A Conceptual Modeling Approach to XML Schema Evolution

Meike Klettke

Database Research Group, University of Rostock, Germany
meike@informatik.uni-rostock.de

Abstract. Conceptual design methods are used in many fields of computer science. Most methods support the design of new applications. The evolution of existing applications is similar to the design task. A unique model, method and tool that support both tasks is desirable.
In this article, such a method is suggested for the design and the evolution of XML schemas. The article focusses on the evolution and introduces the subtasks that are necessary for enhancing XML schemas. It will show that schema evolution can be realized on a conceptual model. Schema evolution always requires the propagation of changes to XML schema and also to XML documents associated to the schema.
The approach is implemented in a tool called CoDEX (*Co*nceptual *D*esign and *E*volution of *X*ML schemas).

1 Introduction

Nearly every field of computer science employs models and modelling methods for the design of new information. In database design, a conceptual model is used for defining new databases. In software development, the widely used Unified Modeling Language (UML) plays this role. Some suggestions for models that can be helpful for defining schemas for XML documents are also available.

1.1 Evolution of existing applications

All information (databases, software, XML schemas) is used over very long periods of time. That's why, it is necessary to update or change the information from time to time. Two main reasons can be given why changes are necessary. First, errors of the software products cause these changes, secondly the environment (for instance other applications, interfaces, processes or regulations) are changed and hence adaptations are necessary. In [25], Parnas enumerates reasons for software aging and illustrates with several examples the need for evolution.

In software development, the well-known waterfall model was replaced by the spiral model that represents the continuous process of development in a better way. Figure 1 shows this model. Obviously, the same phases in the software life cycle are passed through several times.

As shown in this model, design is a continuous process, tools are needed that can be used for the design of new applications as well as for the evolution of

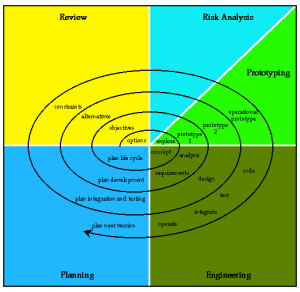

Fig. 1. Spiral model, suggested by Boehm [2], figure from [20]

available solutions. There exist several database design tools, that can generate a conceptual model, for instance an Entity Relationship Model from existing databases – a so called *reverse engineering process*. These conceptual models can be enhanced and translated into a new database. If we choose such a method, we get a new database schema, but the data of the old database is not taken over into the new one. The database evolution can be described with `alter table` statements that are much more difficult to specify than Entity Relationship Models.

We can consider the gap between the design of a new application versus the evolution as is shown in figure 2.

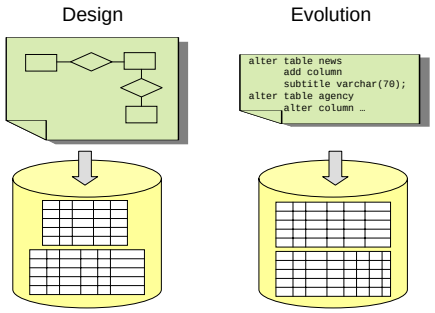

Fig. 2. Comparision of the design and evolution process

1.2 Defining evolution with a conceptual model

To overcome the gap mentioned above, another approach has to be supported by tools, the *propagation of changes* between the model in the version i and the

model in the version *i+1*. The formal description of model changes can be used for generating the `alter table` statements to adapt the database. Although this idea is not widely accepted in the field of databases. it has already been suggested in some publications, for instance by Hick and Hainauth in [9] and by Dominguez, Lloret, Rubio, and Zapata in [6]. Nevertheless, until now most developer use conceptual models for designing new databases and adapt the databases on a more detailed level by writing `alter table` statements manually.

But, evolution based on conceptual models has several advantages, simply speaking all advantages that the conceptual models have for designing new databases also are relevant for the evolution process.

Based on a conceptual model, technical details can be hidden so that a user can concentrate on important objects and relationships. The model is capable to discuss changes with application experts. This article concentrates on the XML schema evolution based on a conceptual model but this approach can also be applied to other redesign methods.

If evolution is specified on a conceptual mode, we can guarantee that the model and the actual software product correspond.

1.3 XML schema evolution

We have to consider that all data which is represented in XML documents also ages and has to be updated from time to time. Nowadays, lots of applications use XML for information storage. Accordingly, the necessity for updating XML documents will increase in the next years. Doing so, 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 by Tatarinov, et al. [31] and by Patrick Lehti [15]. 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 *XQuery Update* [4].

Much more complicated than updates of XML documents are schema changes – this process is called *schema evolution*. Doing schema evolution, 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 either use a language or transformation rules working on a DTD or on an XML schema. For defining this transformation rules, a user needs to have 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 is based on a conceptual model and is implemented as part of a design tool called CoDEX (=*Co*nceptual *D*esign and *E*volution of *X*ML schemas). First, the design steps in a graphical editor are translated to XML schema evolution steps. Then, 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.

This paper is structured as follows: 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 dealing with that topic. An overview article by Olivé [23] about information systems formulates the demand for such a method. For database design this approach is suggested by Hick and Hainaut in [8] and [9]. In both mentioned articles, a schema evolution based on the conceptual level of a database was developed and 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 [6]) which uses UML as conceptual model and realizes the schema evolution and document adaptation by using XSLT programs.

Schema evolution on DTDs or XML schemas. There exist some approaches that can handle XML schema evolution. These approaches developed languages for describing evolution steps on an XML schema or on a DTD. Kramer and Rundensteiner [30, 13] 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 was developed by Guerrini, Mesiti, and Rossi [7, 21]. They assume the schema to be 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 re-validation 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 by Milo, Suciu and Vianu [22] and by Papakonstantinou and Vianu [24].

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 provide the new schema and an XSLT script that generates new updated XML documents.

Tamino, being another XML storage solution, supports schema evolution as follows: all associated XML documents have to be schema-valid according to the new schema. Doing so, schema relaxations are possible.

Conceptual models for XML design. There are several suggestions for conceptual models for designing XML schemas or DTDs. Several approaches introduce extensions of the Entity Relationship Model in order to design DTDs or XML schemas, for instance [17, 19, 26, 28, 14]. Other models base on UML class diagrams and add special extensions, for example ([3, 10, 5, 27, 1].

3 Overview

Figure 3 shows the subtasks that are involved in the CoDEX approach (*Co*nceptual *D*esign and *E*volution of *X*ML schemas). The CoDEX tool is based on a graphical model [29]. 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.

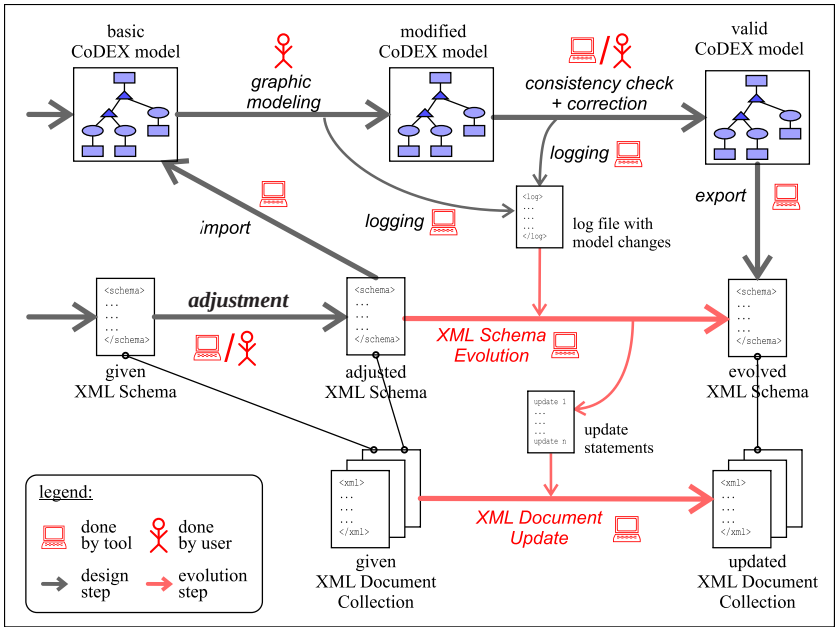

Fig. 3. Subtasks of the CoDEX tool, figure from [29]

For the *design of new applications* the following processes are necessary:

– *graphical modeling*: The user can specify the schema using 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 will be explained in section 4 in more detail.
For *schema evolution* with CoDEX 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 much 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 to a given XML schema. In that case, the re-design 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 [18]. 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 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* [32] 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 4 shows a section of the conceptual model for the data of a *news agency* which is used as running example in this article.

Consistency check and corrections on the conceptual model. The conceptual models can be checked for completeness and correctness. Some extensions 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

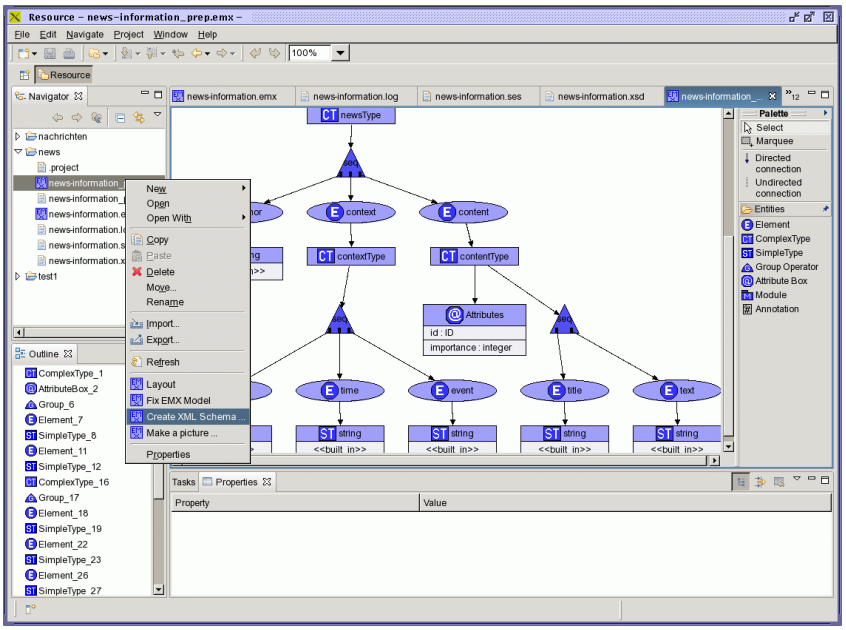

Fig. 4. Conceptual model for describing *news*

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 it is not possible to automatically add extensions. For instance, if a group entity is root element in the conceptual model (which 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 4 the following XML schema is generated:

```
<?xml version="1.0" encoding="UTF-8"?>
    ...
   <xs:complexType id="cdx_0048" name="newsType">
        <xs:sequence id="cdx_0047">
            <xs:element id="cdx_0022" name="author" type="xs:string"/>
            <xs:element id="cdx_0029" name="context" type="contextType"/>
            <xs:element id="cdx_0037" name="content" type="contentType"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType id="cdx_0058" name="contextType">
```

```
    <xs:sequence id="cdx_0057">
        <xs:element id="cdx_0030" name="region" type="xs:string"/>
        <xs:element id="cdx_0031" name="time" type="xs:string"/>
        <xs:element id="cdx_0035" name="event" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType id="cdx_0068" name="contentType">
    <xs:sequence id="cdx_0067">
        <xs:element id="cdx_0023" name="title" type="xs:string"/>
        <xs:element id="cdx_0024" name="text" type="xs:string"/>
    </xs:sequence>
    <xs:attribute id="cdx_0040" name="id" type="xs:ID"/>
    <xs:attribute id="cdx_0041" name="importance" type="xs:integer"/>
</xs:complexType>
...
```

The resulting XML schema always follows the *Venetian blind design style* [18] 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, the XML schema has to be *valid* then an import can be realized. 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 5 shows the subtasks of the approach that are related to schema evolution, these subtasks are described below.

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, the operation `rename` exists as well. All design steps of the user are logged in a history component. The logging component stores all information that was changed, so that all information needed for the 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. This is necessary because design is never a straight-forward process. It is often done iteratively, a user frequently changes or cancels design decisions 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

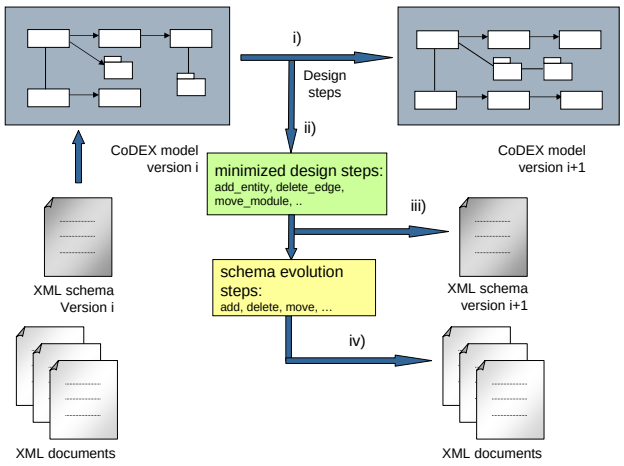

Fig. 5. Overview on the XML schema evolution process in CoDEX

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 temporarely that means that they had been added and deleted in one design process cycle. In summary, there are 53 rules for the combination of evolution steps ([11]). 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.

There exists till now no XML schema evolution language, thus it was necessary to develop such a language. The aim was to be conform with available W3C recommondations. The language is orientated on the XML schema API suggested by Litani [16] and on the XQuery Update language, edited by Chamberlin, Florescu, and Robie [4] and uses XPath expressions for addressing schema components. The following examples deliver an impression of this language:

```
// add a new (optional) attribute
insert attribute '$attributename' of type 'xs:string'
       into //$elementname;
set use='optional' of //$elementname/@$attributename;

// change an element declaration
rename //$elementname as '$new-elementname';
```

```
set maxOccurs='3' of /$new-elementname;
set minOccurs='1' of /$new-elementname;

// delete an element
delete //$elementname;
```

The complete schema evolution language can be found in [33]. Further examples of this language follow in section 6.

The insert of new components in a conceptual model can be realized quite simple, by generating `insert` operations with the evolution language. In the same way `changes` of the `facets` in the conceptual model can be handled. The `delete` operations can also directly be translated into schema evolution operations.

More complicated than changes of `nodes` of the conceptual model are all changes of the *edges*. If a directed connection in the conceptual model is deleted, for instance a connection between an `element` and a `simple type` than the simple type is obtained as global type definitions, the type association in the element declaration is removed.

The contrary operation, the adding of edges causes an association of `types` to an `element` or `attribute`.

In that way, each of the user actions on the conceptual model is translated into XML schema evolution steps. Accordingly to the evolution steps the XML schema file is adapted.

iv) XML Document Update. If we change an XML schema it is possible that the XML documents associated to the schema aren't *valid* any longer. That's why, 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 re-validation.

In that way, the evolution of XML schema also causes and realizes a document adaptation. The schema changes are propagated to the XML documents as well. The operation for updating the XML documents can be processed with all available XML update languages, for instance with [4]. Examples are shown in section 6.

6 Example

The evolution process is demonstrated with the running example, now. Figure 6 show the model for describing news.

Let us assume the following changes on the running example: Simply speaking we are going to move all attributes from the type `contentType` to the type `newsType`. Second, we rename the element `title` into `headline` and third, we insert a new element `subtitle` as child node of `content`.
These modifications are stored in the log file:

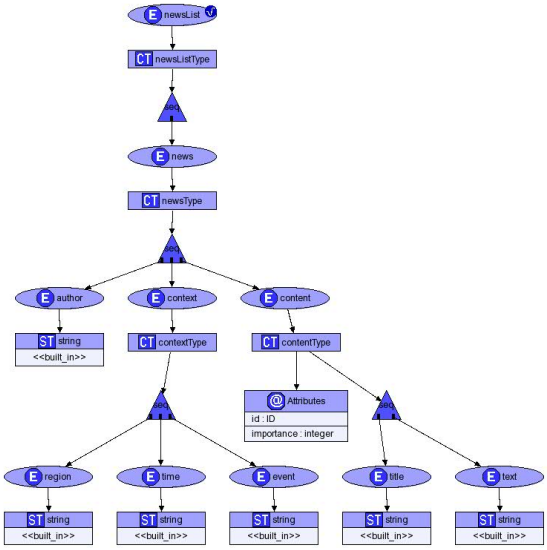

Fig. 6. CoDEX model for describing news

```
// 1. operation: move of attribute
<connectionReconnected ID="cdx_0042" oldSource="cdx_0037"
    oldTarget="cdx_0039" source="cdx_0048" target="cdx_0039"/>

// 2. operation: rename of an element
<propertyChanged key="EmxElement.Name" newValue="headline"
    objectID="cdx_0023" oldValue="title"/>

// 3. operation: insert of a new element and connect it
<entityCreated ID="cdx_0071" bounds="Rectangle(539, 669, 144, 39)">
    <element maxOccurs="1" minOccurs="1" name="Element_71"/>
<propertyChanged key="EmxElement.Name" newValue="subtitle"
    objectID="cdx_0071" oldValue="Element_71"/>
<connectionCreated ID="cdx_0072" source="cdx_0067" sourceDockPos="1"
    style="directed" target="cdx_0071"/>
```

This logfile is normally not visible for a designer, for that not easy to read. The comments are added in log file fragment for the reason of readability.

The information inthe logfile is the basis for the evolution process. We can create the following *XML schema evolution steps* for the model changes:

```
1.   move /typedefinition::contentType/@*
          into /typedefinition::newsType;
2.   rename //title as 'headline';
3.   insert element subtitle of type "xs:string" after //headline;
```

Based on these statements XML update operations are generated:

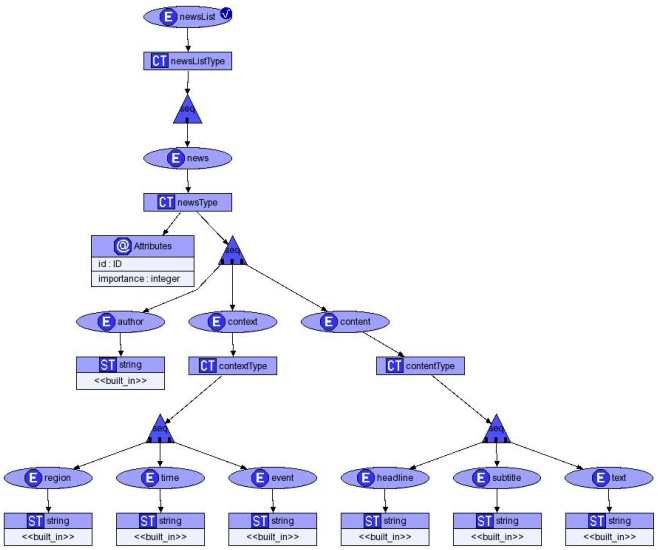

Fig. 7. Modified CoDEX model for describing news

```
1.   do insert //content/@* into ..
     do delete //content/@*
2.   do rename //title as 'headline';
3.   do insert <subtitle/> after //headline
```

These update operations can be applied onto the XML documents associated to the schema.
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 three operations the method can also be applied for more complex changes.

7 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 to 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 a user can specify the changes, so that an easy-understandable, user-friendly approach for maintenance of XML applications is achieved. Based on the changes the *XML documents* associated to the schema are updated, too.

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. 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 a 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 are based 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.

8 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.

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, additional other tools have to be employed, for instance several *mapping tools* offer this component. They support manual customization on the level of XML documents in a comfortable way. The enrichment of the evolution tool with such additional functionality is another future task.

9 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] W. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, pages 61–72, May 1988.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] J.-L. Hainaut, V. Englebert, J. Henrard, J.-M. Hick, and D. Roland. Database Evolution: the DB-Main Approach. In P. Loucopoulos, editor, *Entity-Relationship Approach - ER'94, Business Modelling and Re-Engineering, 13th International Conference on the Entity-Relationship Approach, Manchester, U.K., December 13-16, 1994, Proceedings*, volume 881 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 1994.

[9] 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.

[10] 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.

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

[12] 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.

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

[14] 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.

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

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

[17] 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.

[18] 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.

[19] 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.

[20] J. McCormack. Cse2305 object-oriented software engineering, 2005. http://www.csse.monash.edu.au/ jonmc/CSE2305/.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

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

[30] 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.

[31] 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.

[32] 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.

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