변경 집합을 이용한 온톨로지 버전 관리 기법

윤 홍 원* · 이 중 화** · 김 정 원***

Ontology Versions Management Schemes using Change Set

Hongwon Yun* · Junghwa Lee** · Jungwon Kim***

Abstract

The Semantic Web has increased the interest in ontologies recently. Ontology is an essential component of the semantic web and continues to change and evolve. We consider versions management schemes in ontology. We study a set of changes based on domain changes, changes in conceptualization, metadata changes, and temporal dimension. Our change specification is represented by a set of changes. A set of changes consists of instance data change, structural change, and identifier change. In order to support a query in ontology versions, we consider temporal dimension includes valid time. Ontology versioning brings about massive amount of versions to be stored and maintained. We present the ontology versions management schemes that are 1) storing all the change sets, 2) storing the aggregation of change sets periodically, and 3) storing the aggregation of change sets using an adaptive criterion. We conduct a set of experiments to compare the performance of each versions management schemes. We present the experimental results for evaluating the performance of the three version management schemes from scheme 1 to scheme 3. Scheme 1 has the least storage usage. The average response time in Scheme 1 is extremely large, those of Scheme 3 is smaller than Scheme 2. Scheme 3 shows a good performance relatively.

Keywords : Versions Management, Semantic Web, Ontology

논문접수일 : 2005년 5월 23일 논문게재확정일 : 2005년 7월 21일

* 주저자, 신라대학교 컴퓨터정보공학부 교수, (617-736)부산시 사상구 괘법동 신라대학교 컴퓨터정보공학부,
Tel : 051-999-5065, Fax : 051-999-5657, e-mail : hwyun@silla.ac.kr
** 동의대학교 컴퓨터 · 소프트웨어공학부 교수
*** 신라대학교 컴퓨터정보공학부 교수

1. Introduction

The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. Research on ontology is becoming increasingly widespread in the computer science community. Ontologies have also become important in the Semantic Web. Ontology is an explicit specification of a conceptualization of a domain. Purposes of ontology on the web are making the knowledge about a particular domain explicit, sharing and reusing this knowledge, and analyzing domain knowledge. Like this ontology development is still difficult and time-consuming [Berners-Lee, 1999 ; Berners-Lee et al., 2001 ; Gurrino, 1998].

Ontologies are often seen as basic building blocks for the Semantic Web. Ontologies are like software, they continue to change and evolve over time. The causes of changes are classified as changes in the domain, changes in conceptualization, or changes in the explicit specification. Major changes in ontologies are caused by domain changes and concept changes. Domain changes and concept changes in the shared conceptualization require modifications of the ontology [Klein et al., 2001 ; Noy, 2001 ; Gruber, 1993 ; Noy et al., 2003].

Ontology versioning is related to changes in ontologies. More properly, an ontology versioning consists in a collection of ontology versions. In general it can be said that there is a lack of methods managing ontology. To understand the problem of reusing and evolving ontologies in the Semantic Web, we consider the management of versions in ontology. In this paper, to manage versions in ontology we study a set of changes based on domain changes, changes in conceptualization, metadata changes, and temporal dimension. We use the most common aspect is valid time in order to support historical query in ontology versions.

Ontology versioning brings about massive amount of versions to be stored and maintained. To management massive data, efficient versions management schemes are necessary. Some of version management methods have been developed; these methods can be used to manage versions in different document models [Marian et al., 2001 ; Chien et al., 2001 ; Benatallah et al., 2003]. There are some version management methods to handle XML documents. But, these schemes are unlikely to be appropriate for ontology versions management on the semantic web. Maintainers may wish to keep versions information for classes, properties, and individuals. To keep versions information, we propose change sets to store changed schema and instances. Also we propose the three ontology versions management schemes using change sets and evaluate them.

The rest of the paper is organized as follows. Section 2 introduces some causes of change in ontologies. Section 3 describes the change specification considering time dimension. Section 4 presents the three schemes for ontology versions management. Our experimental results are described in Section 5. Conclusions can finally be found in Section 6.

2. Causes of ontology change

An ontology defines a common vocabulary for researchers need to share information in a domain. It includes machine-interpretable definitions of basic concepts in the domain and relations among them. One widely cited definition of an ontology is Gruber's [Gruber, 1993]. According to Gruber [Gruber, 1993], an ontology is a specification of a shared conceptualization of a domain. Wiederhold [Wiederhold, 1994] describes four types of domain differences : (1) terminology : different names are used for the same concepts, (2) scope : similar categories may not match exactly ; their extensions intersect, but each may have instances that cannot be classified under the other, (3) encoding : the valid values for a property can be different, even different scales could be used, (4) context : a term in one domain has a completely different meaning in another.

This ontology continues to change and evolve. Either causes changes in ontology: changes in the domain, changes in the shared conceptualization, or changes in the specification [Klein, 2001]. It is stated that ontology has classes, class hierarchy, instances of classes, slots as first-class objects, slot attachments to class to specify class properties, and facets to specify constraints on slot values [Noy et al., 2002 ; Chaudhri, 1998]. Those elements involve ontology change.

The research in ontologies started with defining what a formal ontology, shifted to the development of representation languages, and developed the method of evolution and versioning [Noy et al., 2003]. OntoView [Klein et al., 2002] is the ability to compare ontologies at a structure level. OntoView have the comparison function distinguish between the following types of change : non-logical change, logical definition change, identifier change, addition of definitions, deletion of definitions. OntoView deal with four types of change. The complexity of ontology evolution increases as ontologies change and evolve, so a systematic ontology management is required.

We examine a number of ways to represent these change information for an ontology versions. A number of research works were done on ontology change, whereas through studies concerning change specification are still lacking. In this paper, we deal with the change specification in an ontology versioning, also taking into account temporal dimension aspects.

Large ontologies are essential components in web service systems. As ontologies become larger and longer lived, an amount of versions to be stored and maintained become massive volume. We will study the versions management schemes using the change specification. Change sets are used as logs in database to maintain versioned data, so we will show a proper policy taking into account both average response time and storage space usage. Also, we present the experimental results for evaluating the performance of different schemes.

3. Change sets

The distributed and dynamic character of the web will cause that many versions and

variants of ontologies will arise. Domain changes or changes in the conceptualization might cause modifications of the ontology. In case of changes arising in the ontology, ontologies can be read in as a whole, either by providing a URL or by uploading them to the system. It is not appropriate for managing ontology versions. In this case, the system needs large volumes to handle them and spaces are wasted.

3.1 Basic Concepts and Definitions

In this section, we introduce a change specification for changes in ontologies. Our change specification is represented by a set of changes. We assume that a set of changes consists of instances change, structural change, identifier change, and temporal dimension. Instances change in ontology is comparable to update in database instances, e.g., update of a slot values, restrictions or meta data. Structural changes occur when classes are added, removed or moved and change of properties like a change in the classes. Identifier change means a rename of class in ontology. Temporal dimension includes valid time.

The valid time of a fact is the time when the fact is true in the modeled reality. A fact may have associated any number of instants and time intervals, with single instants and intervals being important special cases [Jensen, 1998].

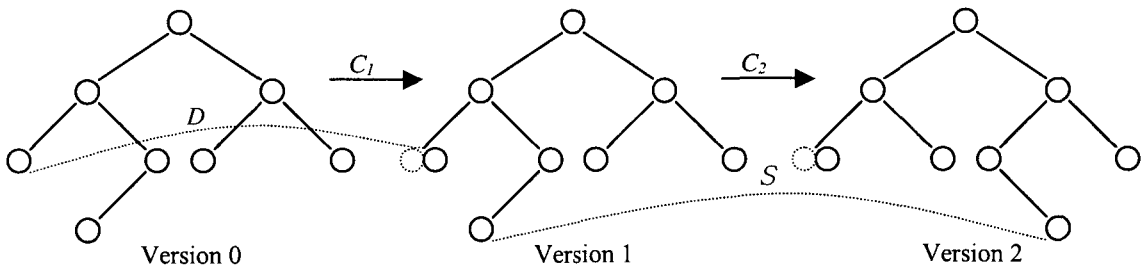
To formally represent a set of changes, we use the symbols as following :

- D : instances change
- S : structural change
- I : identifier change

We define that a set of changes contains four elements are related to changes and time dimension T.

- $C = (D \cup S \cup I) \circ T$

Suppose C_n is a set of changes for n^{th} and V_{n-1} is a version for $n\text{-}1^{th}$ $(n \geq 1)$. A set of changes C_{n+1} applied to V_n that result in V_{n+1}, i.e. $V_{n+1} = V_n \circ C_{n+1}$. <Figure 1> shows a relation between versions and change sets. In <Figure 1>, C_1 or C_2 means a set of changes. D and S mean that instances change and structural change respectively. For instance, we can apply C_1 to Version 0 and then we can produce Version 1. C_1 contains a list of specific operations related to instances change when just instances changes are occurred. Also, C_2 contains a list of structural change operations.



<Figure 1> Relation between version and change set

〈Table 1〉 List of change sets

Category	Set of change
Instance change	*D* = { *Eid, ObjectName, OperationName* }
Structural change	*S* = { *Eid, OperationName* }
Identifier change	*I* = { *Eid, OperationName* }

〈Table 2〉 Basic operations and semantics in change sets

Operations	Semantics
UpdateInstance (*nv, ov*)	Modifies *ov* the instance of the element to *nv*
DeleteInstance ( )	Deletes the instance of the element
InsertInstance (*nv*)	Inserts *nv* the instance of the element
RenameInstance (*ne, oe*)	Renames *oe* the name of the element to *ne*
MoveInstance (*n, p*)	Moves the value in the class *p* to be the value of class in position *n*
CreateClass (*n, p*)	Creates a superclass in position *n*
MoveClass (*n, p*)	Moves the subclass rooted in the class *p* to be the class in position *n*
DeleteClass ( )	Deletes the subclass rooted in the class
InsertClass (*n, C*)	Inserts the class *C* as a subclass of the class in position *n*
UnionClass (e_1, e_2, *p*)	Merges the subclass e_1 and the subclass e_2 results in the class rooted in the class *p*
DivideClass (e_1, e_2, *p*, n_1, n_2)	Divides the class *p* into the class e_1 and the class e_2, and inserts each class in position n_1, n_2
RenameId (*nEid, oEid*)	Renames *oe* the identifier of class to *ne*
InsertSlot (*s, p*)	Inserts the slot *s* as a property of the class *p*
UpdateSlot (*ns, os, p*)	Modifies *os* the slot of the class *p* to *nv*
DeleteSlot (*p*)	Deletes the slot of the class *p*

<Table 1> shows a list of change sets. In <Table 1>, an *Eid* identifies uniquely a particular element in an ontology. The changed object is described with *ObjectName*. The temporal dimension is the core of ontology versioning to support the temporal query. The temporal dimension includes valid time as following :

- *T* = { *Eid, vt* }
  vt = [*valid_start_time, valid_end_time*]

The time dimension naturally has hierarchy as year, month, week, and day. Here we have years as the coarsest granularity and days as the finest granularity.

3.2 Basic Operations and Sematics

<Table 2> gives operations and semantics in detail the corresponding to *OperationName* in the change sets. In <Table 2>, *nv* is a new value after update and *ov* is an old value

previous update. As mentioned previous, change set C is a union D, S, and I, and a combination with temporal dimension T. A change set C is generated like C_1, C_2, C_3, ... whenever an ontology version is created. We can suppose C_{n+1} is a set of changes for $n+1^{th}$ and V_n is a version for n^{th} and then apply C_{n+1} to V_n, we can make V_{n+1} as a version for $n+1^{th}$.

As we mentioned previous, a version is represented by change sets. In order to generate a version from another one, we maintain change sets. To generate a particular version, change sets are applied to previous stored version. A set of changes has temporal dimension to process temporal queries. Examples of temporal queries are: what are the product's names update since time t? When did the car Hyundai Elantra add the directory? The way for querying the history is to search change sets includes that information and reconstruct the version.

4. Versions Management Schemes

4.1 The Three Schemes

In previous section, we described the change sets and the basic operations to represent changes in ontologies. We discuss the management schemes of versioned data in this section. We need to store the first version in the ontology repository as well as the last version. The first (origin) version can be used to create a particular version and the last version be used to query current data. The schemes may be considered as following :

- Scheme 1 : Storing all the change sets
- Scheme 2 : Storing the versions and the aggregation of change sets periodically
- Scheme 3 : Storing the versions and the aggregation of change sets using an adaptive criterion

Whenever versions are created, scheme 1 stores previous the sequence of changes but not versions. The aggregation of change sets from the earliest version to the current version is stored. To access to historical information, this scheme need to generate the versions from the earliest version. There is additional processing overhead to generate queried versions. We can aggregate the change sets periodically and store them. Scheme 2 stores the aggregation of change sets and the versions periodically. Aggregations of change sets are obtained via dividing all the sequence of change sets periodically. When historical queries are occurred, it is required to generate any historical versions from a particular version.

Also, we can aggregate the sequence of change sets not a period but a criterion. Above the third policy means that each versions and change sets are stored using a criterion. When the most update operations within a period are occurred at that time the scheme 3 stores the versions and the aggregation of change sets. The criterion is determined based on a number of update operations. This scheme can reduce the version creation time when temporal queries are requested.

4.2 Scheme 1

<Figure 2> shows the storage policy concept

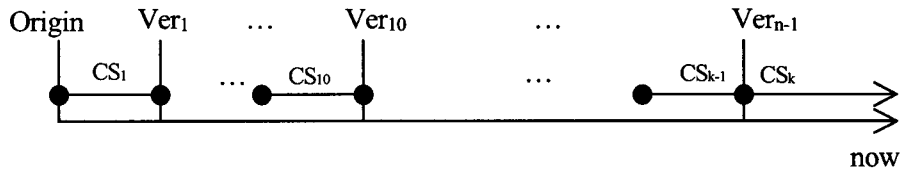
of storing all the change sets. The originated version and the last version (current version) are stored in the storage. Also, A change set is stored whenever a version is generated. However, Versions those are between the originated and the current version, they are not stored. In <Figure 2>, CS_i $(0 < i \leq k)$ means a set of change. A version is represented by an entity: (Ver_{j-1}, CS_i). $Ver_{j-1} \circ CS_i$ generate Ver_i $(1 \leq j < n - 1)$, i.e., Ver_j is generated by applying CS_i to Ver_{j-1}. If only generating versions must start from the originated version. Here origin is a fixed time specifying the creation time of the particular ontology and now corresponds to the current time.

4.3 Scheme 2

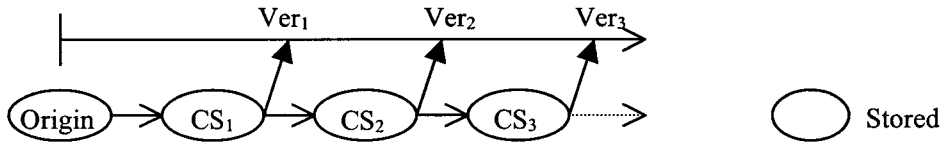
<Figure 3> presents a sequence to generate the particular version when historical queries are occurred on the version. Scheme 1 stores the first version, the last version and just change sets. Applying CS_1 to Ver_0 generates Ver_1, i.e., $Ver_1 = Ver_0 \circ CS_1$. Also, We can get Ver_2 applying CS_2 to result of $Origin \circ CS_1$, i.e., $Ver_2 = (Origin \circ CS_1) \circ CS_2$. If a query is occurred to get a past information on the Ver_3, Ver_3 is generated by $((Origin \circ CS_1) \circ CS_2.) \circ CS_2$.

Above we mentioned scheme 2 that is the versions and the aggregation of change sets periodically, in this scheme, we store a particular versions and an aggregation of changes sets periodically in the repository. <Figure 4> shows scheme of this storage policy.

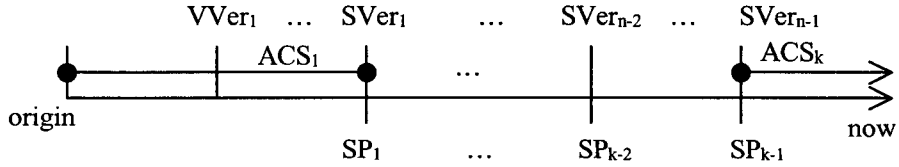
The storing period can become the time granularity as day and month etc. At this scheme we use an aggregation of change sets,



〈Figure 2〉 Scheme of storing all the change sets



〈Figure 3〉 Sequence of generating a particular version in Scheme 1



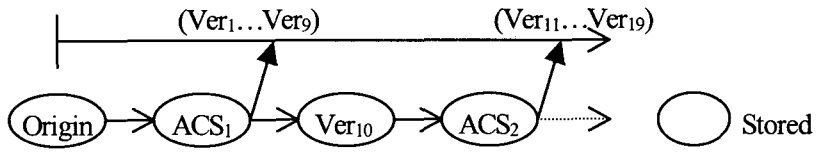
〈Figure 4〉 Scheme of storing the versions and the aggregation of change sets periodically

which means to group by the granularity. We have a 4-level time hierarchy: year, month, week, and day. Here we have days at the finest granularity. Change sets can be grouped in days, weeks, months, or years, and then stored in the repository. In this figure, we assumed the storing period is SP ($SP = SP_{k-1} - SP_{k-2}$) that is constant. Here $VVer_1$ is virtual version that is not stored physically and $SVer_1$ is stored version at save point with SP_1. The ACS_1 is the aggregation of change sets that aggregates all change sets before $SVer_1$ from origin to SP_1. The version data $SVer_1$ and the aggregation of change sets ACS_1 are stored in the repository at point with time SP_1. We assume that $VVer_{n-2}$ is not stored physically as virtual version, and then $VVer_{n-2}$ is reconstructed with the aggregation of change sets ACS_{k-2} before $SVer_{n-2}$ from time SP_{k-3} to time SP_{k-2} i.e., $VVer_{n-2} = (SVer_{n-2} \circ ACS_{k-2})$.

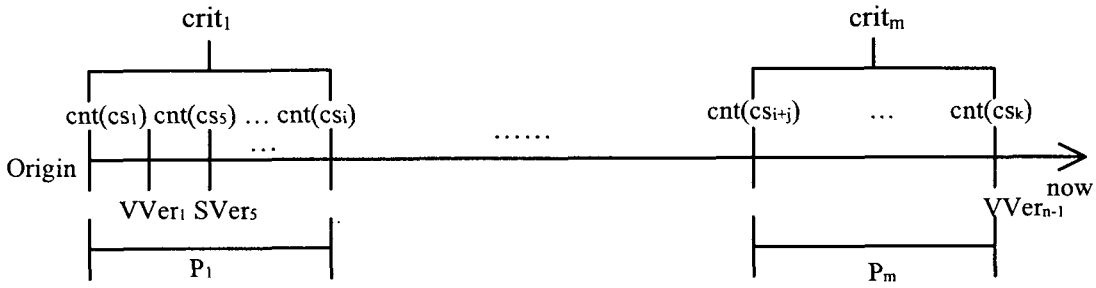
<Figure 5> presents the process to generate a particular version in scheme 2 when some queries are occurred on past version. Scheme 2 stores versions and aggregations of change set as shown in <Figure 5>. All versions between Ver_1 and Ver_9 are generated by applying ACS_1 to *Origin*. Also, applying ACS_2 to Ver_{10} generates all versions between Ver_{11} and Ver_{19}. To generate a particular version, a change set is applied to it's a previous version.

4.4 Scheme 3

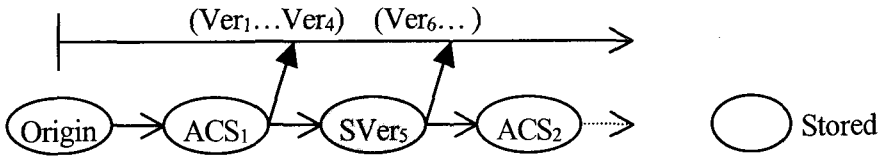
The storage policy that is storing the versions and the aggregation of change sets using a criterion is shown in <Figure 6>. The basic idea of this storage policy is to get a change set which has maximum number of update operations. The maximum value corresponds to the maximum number of update operations in a period, which is a criterion to store a version.



<Figure 5> Sequence of generating a particular version in Scheme 2



<Figure 6> Scheme of storing the versions and the aggregation of change sets using a criterion



〈Figure 7〉 Stored versions and aggregations of change sets in Scheme 3

To get the criterion, we execute the count function during a period and get the maximum value from all the counted value. Here *cnt* is a count function to get a number of update operations and $crit_i$ is a maximum value corresponding to specific time. If *cnt* (cs_5) is a maximum value in this figure, $crit_1$ becomes a criterion to store a particular version $SVer_5$. Just a version is stored at a time instant corresponding a criterion. Other versions are not stored physically during a period except a version that corresponds to a criterion; only aggregations of change sets are stored.

<Figure 7> shows the stored versions and the aggregations of change sets in Scheme 3. In order to reduce the version generation time, we need to get a version that has a maximum value of update operations during time granularity. For example, $SVer_5$ has a maximum value of update operations. $SVer_5$ is already stored in repository ; we don't need to generate it. A lot of processing time to need for generating Ver_5 is saved.

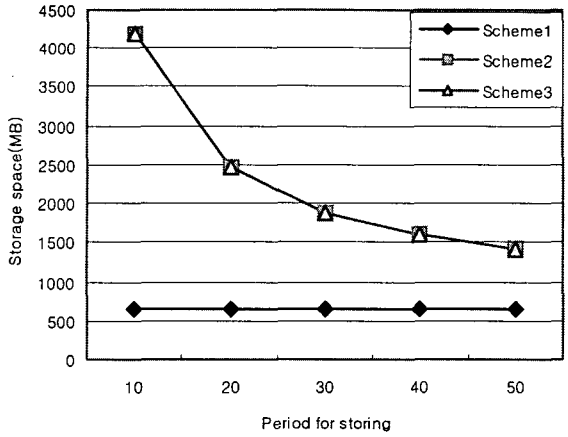
5. Experiments and Analysis

In this section, we present the experimental results for evaluating the performance of the three schemes described in Section 4. We compare the storage space usage and the average response time of the three schemes. A PC with Pentium 4 processor and Windows XP has been used to run the simulation program. We use the following experimental settings <Table 3> :

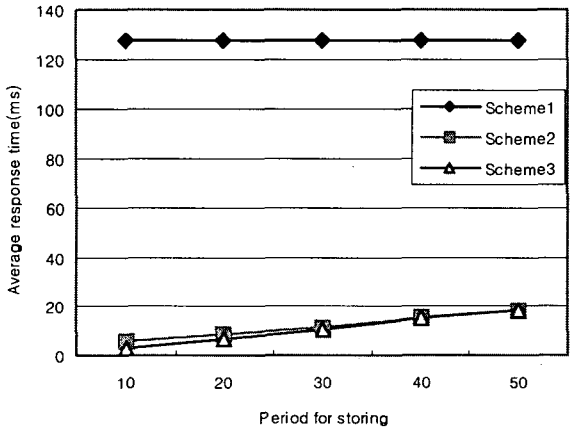
〈Table 3〉 Experimental settings

Parameters	Values
Lifespan	365 days
Change period	3 day
Number of queries	100
Max percent of change	40 %
Query processing time	10 ms
Ontology size	100 MB
Version generation time	1000 ms
Max number of operations	10000

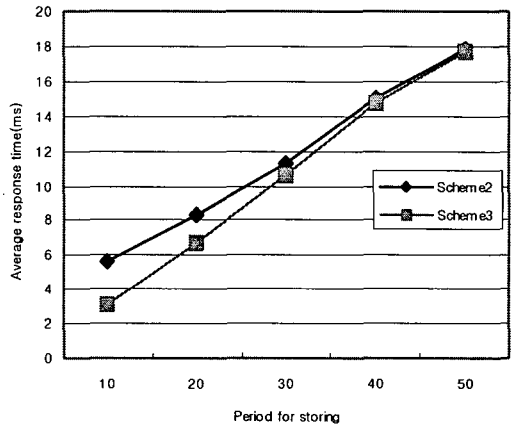
<Figure 8> shows the storage space usage according to change of time period. A period for storing a version varies from 10 to 50. We can see that Scheme 1 has the least storage usage. This is due to the fact that Scheme 1 just stored change sets between the first version and the last version. Scheme 2 and Scheme 3 have similar storage space usage, because they store versions and change sets in turn. Difference of the storage space usage between Scheme and Scheme 3 is small. The Scheme 3 uses a little less space than the Scheme 2. By choosing the largest change set as stored version, the storage space usage of the Scheme 3 is slightly lower than the Scheme 2.



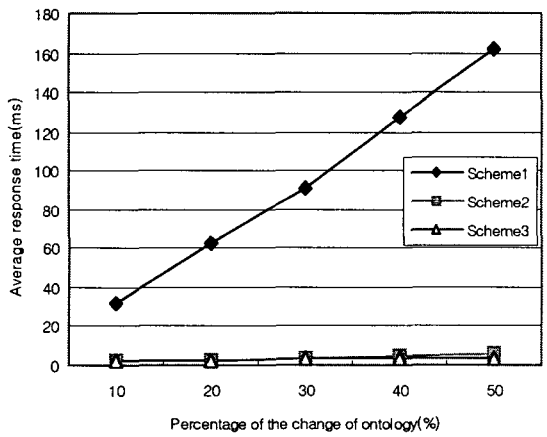
〈Figure 8〉 Storage space usage



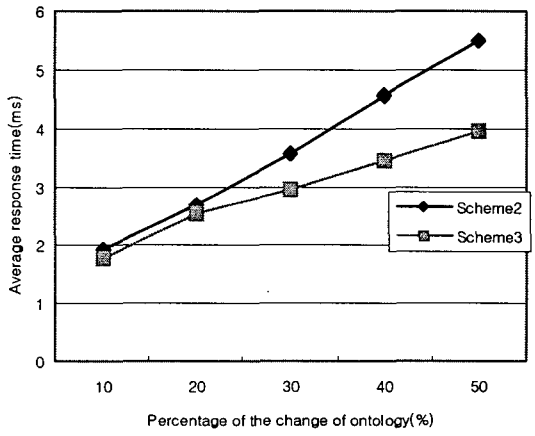
(a) Changing period for storing



(b) Changing period for storing (zoom in)



(c) Changing percentage of the change



(d) Changing percentage of the change (zoom in)

〈Figure 9〉 Average response time

<Figure 9> (a) shows the resulting average response time for the three schemes. We can see that Scheme 1 has the largest average response time and Scheme 3 has the smallest one. The average response time in Scheme 1 is extremely large. This is due to that Scheme 1 needs very much versions generating time because it just stored change sets. Precisely, <Figure 9> (b) shows the average response time just about two schemes. When a period for storing is equal to 10, the average response time of Scheme 3 is smaller than Scheme 2 because very much changed versions are already stored. There is not need to generate those versions. When period for storing increases, the average response time for Scheme 3 is close to those of Scheme2 because the number of generating versions is small, then the effect for average response time is lower. <Figure 9> (c) shows the average response time according to the percentage of change. The average response time in Scheme 1 increases dramatically as the percentage of the change of the ontology. We can see that the percentage of the change can significantly affect the average response time because of the increase of versions generating time. <Figure 9> (d) shows precisely the average response time about two schemes. When percentage of the change of ontology increases, the difference between the Scheme 2 and Scheme 3 increases. This is due to the fact that very much changed versions are already stored in Scheme 3.

6. Conclusions

In this paper we described about the version management schemes in ontology. We have studied cause of ontology change based on domain changes, changes in conceptualization, metadata changes, and temporal dimension. Our change specification is represented by a set of changes. A set of changes consists of instance data change, structural change, and identifier change. In order to support an ontology query language that supports temporal operations, we considered temporal dimension includes valid time. Ontology versioning brings about massive amount of versions to be stored and maintained. We discussed several management schemes of versioned data: Storing all the change sets, Storing the versions and the aggregation of change sets periodically, and Storing the versions and the aggregation of change sets using a criterion. Scheme 1 stores previous the sequence of changes but not versions. Scheme 2 stores the aggregation of change sets and the versions periodically. The basic idea of scheme3 is to get a change set which has maximum number of update operations. Also, we presented the experimental results for evaluating the performance of the three version management schemes from scheme 1 to scheme 3. Scheme 1 has the least storage usage. Scheme 2 and Scheme 3 have similar storage space usage, Scheme 3 uses slightly smaller than the Scheme 2 precisely. The average response time in Scheme 1 is extremely large. Scheme 1 needs very much versions generating time because it just stored change sets. When a period for storing is small, the average response time of Scheme 3 is smaller than Scheme 2. We can

see that the percentage of the change can significantly affect the average response time because of the increase of versions generating time. When percentage of the change of ontology increases, Scheme 3 shows a good performance relatively.

References

[1] Benatallah, B., Mahdavi, M., Nguyen, P., Sheng, Q.Z., Port, L., and McIver, B., "An Adaptive Document Version Management Scheme", The 15th Conference on Advanced Information Systems Engineering (CAiSE'03), Austria, 2003, pp. 16-20.

[2] Berners-Lee, T. (with Mark Fischetti), Weaving the Web, The original design and ultimate destiny of the World Wide Web, Harper, 1999.

[3] Berners-Lee, T., Hendler, J., and Lassila, O., The Semantic Web : A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities, Scientific American, May 2001.

[4] Chaudhri, V.K., Farquhar, A., Fikes, R., Karp, P.D., and Rice, J.P., "OKBC : A programmatic foundation for knowledge base interoperability", In 15th Nat. Conf. on Artificial Intelligence (AAAI-98), 1998, pp. 600-607.

[5] Chien, S.Y., Tsotras, V.J., and Zaniolo, C., "XML Document Versioning", *SIGMOD Record*, Vol. 30, 2001, pp. 46-53.

[6] Deborah, McGuinness, L., Fikes, R., Rice, J., and Wilder, S., "An Environment for Merging and Testing Large Ontologies", In Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000), Colorado, 2000.

[7] Gruber, T.R., "A translation approach to portable ontology specifications", *Knowledge Acquisition*, Vol. 5, No. 2, 1993.

[8] Guarino, N., "Formal Ontology in Information Systems", Proc. of the 1st International Conference, Trento, Italy, 6-8 June 1998.

[9] Heftin, J. and Hendler, J.A., "Dynamic ontologies on the web", In Proc. of AAAI/IAAI 2000, 2000, pp. 443-449.

[10] Jensen, C.S., Dyreson, C.E., (Eds.), M. B¨ohlen, Clifford, J., Elmasri, R.A., Gadia, S.K., Grandi, F., Hayes, P., Jajodia, S.K., K¨afer, W., Kline, N., Lorentzos, N., Mitsoupoulos, Y., Montanari, A., Nonen, D., Peressi, E., Pernici, B., Roddick, J.F., Sarda, N.L., Scalas, M.R., Segev, A., Snodgrass, R.T., Soo, M.D., Tansel, A.U., Tiberio, P., and Wiederhold, G., The Consensus Glossary of Temporal Database Concepts - February 1998 Version, in : O. Etzion, S. Jajodia, S. Sripada (Eds.), Temporal Databases—Research and Practice, Springer-Verlag, lNCS No. 1399, 1998, pp. 367-405.

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

[12] Klein, M., Kiryakov, A., Ognyanov, D., and Fensel, D., "Ontology Versioning and Change Detection on the Web", In 13th International Conference on Knowledge Engineering and Knowledge Management *(EKAW02)*, Sig-üenza, Spain, October 1-4, 2002.

[13] Maedche, A., Motik, B., Stojanovic, L., Studer, R., and Volz, R., "An Infrastructure for Searching, Reusing and Evolving Distributed Ontologies", WWW2003, Hungary, 2003.

[14] Marian, A., Abiteboul, S., Cobena, G., and Mignet, L., "Change-Centric Management of Versions in an XML Warehouse", In Proc. of 27th Int. Conf. on Very Large Data Bases (VLDB), 2001, pp. 581-590.

[15] Noy, N. and Musen, M., "PROMPTDIFF: A fixed-point algorithm for comparing ontology versions", In 18th National Conference on Artificial Intelligence (AAAI2002), 2002.

[16] Noy, N.F. and Klein, M., "Ontology evolution : Not the same as schema evolution", Knowledge and Information Systems, 5, 2003.

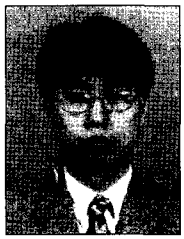
[17] Noy, N.F., "Ontology Engineering", In Proceedings of the International Semantic Web Working Symposium (SWWS), Stanford University, California, USA, p. 2.

[18] Wiederhold, G., "An Algebra for Ontology Composition", In Proceedings of 1994 Monterey Workshop on Formal Methods, U.S. Naval Postgraduate School, 1994, pp. 56-62.

□ 저자소개

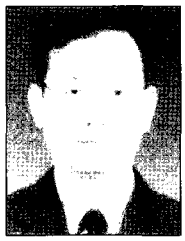
윤 홍 원

저자는 부산대학교 계산통계학과를 졸업하고 같은 대학의 전자계산학과에서 박사를 취득하였다. 현재 신라대학교 컴퓨터정보공학부 교수로 재직하고 있다. 주요 연구분야는 데이터베이스, 시간 데이터베이스, 시맨틱 웹 등이다.



이 중 화

저자는 부산대학교 전자계산학과를 졸업하고 동 대학원에서 이학박사 학위를 취득하였다. 현재 동의대학교 컴퓨터-소프트웨어공학부 조교수로 재직중 이다. 주요 관심 분야는 데이터베이스, 시맨틱 웹, 한글정보처리 등이다.



김 정 원

저자는 부산대학교 전자계산학과를 졸업하고 동 대학원에서 전자계산학 석사, 박사 학위를 취득하였다. 기술신용보증기금에서 기술평가역 차장으로 근무하고 현재 신라대학교 컴퓨터정보공학부 조교수로 재직중이다. 주요 관심분야는 임베디드 시스템, 센스네트워크 등이다.

◈ 이 논문은 2005년 5월 23일 접수하여 1차 수정을 거쳐 2005년 7월 21일 게재확정되었습니다.