X-Spread - A software modeling approach of schema evolution propagation to XML documents

Vincent Nelson Kellers da Silveira and Renata de Matos Galante

Instituto de Informática - Universidade Federal do Rio Grande do Sul
{vincent, galante}@inf.ufrgs.br

Abstract. This paper presents X-Spread, a software modeling approach to propagation of XML schemata evolution to XML documents referring the modified schemata. This mechanism focuses on modifications of XML schemata on different distribution scenarios considering the possible distribution scenarios of schemata and documents as well, allowing for software engineers to focus on application design and not in the issue of XML documents adaptation. This mechanism is composed by change detection algorithms, responsible by the detection of differences between schemata, by storage of the different schema versions, by a revalidation process of existing documents, considering only modifications performed on parts of the schemata and by an adaptation process of documents considered invalid due to schema modifications, with the mechanism execution as a whole demanding no participation of the end user. This mechanism addresses semistructured artifacts distribution scenarios usually not addressed by papers about propagation of XML schema modifications to documents.

Key words:XML, schema, evolution, adaptation

1 Introduction

The disjoint nature of XML documents and schemata, associated to the potential distributed nature of semistructured database enabled applications are factors that increase the complexity of XML schema evolution process, adding to the software modeling complexity of current large software systems. Given that modifications performed on schemata are not automatically reflected on documents, that may imply on invalidation of documents until the documents or schemata themselves are fixed.

Applications based on exchange of schemata and XML messages are examples of applications that may be subject to a schema evolution process. A distributed book database, updated and queried over a network could use the schema depicted on Figure 1a. In a new version of the application responsible for the distributed data management such schema could be changed in order

to address new attributes not modeled by the first application implementation, with the addition of elements to the database, as depicted on Figure 1b.

Considering the distributed nature of this database, the entirety of documents stored by the database may not be accessible on a given time instant, since a given network node may not be operational. Other than the physical distribution of documents referring to a schema, the schema itself may be found in only one network node or replicated on different nodes, referred over the network by applications performing document validations. An alternative scenario for schema distribution is observed when a given schema version is encapsulated in an application performing user-based generation of new database records.

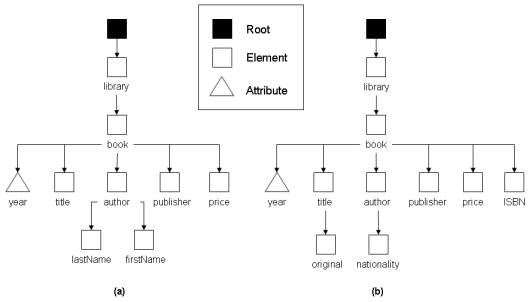

Fig. 1. XML schema evolution

Usually the impact of schemata evolution on XML documents is not addressed by recently proposed mechanisms of schema evolution such as [1], [2] and [3]. When this feature is specifically addressed, some XML artifacts usage scenarios were not considered, for instance, XML documents automatically generated by applications or documents found on unreachable network nodes at the moment of schema evolution [4], what renders these solutions as incomplete.

This work aims to provide a mechanism that stands as a common platform for detection, storage, document revalidation and propagation of modifications performed on schemata to XML documents referring to schemata subjected to an evolution process, considering the different distribution scenarios of the involved artifacts, with minimization of user input during the documents adaptation process.

The contribution of this work is based on the approach of different distribution scenarios of semistructured artifacts usually not addressed by studies on propagation of modifications on XML schemata to documents. Such scenarios encompass XML documents found on unreachable network nodes at the moment the schema is modified or documents that can be deemed as invalid when sent to other network nodes.

The proposed mechanism stands as a common platform for XML documents adaptation leveraged by a whole set of different semistructured data enabled applications, which allows engineers to focus on the design of these application

themselves, leaving the XML documents adaptation to be handled by the proposed mechanism.

The rest of the paper is structured as follows: Section 2 briefly exposes the proposed mechanism to later describe the differences detection algorithm applied to XML documents, the logical data model adopted for storage of different schema versions and associated information, the approaches to document revalidation after schema modification and the adaptation process of invalid XML documents. Section 3 discusses related work and Section 4 concludes the paper with final remarks and comments on future work.

2 X-Spread Overview

The mechanism proposed in this paper is based on four main components:

- Diff - Difference detection between different XML schema versions.
- Store - Storage of XML schema versions and related information.
- Revalidation - Revalidation of XML documents referring to modified schemata.
- Adaptation - Adaptation of XML documents considered invalid after schema modifications.

Figure 2 depicts the X-Spread components and the relationship between them.

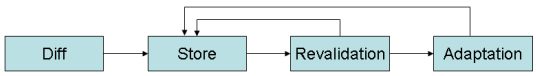

Fig. 2. X-Spread components

The first X-Spread component, Diff, performs difference detection on versions of a given XML schema. Differences found between versions of a schema are named deltas, and form the scripts on which transformation from the original schema version to newer versions is based on.

Diff's generated deltas and XML schema versions themselves are stored by Store component, in order to build the historical record on schema evolution. Since XML documents referring to considerably older versions of the schema may become part of the database at any given moment, the history of modifications performed on a schema must be ensured to exist, in case these documents have to be adapted to newer versions of the schema.

Component Revalidation takes as input a set of XML documents, a set of schema versions and a set of deltas referring to these versions, and performs validations on portions of the XML documents, restricting the validations to portions potentially affected by the schema evolution process.

Component Adaptation receives as input a set of XML documents identified as invalid by the Revalidation component, a set of deltas and a XML schema,

and proceeds with the documents adaptation. The component task is to make a set of XML documents valid again with respect to the schema specification. In the following sections we describe each X-Spread component in details.

2.1 Diff - Difference Detection Algorithms for XML Schemata

The Diff component is based on difference detection algorithms for XML documents. This component receives as input two XML documents and it generates the specification of the differences found as output. This component works with configurations set by the X-Spread administrator, allowing the specification of XML schema locations, whose evolution process must be observed.

As described on [5], many factors must be taken into account when studying XML documents comparison algorithms, for instance, their time and space complexity, output quality, the number of supported operations and distinction between ordered and unordered documents.

Even though usually schemata do not constitute large documents, the time complexity of some algorithms for XML documents comparison may generate results on unacceptable time frames on scenarios where the schemata must be compared with high performance, such as applications in a web environment.

Since the algorithm output will lead to the generation of documents adaptation scripts, the output quality of the algorithm is quite relevant. Thus, the algorithm must generate only correct, relevant and preferably compact outputs, with the concatenation of several primitive operations for representation of complex update operations. Outputs featuring these characteristics will cause improved performance when the adaptation scripts are applied to invalid XML documents.

Existence of an open source implementation of the difference detection algorithm for XML documents is relevant for this work given that definition and implementation of a whole new diff algorithm is not on the scope of this work.

Considering the information gathered on differences detection algorithms for XML documents, the performance and the output quality of algorithms with an available open source implementation, the chosen algorithm was XyDiff[1] [6].

X-Spread mechanism abstracts the implementation of schemata comparison algorithm, allowing for future algorithm substitution, in case a new, better suited algorithm is developed to this task, with no impact on other X-Spread components and on the stored set of schema versions and deltas.

2.2 Store - Schema and Deltas Storage

Store component is responsible by storage of XML schema versions and Diff component outputs. Store, as pictured in Figure 3, has three different modules: DeltaPointer, responsible by translation of deltas generated by the Diff component, in order to guarantee an homogeneous delta storage even if the difference detection algorithm is changed; Persist, responsible by delta and schema versions storage itself and Retrieve, responsible by queries to stored artifacts.

[1] Open source implementation available at http://potiron.loria.fr/projects/jxydiff

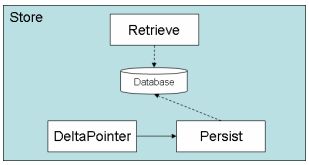

Fig. 3. Store component modules

DeltaPointer module is responsible by translation of Diff output and generation of XPath references, in order to ensure the storage of deltas generated by Diff component in a standard format, indifferently of the adopted difference detection algorithm.

Most of the XML schema modifications detected by the Diff component will lead to the generation of two XPath references, each associated to an operation identifier. Operations such as element or attribute deletion will lead to the generation of a single XPath reference.

Persist module is responsible by storage of DeltaPointer output and the input schemata submitted to the component. These artifacts persistence is needed since they may be queried in order to start the adaptation process of an invalid XML document generated by an application based on an older version of a schema.

In order to perform schemata and deltas persistence, the set of approaches proposed for the storage of XML documents addressing time [7] [8] and version aspects [9] [10] [11], or approaches using the parametric data model are not suitable, since they do not hold to one premise used on the Persist module regarding XML schemata: all stored versions must be valid. Inclusion of new attributes on XML schema artifacts will lead the artifact to an invalid state.

For instance, versioning and temporality related attributes are not part of the original format definition [12], and modeling of such attributes on XML schema would demand development and usage of new schema parsers and document validators, which should be attached to these schemata in all environments using them.

Discarded the inclusion of new XML schema elements and attributes, XML schema versioning attributes were discarded for versioning control as well. Version attribute defined in [12] has no associated semantics, which renders the attribute ignored by XML documents validators.

Usage of different namespaces for each different schema version was also discarded since it implies on explicit modification of XML documents referring to the schema, what breaks the second premise used on the Store component: XML documents must not be changed due to X-Spread's schemata storage model.

With that in mind, the logical data model depicted on Figure 4 was developed for schema versions and deltas storage. Physical implementation of this data model can be done through a relational database or even through storage of schema versions and deltas on the file system, with storage of additional information, such as validity dates, version identifiers, references to other versions

and identification of operations in a XML document developed specifically for this purpose.

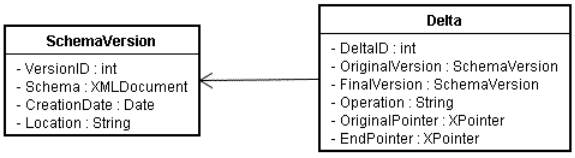

Fig. 4. Store component modules

Retrieve module is responsible for supplying querying tools to deltas and schema versions persisted by Store component. Independently of the DBMS or the storage device used for persistence of XML artifacts, queries performed against this repository will always be performed through the same interfaces.

2.3 Revalidation - XML Documents Revalidation

The Revalidation component is responsible by validation of a set of XML documents, finding out, after the schema evolution process, whether the documents are still valid.

This component has as premise the high performance, given that the number of documents to revalidate is unknown, and ability to abstract the XML documents location, applying the same algorithms both to documents stored at the file system where the schema is found and to documents exchanged over a network.

The Revalidation component has settings specified by the X-Spread administrator, allowing for specification of locations both on local and remote file systems whose contents are subject to revalidation when a XML schema modification is recorded by Store component. Changing these settings will lead to the start of a document revalidation process, considering that the newly included locations on the component settings may contain invalid XML documents.

Input given to the Revalidation component is based on a set of deltas generated by the Diff component, a set of XML documents and a set of XML schema versions. Based on the set of deltas, a minimum set of XML documents sections is defined, where this minimum set of sections corresponds only to parts of the XML schema affected by modifications. The minimum set of sections is used in a revalidation process quite similar to that described in [13].

This revalidation takes into account only sections of the XML documents in order to improve the revalidation process performance, given that a full revalidation of the XML documents could imply in low performance of the X-Spread mechanism.

Several approaches to XML documents incremental validation were recently described. However, some of them, such as [14] are not suitable to some scenarios addressed by the X-Spread mechanism, since they demand a pre-processing phase of the XML documents.

On X-Spread in particular, the full set of XML documents referencing a given schema is not known by the time the schema is modified. Given that the documents may be located at other network nodes, with their inclusion on the set of documents to revalidate only after the schema evolution has already taken place, or the documents may be automatically generated by user applications that necessarily implies no pre-processing can be performed.

Validation of XML documents generated by applications is achieved through the specification of communication ports that must be watched by a proxy and reverse proxy tool like WebScarab[2].

This tool must have not only the ability to intercept requests and responses exchanged over a network, where these requests or responses are originated from or sent to a node hosting X-Spread, but also it must have abilities to manipulate these requests contents and execute the very same validations performed by X-Spread on XML documents located on a file system.

The distinct feature of the revalidation process applied to XML documents exchanged over a network and on the scenario of inclusion of different and new file system locations whose contents must be revalidated on the components configurations is based on the fact that one of the parameters accepted by the Revalidation component, the schema version referred to by the XML documents, can be unknown: prior to identifying the document validity with respect to newer schema versions, the application has to identify to what schema version the XML document is referring to, in order to start the adaptation process in case the document is deemed as invalid.

A first approach to schema identification on XML messages is based on schema extraction from the message, followed by comparison of the extracted schema with versions recorded by the Store component.

A second approach is based on validation of the document with respect to the current version of the schema and, if the document turns out to be invalid, a validation of the document with respect to previous schema versions follows, until a schema where the document is deemed as valid is found.

A third approach is based on finding out if the deltas generated by the Diff component have any reference to the element set that caused the invalidation of the document with respect to the current schema version. In case these deltas are found, the document will be adapted by applying on the document a chain of reverse deltas, until the document is in a state where it can be validated with respect to the schema version that features modifications on elements that first caused the document invalidation. In case the document is once again deemed as invalid, the process will be repeated until the whole chain of schema deltas is applied to the document or until the applications finds out that the elements causing the document invalidation are subject of no delta, what would halt the

[2] Implementation available at http://www.owasp.org/software/webscarab.html

revalidation and adaptation process of the document. In case the schema version referred to by the document is found, the remaining processes can take place.

In order to avoid full execution of the schema detection process whenever a XML message with no schema associated is exchanged over a network, a cache can be created, associating a given network address or file system location to a given schema version, improving the performance of the schema detection process on future validations originated from or executed on a file of a cached location.

Considering the designed functionality of the Revalidation component, a possible execution scenario is based on unfeasibility of identification of the schema used by a XML document, what can happen when the Diff component was not able to identify partial versions of a XML schema, what means that no delta and versions storage process was executed. When this scenario is detected, the Revalidation component will sign to X-Spread's administrator such occurrence and the revalidation and adaptation process of the document will be halted.

XML documents stored on a file system or exchanged over a network and deemed as invalid with respect to a schema by the end of the revalidation process, along with the referred schema itself and a set of deltas will be taken as inputs by the Adaptation component, which will have the document contents modified, so it can regain validity with respect to a schema. The Adaptation component is described in details in the next section.

2.4 Adaptation - Invalid XML Documents Adaptation

The Adaptation component is responsible by adaptation of XML documents deemed as invalid by the Revalidation component due to an evolution process that took place on a XML schema. This component takes as inputs a set of XML documents to adapt, a set of deltas and a XML schema referred to by the document.

The XML document adaptation takes place based on operations described by deltas generated by the Diff component and taken as inputs, thus, these deltas are associated to the schema version taken as a parameter on this component as well. The Adaptation component will issue a query to the Store component, in order to identify whether the given schema has child versions. When child versions are found, successive deltas will be applied to the XML document, until the current schema version deltas are reached and applied.

The difference detection algorithms subject to usage by the Diff component can generate many XML document modification operations other than the basic inclusion and deletion of elements and attributes, such as move and copy of complex elements. Even though not all operations are used on a given moment by the algorithm applied by the Diff component, the Adaptation component must have the ability to understand and deal with all update operations that can be generated by the Diff component.

The move and copy of elements in a XML schema has trivial impact and implementation on XML documents taken as input by the Adaptation component. The adaptation process basically consists on moving an element inside a XML document, or copying an element from one location to another.

Deletion of attributes and elements also has a trivial implementation, based on removal of the corresponding artifact from the XML document. The exception to this trivial implementation is verified when exclusion of control artifacts of the XML Schema takes place, such as exclusion of use, minOccurs and maxOccurs attributes.

In case of exclusion of XML Schema control attributes, the semantics of each attribute must be subject to analysis in order to identify what is the impact on XML documents. Control attributes semantics must also be taken into account by the Revalidation component when validation of sections of documents takes place.

The inclusion operation has some particular and non trivial cases: inclusion of XML Schema control attributes and elements, inclusion of application attributes, inclusion of optional elements and inclusion of required elements. Among these cases, inclusion of optional elements only has trivial implementation on the documents adaptation process, which demands no adaptation action at all. Inclusion of XML schema control elements must be subject of detailed analysis, in order to identify what is the impact on XML documents.

A possible approach to the problem of specification of new values for new elements and attributes of a XML schema is based on end user input. The application user could supply the missing values during the adaptation process of the documents.

Even though this approach will generate valid documents whose content is valid from an application standpoint, some scenarios such as those where XML messages are adapted and exchanged over a network, may prove this approach as unfit due to many aspects. On these scenarios, where the user input is needed on the adaptation process of XML messages exchanged by distributed applications, a request timeout can occur, other than the need of X-Spread installation not only on the network node where the application is executed, but also on client machines, or at least, the modification of client source code would be needed, in order to enable them to interpret some kind of messaging from X-Spread, which would sign the need for user input on the adaptation process of XML messages.

A possible approach to this problem is creation of new structurally valid elements, even though potentially invalid from an application standpoint, given that the values used on attributes and on primitive data types would be default values of each data type, i.e., an empty array of characters, the zero value for integers, and so on.

In order to adopt an homogeneous approach to all possible XML documents adaptation scenarios, both when documents are located at a file system reachable by X-Spread and when XML messages are exchanged over a network, the mechanism can use settings specified by the X-Spread administrator, where a default value is associated to operations described by deltas persisted by the Store component. The mechanism would first perform a detailed analysis of the stored deltas, finding out those that require user input, present them to the X-Spread and ask for specification of values that should be applied to documents whenever these deltas are invoked by the documents adaptation process.

These approaches to the document adaptation process can be configured by the X-Spread administrator in order to achieve the best results considering the different usage scenarios addressed by the mechanism.

3 Related Work

Researches on semistructured database evolution usually address the document evolution, in other words, only part of the database evolution process is modeled. Documents evolution usually is addressed by implementation of temporality [7] [8], versioning [10] [11] or by a mix of these concepts [15].

Schema evolution is addressed in [1] however the impact of schema evolution on existing XML documents is not taken into account. On the other hand, [4] describes a document evolution and adaptation proposal, with an implementation description on [3]. However, this proposal is based on particular schema update tools, what interferes on one of the main XML format features, which is the ease of update of each artifact, which can be performed with simple tools such as text editors with no outstanding features.

Other than that, the proposal described in [4] may demand user input during documents adaptation to a new schema version. This may render this proposal as unfit for scenarios where XML documents are automatically generated by distributed applications.

A DTD-based evolution mechanism is described on [2], where schema evolution is triggered by patterns detected on documents through usage of data mining techniques. However, when a modification on a schema is performed due to patterns discovered on a given number of documents, documents referring to the schema remain unchanged even after schema evolution. Considering that the schema evolution is not propagated to the XML documents and that patterns that led to the schema evolution may not be found on the entirety of XML documents that form the semistructured database, some formerly valid documents can be deemed as invalid after the schema evolution.

A new approach to XML documents adaptation, based on object oriented databases is described in [16]. In this approach DTDs are converted to user data types and the XML documents are inserted into the database. Modifications to the DTD schema are mapped to database modification operations. These database modifications are subject to validations that may accept or reject the modification if the database consistency is not guaranteed after the modification execution.

Another approach where an object oriented database is used to control XML schemata evolution is described in [17]. In this approach, XML documents are loaded into the database only after the DTD is registered into the database management system. Valid schema modifications supported by XEM are automatically propagated into XML documents, while modifications that lead the database into an inconsistent state are rejected.

As these approaches are very similar to traditional databases approach to schema evolution, these mechanisms may not be suitable for all semistructured

data enabled applications. For instance, applications that use XML message to communicate over a network may not count on this mechanism due to performance issues or plain unfeasibility of the attachment of an object oriented database to the current system implementation.

Evolution of semistructured schemata and documents usually is addressed on the literature as different problems, when in fact they are strongly linked. The proposed mechanism aims to combine different techniques in order to approach in an homogeneous and as non-intrusive manner as possible from the end user and semistructured database enabled application standpoints, all the phases that take place during the evolution process of a XML schema.

The unique feature of this work is the approach to different semistructured database distribution scenarios, featuring flexibility on the revalidation and adaptation processes of documents spread over different parts of local and remote file systems and of XML messages generated by applications commnicating over a network.

4 Concluding Remarks

This paper presented X-Spread, a mechanism to propagate XML schema modifications to documents referring to modified schemata. X-Spread has as premises the high performance, ability to abstract the XML documents location subject to revalidation and adaptation and ability to adapt documents without input from the application user nor from X-Spread's administrator.

Acting as an observant system, as described in [18], the X-Spread architecture offers choice flexibility on the algorithm responsible by difference detection between schema versions and on schema versions and deltas physical storage.

Featuring observation of modifications executed on XML schemata, semantic analysis of the executed modifications, abstraction of XML documents location during revalidation and adaptation processes, the X-Spread differences in comparison with existing work such as [1], [4], [2] and [3] are based on the combination of a wide array of different techniques and the approach of different phases of the schema evolution process.

X-Spread takes into account parts of the process usually addressed separately in the literature, allowing for a schema modification followed by propagation of this modification to documents referring to the schema without use of specific tools. With that feature, X-Spread holds on to one of the features of the XML format, which is the ease of artifacts update.

As of right now, the mechanism components are defined and these definitions are subject to improvements, in order to encompass a greater set of possible evolution scenarios of a XML schema. In parallel, after implementation of the Diff component, a study on XML diff algorithms performance and quality is being executed, in order to validate XyDiff choice.

References

1. Coox, S.V.: Axiomatization of the evolution of xml database schema. Program. Comput. Softw. **29**(3) (2003) 140–146
2. Bertino, E., Guerrini, G., Mesiti, M., Tosetto, L.: Evolving a set of dtds according to a dynamic set of xml documents. In: EDBT '02: Proc. of the Worshops XMLDM, MDDE, and YRWS on XML-Based Data Management and Multimedia Engineering-Revised Papers, London, UK, Springer-Verlag (2002) 45–66
3. Mesiti, M., Celle, R., Sorrenti, M.A., Guerrini, G.: X-evolution: A system for xml schema evolution and document adaptation. In Ioannidis, Y.E., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Böhm, K., Kemper, A., Grust, T., Böhm, C., eds.: EDBT. Volume 3896 of LNCS., Springer (2006) 1143–1146
4. Guerrini, G., Mesiti, M., Rossi, D.: Impact of xml schema evolution on valid documents. In: WIDM '05: Proc. of the 7th annual ACM Intl. workshop on Web information and data management, New York, NY, USA, ACM Press (2005) 39–44
5. Peters, L.: Change detection in xml trees: a survey (2004)
6. Cobena, G., Abiteboul, S., Marian, A.: Detecting changes in xml documents. In: ICDE, IEEE Computer Society (2002) 41–52
7. Amagasa, T., Yoshikawa, M., Uemura, S.: A data model for temporal xml documents. In Ibrahim, M.T., Küng, J., Revell, N., eds.: DEXA. Volume 1873 of LNCS., Springer (2000) 334–344
8. Wang, F., Zaniolo, C.: Temporal queries in xml document archives and web warehouses. In: TIME, IEEE Computer Society (2003) 47–55
9. Wong, R.K., Lam, N.: Managing and querying multi-version xml data with update logging. In: ACM Symposium on Document Engineering, ACM (2002) 74–81
10. Iwaihara, M., Chatvichienchai, S., Anutariya, C., Wuwongse, V.: Relevancy based access control of versioned xml documents. In Ferrari, E., Ahn, G.J., eds.: SACMAT, ACM (2005) 85–94
11. Wuwongse, V., Yoshikawa, M., Amagasa, T.: Temporal versioning of xml documents. In Chen, Z., Chen, H., Miao, Q., Fu, Y., Fox, E.A., Lim, E.P., eds.: ICADL. Volume 3334 of LNCS., Springer (2004) 419–428
12. W3C: W3c xml schema (2004) Available at: <http://www.w3.org/XML/Schema>. Last accessed: September 2006.
13. Raghavachari, M., Shmueli, O.: Efficient schema-based revalidation of xml. In Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E., eds.: EDBT. Volume 2992 of LNCS., Springer (2004) 639–657
14. Barbosa, D., Mendelzon, A.O., Libkin, L., Mignet, L., Arenas, M.: Efficient incremental validation of xml documents. In: ICDE, IEEE Computer Society (2004) 671–682
15. Santos, R.G.: Evolução de documentos xml com tempo e versões. Master's thesis, UFRGS, Porto Alegre (2005)
16. Al-Jadir, L., El-Moukaddem, F.: F2/xml: Managing xml document schema evolution. In: ICEIS (1). (2004) 251–258
17. Su, H., Kramer, D., Chen, L., Claypool, K.T., Rundensteiner, E.A.: Xem: Managing the evolution of xml documents. In Aberer, K., Liu, L., eds.: RIDE-DM, IEEE Computer Society (2001) 103–110
18. Dyreson, C.E.: Observing transaction-time semantics with ttxpath. In: WISE '01: Proc. of the Second Intl. Conf. on Web Information Systems Engineering (WISE'01) Vol.1, Washington, DC, USA, IEEE Computer Society (2001) 193