 늘어날수록 급격하게 평균 응답 시간이 늘어났다. 기법 0, 기법 2, 기법 3은 질의 횟수가 늘어날수록 평균 응답시간이 점점 커지는 것을 알 수 있다.

그림 9와 그림 10에서 보듯이, 기법 0은 평균 응답 시간은 우수하나 지나치게 많은 저장 공간을 차지하며, 기법 1은 가장 적은 저장 공간을 차지하지만 사용자 질의에 대해 지나치게 큰 응답 시간을 보이고 있어서, 본 실험에서 실용적으로 사용할 수 있는 기법은 아니라고 판단한다. 기법 2와 기법 3은 저장 공간 측면에서 격차가 거의 없지만, 평균 응답 시간 측면에서는 기법 3이 기법 2보다 나은 성능을 보이고 있다. 따라서 본 실험에서는 기법 3을 가장 효율적인 저장 방안으로 채택한다.

V. 결 론

본 연구에서는 시간이 흐를수록 방대해지는 많은 양의 온톨로지 버전 관리를 위해 온톨로지 버전을 생성하는 변경 집합을 정의하고, 변경 집합을 이용한 온톨로지의 저장 방법을 제안하였다. 변경 집합에는 변경 연산 집합과 시간 차원이 들어 있고, 변경 연산 집합은 구조 변경 연산과 내용 변경 연산으로 구성된다.

본 연구에서 제안한 온톨로지 저장 기법은 기법 1, 기법 2, 기법 3이 있다. 기법 1은 최초 버전과 가장 최근 버전만을 저장하고 나머지는 변경 집합을 중심으로 저장하는 방법이며, 기법 2는 주기적으로 버전을 저장하고, 그 사이사이는 변경 집합으로 저

장하는 방법이고, 기법 3은 변경 집합 안의 변경 연산 개수의 누적합이 임계값을 넘으면 버전으로 저장하고, 그 나머지는 변경 집합으로 저장하는 방법이다.

기존의 저장 기법(기법 0)을 포함하여 본 연구에서 제안한 기법들을 실험을 통하여 성능을 평가하였다. 기법 0은 평균 응답 시간은 매우 우수하나, 지나치게 많은 저장 공간을 차지하였으며, 기법 1은 가장 적은 저장 공간이 필요하지만, 사용자 질의에 대하여 지나치게 큰 응답 시간을 보였다. 기법 2와 기법 3은 비슷하게 적은 저장 공간을 차지하였으며, 기법 3은 응답 시간에서 우수한 성능을 보였다. 따라서, 본 연구에서는 기법 3이 가장 효율적인 온톨로지 버전 관리 기법이라고 제안한다.

본 연구의 한계는 실제 온톨로지를 적용하지 않고, 모의 시뮬레이션 환경을 구축하여 실험을 하였다는 것이다. 향후 실제 온톨로지에 적용하여 버전을 관리하는 기법을 구현할 예정이다.

참고문헌

[1] 이재호, "시맨틱 웹의 온톨로지 언어", 정보과학회지, 제21권 제3호, pp. 18-27, 2003. 03.

[2] 최중민, "시맨틱 웹의 개요와 연구동향", 정보과학회지, 제21권 제3호, pp. 4-10, 2003. 03.

[3] M. Klein and D. Fensel, "Ontology versioning on the Semantic Web", In Proceeding of International Semantic Web Working Symposium(SWWS), pp. 75-91, Stanford University, California, USA, 2001.

[4] Hongwon Yun, "Storage Policies of Ontology Versions in the Semantic Web", International Semantic Web Conference (submitted), Japan, 2004. 11.

[5] Berners-Lee, T., Hendler, J. and Lassila, O., "The Semantic Web", Scientific American, 2001.

[6] Gruber, T., "A translation approach to portable ontologies", Konwledge Acquisition, Vol. 5, No. 2, pp. 199-220, 1993.

[7] Dieter Fensel, James A. Hendler, Henry Lieberman and Wolfgang Wahlster, "Spinning the Semantic Web : Bringing the World Wide Web to Its Full Potential", MIT Press, 2003. 2.

[8] 오상엽, 김홍진, 장덕철, "버전 제어를 위한 소프트웨어 구성요소의 검색 시스템", 한국정보처리학회 논문지 제3권 제5호, 1996.

[9] 정현주, "XML 저장관리 시스템의 효율적인 버전 관리 기법과 문서 저장 방안", 신라대학교 교육대학원 컴퓨터교육전공, 2003. 8.

[10] Shu-Yao Chien, Carlo Zaniolo, "Copy-Based versus Edit-Based Version Management Schemes for Structured Documents", In: RIDE-DM'2001, Heidelberg, Germany, 2001. 4.

저자소개

윤홍원(Hong-won Yun)

E-mail: hwyun@silla.ac.kr

1986년 부산대학교 계산통계학과 졸업(학사)

1990년 한국외국어대학교 경영정보대학원 전자계산학과(이학석사)

1998년 부산대학교 대학원 전자계산학과(이학박사)

2003년 North Carolina State University 객원교수

1996년~현재 신라대학교(구.부산여자대학교) 컴퓨터정보공학부 부교수

※관심분야: 데이터베이스 시스템, 시간 데이터베이스, 인터넷 컴퓨팅