Conceptual XML Schema Evolution
– the CoDEX approach for Design and Redesign

Meike Klettke

University of Greifswald, Germany
meike.klettke@uni-greifswald.de

Abstract. Most available approaches for XML schema evolution specify the evolution steps for an XML schema or a DTD. This article will show that schema evolution can also be realized on a conceptual model. Schema evolution always requires propagating the changes to the XML documents that are already associated to the schema. This article suggests a method for conceptual schema evolution concerning all these subtasks. It is implemented in a tool called CoDEX (*Co*nceptual *D*esign and *E*volution of *X*ML schemas).

1 Introduction

"Software aging will occur in all successful product." (David Parnas, [22]).

Parnas gives two (very different) reasons for software aging. The first is caused by the failure of the product's owners to modify it to meet changing needs; the second is the result of the changes that are made [22].

We have to consider that all data which are represented in XML documents also ages and have to be updated from time to time. Nowadays, lots of applications use XML for storing information. Accordingly, the necessity for updating XML documents will increase in the next years. Thereby, not only the content of the XML documents but also the structure underlies changes. Changes of the content can be realized by using *update languages*. Some XML update languages were already suggested, for instance from Tatarinov, et al. [28] and from Patrick Lehti [13]. Several XML database systems can realize updates on XML documents, for instance Tamino, Galax and Oracle. In January 2006, the W3C also suggested an XML update language [3].

Much more complicated than updates of XML documents are schema changes – this process is called *schema evolution*. Schema evolution means that the schema is modified and the XML documents associated to the schema have to be adapted. We need a method for evolution that is easy-to-handle and ensures schema-validity of the XML documents after the evolution.

There exist some approaches that use a language or transformation rules working on a DTD or on an XML schema. For using it, a user needs knowledge about the evolution language as well as detailed knowledge about the syntax of his XML schema.

Conceptual models are used for designing new schemas. In this article, a method for XML schema evolution is suggested that base on a conceptual model and is implemented as part of a design tool CoDEX (=*Co*nceptual *D*esign and *E*volution of *X*ML schemas). The design steps in a graphical editor are translated to XML schema evolution steps. The schema evolution is realized and the XML documents associated to the schema are revalidated and if necessary updated. The whole process is represented in this article.

The structure of the paper follows this order. First, related work and similar approaches are enumerated. An overview of the CoDEX approach is given in section 3. The conceptual model is introduced in section 4. Section 5 represents the conceptual schema evolution and its subtasks in detail. It is also shown how XML documents are adapted onto the schema changes. Section 6 contains a complete example for the evolution process. The article summarizes with a conclusion and remarks about future work.

2 Related work

Schema evolution on conceptual models. Although the idea of realizing schema evolution on a conceptual model is quite obviously there exist only a few publications about that topic. An overview article of Olivé [20] about information systems formulates the demand for such a method. For database design this approach is suggested from Hick and Hainaut in [7]. In these articles, a schema evolution based on the conceptual level of a database was developed, the changes of the user are propagated to the logical and physical level. This approach was implemented as part of the database design tool DB-MAIN. To the best of my knowledge there is only one article suggesting a similar idea for XML (by Dominguez, Lloret, Rubio, and Zapata [5]). In this article UML is used as conceptual model and the schema evolution and document adaptation are realized by using XSLT programs.

Schema evolution on DTDs or XML schemas. There exist some approaches that can handle an XML schema evolution. These approaches developed languages for describing evolution steps on an XML schema or on a DTD. Kramer and Rundensteiner [27, 11] suggest an XML Evolution Management and develop a language for schema evolution and realize changes on the DTD and XML documents.

Another approach for XML schema evolution on DTDs or XML schemas is developed from Guerrini, Mesiti, and Rossi [6, 18]. They assume the schema as a graph. Labels in the tree can represent three different states: i) a node has to be changed, ii) no changes are needed or iii) changes maybe necessary. This labelled tree is used for an efficient revalidation of the XML documents and for updating the documents.

The incremental validation of XML documents is necessary for schema evolution. There are some publications that concentrate on this task. Incremental schema validation after updates was developed from Milo, Suciu and Vianu [19] and from Papakonstantinou and Vianu [21].

Nowadays, most available XML database systems don't support schema evolution. An exception is the XML storage solution of Oracle 10g that integrates a simple support for XML schema evolution. The user has to input the new schema and an XSLT script that generates new updated XML documents. Tamino supports schema evolution as follows: all associated XML documents have to be schema-valid according to the new schema. So, schema relaxations are possible.

Conceptual models for XML design. There are several suggestions of the conceptual model for designing XML schemas or DTDs. Several approaches introduce extension of the ER model to design DTDs or XML schemas, for instance [15, 17, 23, 25, 12]. Some approaches base on UML class diagrams and add special extensions, for example ([2, 8, 4, 24, 1].

3 Overview

Figure 1 shows the subtasks that are involved in the CoDEX approach. The CoDEX tool bases on a graphical model [26]. The focus during the development of the approach was the evolution of existing schemas, but the CoDEX tool can also be used for the design of new XML schemas.

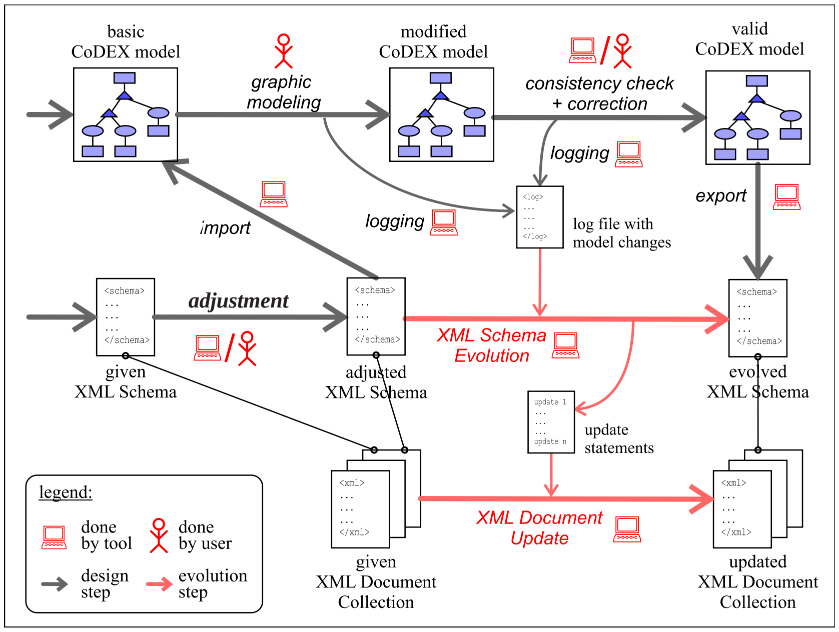

Fig. 1. Subtasks of the CoDEX tool, figure from [26]

For design of new applications the following processes are necessary:

– *graphical modeling*: The user can specify the schema with the conceptual model.

- *consistency check and correction*: Completeness and correctness of the conceptual model is checked.
- *export*: An XML schema according to the conceptual model is generated.

These subtasks are explained in section 4 more detailled.
For schema evolution with the tool the following processes are needed:

- *graphical modeling*: The user can specify his *changes* on the conceptual model.
- *logging*: All design decisions are logged and summarized as far as possible if the same objects had been changed several times.
- *XML schema evolution*: The XML schema is changed according to the schema evolution steps.
- *XML document update*: The associated XML documents are updated according to the XML schema evolution steps.

Schema evolution can be done on an available conceptual model. It is also possible to apply the approach for a given XML schema. In that case, the redesign process starts with a reverse-engineering, consisting of two additional tasks:

- *adjustment of an XML schema*: A given XML schema is "normalized". The schema is translated into the venetian blind design style. All information are presented as global type definitions. This process can be done for each schema. In some cases, (artificial) type names have to be added. These type names do not influence the associated XML documents of a schema.
- *import*: The CoDEX model for the given XML schema and its normalized representation is generated.

All these subtasks are part of the CoDEX tool and are now shortly described.

4 Conceptual model

The basic components of the conceptual model are *elements, types, groups*, and *modules*. The underlying formal model is a graph, the basic components are represented as nodes. We use a *mixed graph* [29] because connections between the basic components can be directed or undirected. Additional information of the components is stored in *properties*, which are simple key-value pairs. For reading and editing properties there exists a table with the corresponding properties for each component.

Figure 2 shows a section of the conceptual model for the data of a wind energy plant (wep) which is used as running example in this article.

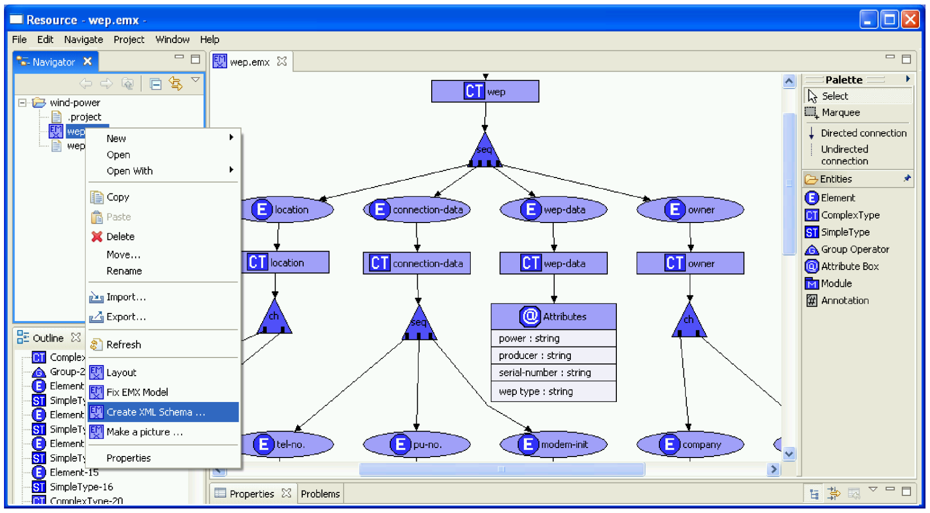

Fig. 2. Conceptual model of a wind energy plant (wep)

Consistency check and corrections on the conceptual model The conceptual models can be checked whether it is complete and correct. Some extension of a design can be added automatically, for instance we can associate default simple types to the elements. If no group entity for the child elements of a complex type is given then a `sequence` is added as default. The automatic extensions are similar to the idea of *model-driven design*.

In other cases, correctness can be tested but no automatic extensions or corrections are possible. For instance, if a group entity is root element in the conceptual model (that is not allowed in XML schema), an algorithm can detect this case but cannot solve it. User interaction is necessary for correction.

Export (translation to XML schema) Conceptual models that are correct and complete can be translated into an XML schema. For the running example in figure 2 the following XML schema is generated:

```
<xs:complexType name="wep" id="cdx_0002">
    <xs:sequence id="cdx_0004">
        <xs:element name="location" type="location" id="cdx_0006"/>
        <xs:element name="connection-data" type="connection-data"
                    id="cdx_0027"/>
        <xs:element name="wep-data" type="wep-data" id="cdx_0115"/>
        <xs:element name="owner" type="owner" id="cdx_0047"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="location" id="cdx_0007">
    <xs:choice id="cdx_0008">
        <xs:element name="coordinates" type="xs:string" id="cdx_0023"/>
        <xs:element name="address" type="address" id="cdx_0063"/>
    </xs:choice>
</xs:complexType> ...
```

The resulting XML schema always follows the *Venetian blind design style* [16] that means all information are defined as global types and element declarations use these type information.

Import (translation of an XML schema into the CoDEX model) The import of available XML schemas into the CoDEX tool is also possible. This import facility is the prerequisite for evolving existing XML document collections with the CoDEX approach.

5 Schema evolution

In this section, we focus on schema evolutions that a user can describe with edit operations on the conceptual model. Figure 3 shows the subtasks of the approach that are related with the schema evolution, these subtasks are described below.

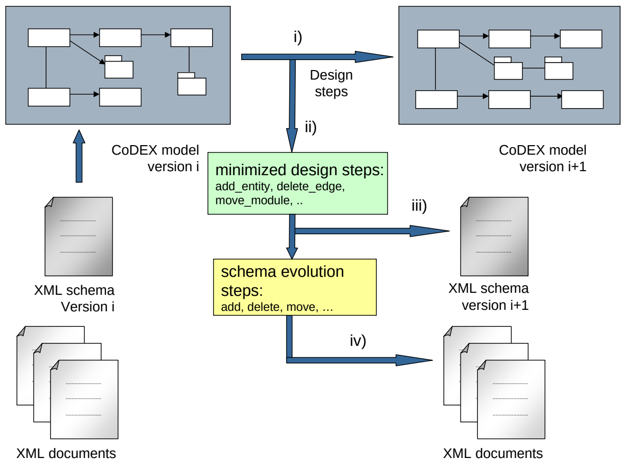

Fig. 3. Overview on the XML schema evolution process in CoDEX

i) Operations on the conceptual model/ Logging component For each component of the conceptual model, the operations `add`, `delete`, `change`, and `move` can occur in an evolution process. For elements and modules also the operation `rename` exists. All design steps of the user are logged in a history component. The logging component entails for each user interaction all information that was changed, so that all information needed for XML schema evolution can be derived from the logfile.

ii) Minimization and normalization of the operations Before the XML schema evolution steps are derived from the design steps a minimization of the number of steps takes place. It is necessary because design is never a straight-forward process. It is often iteratively done, a user frequently changes or cancels design decision during this process.

For instance, if a user creates a new element `wep` and renames it during the design process to `wind-energy-plant` then it has obviously the same meaning as creating a new element with the name `wind-energy-plant`. A rule can be given that summarizes both operations:

$$\text{create\_element(id, name, content)} + \text{rename\_element(id, name, name')} \longrightarrow \\ \text{create\_element(id, name', content)}$$

Other rules remove objects that existed only temporary that menas that had been added and deleted in one design process cycle. In summary, there are 53 rules for combining of evolution steps ([9]). None of the rules is complicated, they all summarize operations concerning the same objects.

iii) XML Schema evolution steps The minimized set of changes on a conceptual model is translated in XML schema evolution steps. The schema evolution steps change the XML schema according to the changes that a user made on a conceptual model. For specifying schema evolution steps an own language was developed [30]. This language bases on an API for accessing XML schemas [14], that supports reading of XML schema components. It was extended so that also modifications of the schema components are allowed. Examples of this language follow in section 6.

iv) XML Document Update If we change an XML schema then it is possible that the XML documents associated to the schema aren't *valid* any longer. Therefore, we have to check and if necessary to update the XML documents. For that, a corresponding update operation for the associated XML documents is executed. The update operations are augmented with all conditions that are tested for revalidation.

In that way, the evolution of XML schema also causes and realizes a document adaptation. The operation for updating the XML documents can be processed with all available XML update languages, for instance with [3]. Examples are shown in section 6.

6 Example

Let us assume the following changes on the running example: Simply speaking we are going to move all attributes from the element `wep-data` to the element `wep`. Second, we delete the now unnecessary element `wep-data`.
These modifications are stored in the log file:

```
<connectionReconnected id="cdx_0040" oldSource="cdx_0010"
                       oldTarget="cdx_0022" source="cdx_0014"
                       target="cdx_0022">
<entityDeleted id="cdx_0010">
```

We can create the following *XML schema evolution steps*:

```
move "/wep/typedefinition::wep-data/@*" into "../typedefinition::wep"
delete select "/wep/wep-data"
```

Based on these statements XML update operations are generated:

```
do insert /wep/wep-data/@* into ..
do delete /wep/wep-data/@*
do delete /wep/wep-data
```

This example shows the complete process from the edit operations in the graphical editor to the schema evolution and XML document update. Although this example is a quite simple evolution step and contains only two operations the method can also be applied for more complex changes.

7 Limits of the conceptual schema evolution

There is only one case where the approach fails: This case shall be explained with an example. The operation `move` on a schema normally entail a move operation in the XML documents. Our running example demonstrated this case.

If we assume another evolution step: we move the `address` from the element `owner` to the element `producer`, we cannot move the values because the `address` of the `owner` is normally not the `address` of the `producer`.

The elements `address` (of `owner` and `producer`) have the same structure but the *values* are different. Such cases cannot automatically be found. This is an open problem: how to detect the cases in which the semantics of the structurally identical components changes by realizing a `move` operation because of the context information.

This problem is not specific for the CoDEX approach but it occurs in all schema evolution approaches. The only reasonable way is user interaction for solving this problem. A possible solution could be two offer two different edit operations: `move` with data and `move` without the associated data.

8 Summary

XML Schema evolution is one of the future research fields. All applications that are used over longer periods of time sometimes have to be evolved and adapted onto new requirements.

With the CoDEX tool, schema evolution can be realized with the same conceptual model that is used for designing new applications. It is not necessary that a user specifies the schema evolution steps directly on the XML schema syntax. During editing the conceptual model he can specify the changes, so that we achieve an easy-understandable, user-friendly approach for maintenance of XML applications. Based on the changes the *XML documents* associated to the schema are updated, too.

This method can realize evolution steps that modify elements or attributes. It is not possible to modify parts of the XML documents with higher granularity. For instance, we cannot split an element content into two elements or add the values of two attributes to a new attribute value. Edit operations on a conceptual model cannot describe such changes. If such evolution steps are necessary, than additionally other tools have to be employed, for instance several *mapping tools* offer this functionality. They support manual customization on the level of XML documents.

Mapping and matching approaches derive differences between the new and the old schema. Based on these differences, updates for the XML documents can be realized. Matching approaches use heuristics for determining similarities and accordingly the results can be incorrect. During schema evolution it is possible to log the operation of a user, this information is more reliable for adapting the schema and the XML documents. That is the reason why this article suggests an *extension of a design method* for evolution instead of automatic matching approaches.

The CoDEX model editor is implemented in Java as plug-in for the open-source IDE Eclipse. Different Eclipse concepts like properties, preferences, problem markers are used. The schema evolution and document update are implemented on the basis of `eXist` and base on the update language offered in that system. In future, the implementation shall be realized with the update language of the W3C. For that the update statements are additionally generated in this language yet.

9 Future work

A future plan is the integration of a function that calculates the effort of each evolution step before it is realized. This function shall determine which parts of the schema are concerned and how many XML documents will be changed by an evolution step.

All parts described in this article are implemented but till now not completely integrated. That is also one of the future tasks on that field.

10 Acknowledgement

I want to thank my students Robert Stephan, Tobias Tiedt, Marcus Oertel, Christian Will, and Maike Milling for their work that they investigated with their diploma or study theses on this topic very much.

References

[1] M. Bernauer, G. Kappel, and G. Kramler. Representing XML Schema in UML - A Comparison of Approaches. In N. Koch, P. Fraternali, and M. Wirsing, editors, *ICWE*, volume 3140 of *Lecture Notes in Computer Science*, pages 440–444. Springer, 2004.

[2] D. Carlson. *Modeling XML Applications with UML: Practical e-Business Applications.* Addison-Wesley Object Technology Series, 2001.

[3] D. Chamberlin, D. Florescu, and J. Robie. XQuery Update Facility, 2006. http://www.w3.org/TR/xqupdate/.

[4] R. Conrad, D. Scheffner, and J.-C. Freytag. XML Conceptual Modelling using UML. In A. H. F. Laender, S. W. Liddle, and V. C. Storey, editors, *Proceedings of the 19th International Conference on Conceptual Modeling, ER*, Lecture Notes in Computer Science 1920. Springer, 2000.

[5] E. Dominguez, J. Lloret, A. L. Rubio, and M. A. Zapata. Evolving XML Schemas and Documents Using UML Class Diagrams. In *Database and Expert Systems Applications: 16th International Conference, DEXA, Lecture Notes in Computer Science*, volume 3588, pages 343–352. Springer Berlin / Heidelberg, 2005.

[6] G. Guerrini, M. Mesiti, and D. Rossi. Impact of XML schema evolution on valid documents. In A. Bonifati and D. Lee, editors, *WIDM*, pages 39–44. ACM, 2005.

[7] J.-M. Hick and J.-L. Hainaut. Strategy for Database Application Evolution: The DB-MAIN Approach. In *Conceptual Modeling - ER*, volume 2813 of *Lecture Notes in Computer Science*, pages 291 – 306. Springer, 2003.

[8] G. Kappel, E. Kapsammer, S. Rausch-Schott, and W. Retschitzegger. X-Ray — Towards Integrating XML and Relational Database Systems. In A. H. F. Laender, S. W. Liddle, and V. C. Storey, editors, *Proceedings of the 19th International Conference on Conceptual Modeling, ER*, Lecture Notes in Computer Science 1920, pages 339–353. Springer, 2000.

[9] M. Klettke. Modellierung, Bewertung und Evolution von XML-Dokumentkollektionen, 2006. eingereicht an der Universität Rostock, Fakultät für Informatik und Elektrotechnik.

[10] M. Klettke, H. Meyer, and B. Hänsel. Evolution — The Other Side of the XML Update Coin. In *2nd International Workshop on XML Schema and Data Management (XSDM), in conjunction with ICDE*, 2005.

[11] D. Kramer. XEM: XML Evolution Management. Master's thesis, Worchester Polytechnic Institute, 2001.

[12] M.-L. Lee, S. Y. Lee, T. W. Ling, G. Dobbie, and L. A. Kalinichenko. Designing semistructured databases: A conceptual approach. In H. C. Mayr, J. Lazanský, G. Quirchmayr, and P. Vogel, editors, *DEXA*, volume 2113 of *Lecture Notes in Computer Science*, pages 12–21. Springer, 2001.

[13] P. Lehti. Design and Implementation of a Data Manipulation Processore for an XML Query Language. Diplomarbeit, Technische Universität Darmstadt, Fachbereich Elektrotechnik und Informationstechnik, 2001.

[14] E. Litani. XML Schema API, 2004. http://www.w3.org/Submission/xmlschema-api/.

[15] B. F. Lóscio, A. C. Salgado, and L. do Rêgo Galvão. Conceptual modeling of XML schemas. In R. H. L. Chiang, A. H. F. Laender, and E.-P. Lim, editors, *WIDM*, pages 102–105. ACM, 2003.

[16] E. Maler. Schema Design Rules for UBL ... and Maybe for You. In *XML conference and exposition*, 2002.

http://www.idealliance.org/papers/xml02/dx_xml02/papers/05-01-02/05-01-02.html.

[17] M. Mani. EReX: A Conceptual Model for XML. In Z. Bellahséne, T. Milo, M. Rys, D. Suciu, and R. Unland, editors, *Database and XML Technologies: Second International XML Database Symposium, XSym*, volume 3186, pages 128–142. Springer-Verlag, 2004.

[18] M. Mesiti, R. Celle, M. A. Sorrenti, and G. Guerrini. X-Evolution: A System for XML Schema Evolution and Document Adaptation. In *International Conference on Extending Database Theory (EDBT), Demonstration*, 2006.

[19] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML Transformers. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 11–22, Dallas, Texas, USA, 2000. ACM.

[20] A. Olivé. Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research. In O. Pastor and J. F. e Cunha, editors, *CAiSE*, volume 3520 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2005.

[21] Y. Papakonstantinou and V. Vianu. Incremental Validation of XML Documents. In D. Calvanese, M. Lenzerini, and R. Motwani, editors, *Proceedings of the International Conference on Database Theory (ICDT)*, volume 2572 of *Lecture Notes in Computer Science*, pages 47–63, Siena, Italy, 2002. Springer-Verlag.

[22] D. L. Parnas. Software Aging. In *ICSE: Proceedings of the 16th international conference on Software engineering*, pages 279–287, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[23] K. Passi, L. Lane, S. K. Madria, B. C. Sakamuri, M. K. Mohania, and S. S. Bhowmick. A Model for XML Schema Integration. In K. Bauknecht, A. M. Tjoa, and G. Quirchmayr, editors, *EC-Web*, volume 2455 of *Lecture Notes in Computer Science*, pages 193–202. Springer, 2002.

[24] N. Routledge, L. Bird, and A. Goodchild. UML and XML-Schema. In *Thirteenth Australasian Database Conference*, 2002.

[25] A. Sengupta, S. Mohan, and R. Doshi. Extensible Entity Relationship Modeling. In *XML*, 2003. www.idealliance.org/papers/dx_xml03/papers/06-01-01/06-01-01.html.

[26] R. Stephan. Entwicklung und Implementierung einer Methode zum konzeptuellen Entwurf von XML-Schemata. Diplomarbeit, Universität Rostock, Institut für Informatik, 2006.

[27] H. Su, D. K. Kramer, and E. A. Rundensteiner. XEM: XML Evolution Management. Computer Science Technical Report Series, Worchester Polytechnic Institute, WPI-CS-TR-02-09, Jan. 2002.

[28] I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. Updating XML. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2001.

[29] E. Wanke and R. Kötter. Oriented Paths in Mixed Graphs. In R. Fleischer and G. Trippen, editors, *ISAAC*, volume 3341 of *Lecture Notes in Computer Science*, pages 629–643. Springer, 2004.

[30] C. Will. Entwicklung und Implementierung einer Sprache zur Evolution von XML-Schemata. Diplomarbeit, Universität Rostock, Institut für Informatik, 2006.