A Conceptual Model for the XML Schema Evolution

Overview: Storing, Base-Model-Mapping and Visualization

Thomas Nösinger, Meike Klettke, Andreas Heuer
Database Research Group
University of Rostock, Germany
(tn, meike, ah)@informatik.uni-rostock.de

ABSTRACT

In this article the conceptual model EMX (Entity Model for XML-Schema) for dealing with the evolution of XML Schema (XSD) is introduced. The model is a simplified representation of an XSD, which hides the complexity of XSD and offers a graphical presentation. For this purpose a unique mapping is necessary which is presented as well as further information about the visualization and the logical structure. A small example illustrates the relationships between an XSD and an EMX. Finally, the integration into a developed research prototype for dealing with the co-evolution of corresponding XML documents is presented.

1. INTRODUCTION

The eXtensible Markup Language (XML) [2] is one of the most popular formats for exchanging and storing structured and semi-structured information in heterogeneous environments. To assure that well-defined XML documents can be understood by every participant (e.g. user or application) it is necessary to introduce a document description, which contains information about allowed structures, constraints, data types and so on. XML Schema [4] is one commonly used standard for dealing with this problem. An XML document is called valid, if it fulfills all restrictions and conditions of an associated XML Schema.

XML Schema that have been used for years have to be modified from time to time. The main reason is that the requirements for exchanged information can change. To meet these requirements the schema has to be adapted, for example if additional elements are added into an existing content model, the data type of information changed or integrity constraints are introduced. All in all every possible structure of an XML Schema definition (XSD) can be changed. A question occurs: In which way can somebody make these adaptions without being coerced to understand and deal with the whole complexity of an XSD? One solution is the definition of a conceptual model for simplifying the base-model; in this paper we outline further details of our conceptual model called EMX (Entity Model for XML-Schema).

A further issue, not covered in this paper, but important in the overall context of exchanging information, is the validity of XML documents [5]. Modifications of XML Schema require adaptions of all XML documents that are valid against the former XML Schema (also known as co-evolution).

One unpractical way to handle this problem is to introduce different versions of an XML Schema, but in this case all versions have to be stored and every participant of the heterogeneous environment has to understand all different document descriptions. An alternative solution is the evolution of the XML Schema, so that just one document description exists at one time. The above mentioned validity problem of XML documents is not solved, but with the standardized description of the adaptions (e.g. a sequence of operations [8]) and by knowing a conceptual model inclusively the corresponding mapping to the base-model (e.g. XSD), it is possible to derive necessary XML document transformation steps automatically out of the adaptions [7]. The conceptual model is an essential prerequisite for the here not in detail but incidentally handled process of the evolution of XML Schema.

This paper is organized as follows. **Section 2** gives the necessary background of XML Schema and corresponding concepts. **Section 3** presents our conceptual model by first giving a formal definition (**3.1**), followed by the specification of the unique mapping between EMX and XSD (**3.2**) and the logical structure of the EMX (**3.3**). After introducing the conceptual model we present an example of an EMX in **section 4**. In **section 5** we describe the practical use of EMX in our prototype, which was developed for handle the co-evolution of XML Schema and XML documents. Finally in **section 6** we draw our conclusions.

2. TECHNICAL BACKGROUND

In this section we present a common notation used in the rest of the paper. At first we will shortly introduce the *abstract data model* (ADM) and *element information item* (EII) of XML Schema, before further details concerning different modeling styles are given.

The XML Schema *abstract data model* consists of different components or node types[1], basically these are: type definition components (simple and complex types), declaration components (elements and attributes), model group components, constraint components, group definition components

[1]An XML Schema can be visualized as a directed graph with different nodes (components); an edge realizes the hierarchy

25^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Ilmenau, Germany.

and annotation components [3]. Additionally the *element information item* exists, an XML representation of these components. The *element information item* defines which content and attributes can be used in an XML Schema. Table 1 gives an overview about the most important components and their concrete representation. The <include>,

ADM	Element Information Item
declarations	<element>, <attribute>
group-definitions	<attributeGroup>
model-groups	<all>, <choice>, <sequence>, <any>, <anyAttribute>
type-definitions	<simpleType>, <complexType>
N.N.	<include>, <import>, <redefine>, <overwrite>
annotations	<annotation>
constraints	<key>, <unique>, <keyref>, <assert>, <assertion>
N.N.	<schema>

Table 1: XML Schema Information Items

<import>, <redefine> and <overwrite> are not explicitly given in the *abstract data model* (N.N. - Not Named), but they are important components for embedding externally defined XML Schema (esp. element declarations, attribute declarations and type definitions). In the rest of the paper, we will summarize them under the node type "module". The <schema> "is the document (root) element of any W3C XML Schema. It's both a container for all the declarations and definitions of the schema and a place holder for a number of default values expressed as attributes" [9]. Analyzing the possibilities of specifying declarations and definitions leads to four different modeling styles of XML Schema, these are: *Russian Doll*, *Salami Slice*, *Venetian Blind* and *Garden of Eden* [6]. These modeling styles influence mainly the re-usability of element declarations or defined data types and also the flexibility of an XML Schema in general. Figure 1 summarizes the modeling styles with their scopes. The

	Scope	Russian Doll	Salami Slice	Venetian Blind	Garden of Eden
element and attribute declaration	local	x		x	
	global		x		x
type definition	local	x	x		
	global			x	x

Figure 1: XSD Modeling Styles according to [6]

scope of element and attribute declarations as well as the scope of type definitions is global iff the corresponding node is specified as a child of the <schema> and can be referenced (by knowing e.g. the name and namespace). Locally specified nodes are in contrast not directly under <schema>, the re-usability is not given respectively not possible.

An XML Schema in the *Garden of Eden* style just contains global declarations and definitions. If the requirements against exchanged information change and the underlying schema has to be adapted then this modeling style is the most suitable. The advantage of the *Garden of Eden* style is that all components can be easily identified by knowing the *QNAME* (qualified name). Furthermore the position of components within an XML Schema is obvious. A qualified name is a colon separated string of the target namespace of the XML Schema followed by the name of the declaration or definition. The name of a declaration and definition is a string of the data type *NCNAME* (non-colonized name), a string without colons. The *Garden of Eden* style is the basic modeling style which is considered in this paper, a transformation between different styles is possible.[2]

3. CONCEPTUAL MODEL

In [7] the three layer architecture for dealing with XML Schema adaptions (i.e. the XML Schema evolution) was introduced and the correlations between them were mentioned. An overview is illustrated in figure 2. The first

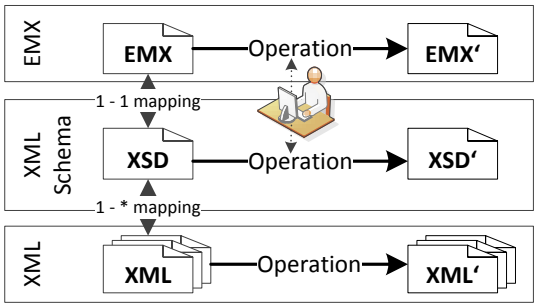

Figure 2: Three Layer Architecture

layer is our conceptual model EMX (Entity Model for XML-Schema), a simplified representation of the second layer. This layer is the XML Schema (XSD), where a unique mapping between these layers exists. The mapping is one of the main aspects of this paper (see section 3.2). The third layer are XML documents or instances, an ambiguous mapping between XSD and XML documents exists. It is ambiguous because of the optionality of structures (e.g. minOccurs = '0'; use = 'optional') or content types (e.g. <choice>). The third layer and the mapping between layer two and three, as well as the operations for transforming the different layers are not covered in this paper (parts were published in [7]).

3.1 Formal Definition

The conceptual model EMX is a triplet of nodes (N_M), directed edges between nodes (E_M) and features (F_M).

$$EMX = (N_M, E_M, F_M) \quad (1)$$

Nodes are separated in simple types (st), complex types (ct), *elements*, *attribute-groups*, *groups* (e.g. content model), *annotations*, *constraints* and *modules* (i.e. externally managed XML Schemas). Every node has under consideration of the *element information item* of a corresponding XSD different attributes, e.g. an *element* node has a name, occurrence values, type information, etc. One of the most important

[2]A student thesis to address the issue of converting different modeling styles into each other is in progress at our professorship; this is not covered in this paper.

attributes of every node is the *EID* (EMX ID), a unique identification value for referencing and localization of every node; an *EID* is one-to-one in every EMX. The directed edges are defined between nodes by using the *EID*s, i.e. every edge is a pair of *EID* values from a source to a target. The direction defines the include property, which was specified under consideration of the possibilities of an XML Schema. For example if a *model-group* of the *abstract data model* (i.e. an EMX *group* with "EID = 1") contains different *elements* (e.g. EID = {2,3}), then two edges exist: (1,2) and (1,3). In section 3.3 further details about allowed edges are specified (see also figure 5). The additional features allow the user-specific setting of the overall process of co-evolution. It is not only possible to specify default values but also to configure the general behaviour of operations (e.g. only capacity-preserving operations are allowed). Furthermore all XML Schema properties of the *element information item* <schema> are included in the additional features. The additional features are not covered in this paper.

3.2 Mapping between XSD and EMX

An overview about the components of an XSD has been given in table 1. In the following section the unique mapping between these XSD components and the EMX nodes introduced in section 3.1 is specified. Table 2 summarizes the mapping. For every *element information item* (EII) the

EII	EMX Node	Visualization
<element>	*element*	
<attribute>, <attributeGroup>	*attribute-group*	
<all>, <choice>, <sequence> <any> <anyAttribute>	*group*	
<simpleType>	*st*	implicit and specifiable
<complexType>	*ct*	implicit and derived
<include>, <import>, <redefine>, <overwrite>	*module*	
<annotation>	*annotation*	
<key>, <unique>, <keyref> <assert> <assertion>	*constraint*	implicit in *ct* restriction in *st*
<schema>	the EMX itself	

Table 2: Mapping and Visualization

corresponding EMX node is given as well as the assigned visualization. For example an EMX node *group* represents the *abstract data model* (ADM) node *model-group* (see table 1). This ADM node is visualized through the EII content models <all>, <choice> and <sequence>, and the wildcards <any> and <anyAttribute>. In an EMX the visualization of a *group* is the blue "triangle with a G" in it. Furthermore if a *group* contains an element wildcard then this is visualized by adding a "blue W in a circle", a similar behaviour takes place if an attribute wildcard is given in an <attributeGroup>.

The *type-definitions* are not directly visualized in an EMX. Simple types for example can be specified and afterwards be referenced by *elements* or *attributes*[3] by using the *EID* of the corresponding EMX node. The complex type is also implicitly given, the type will be automatically derived from the structure of the EMX after finishing the modeling process. The XML Schema specification 1.1 has introduced different logical constraints, which are also integrated in the EMX. These are the EIIs <assert> (for constraints on complex types) and <assertion>. An <assertion> is under consideration of the specification a facet of a restricted simple type [4]. The last EII is <schema>, this "root" is an EMX itself. This is also the reason why further information or properties of an XML Schema are stored in the additional features as mentioned above.

3.3 Logical Structure

After introducing the conceptual model and specifying the mapping between an EMX and XSD, in the following section details about the logical structure (i.e. the storing model) are given. Also details about the valid edges of an EMX are illustrated. Figure 3 gives an overview about the different relations used as well as the relationships between them. The logical structure is the direct consequence of the used

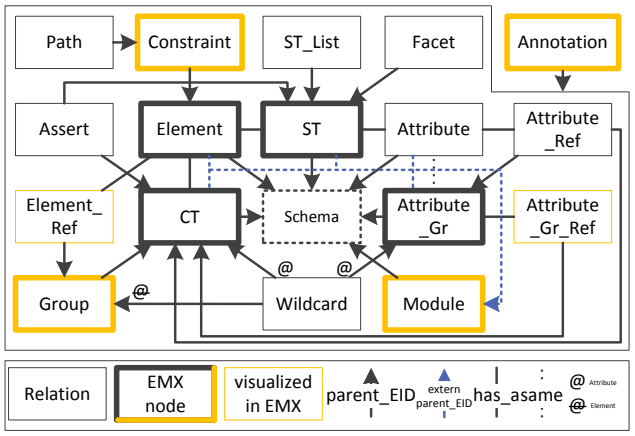

Figure 3: Logical Structure of an EMX

modeling style *Garden of Eden*, e.g. *elements* are either element declarations or element references. That's why this separation is also made in the EMX.

All in all there are 18 relations, which store the content of an XML Schema and form the base of an EMX. The different nodes reference each other by using the well known foreign key constraints of relational databases. This is expressed by using the directed "parent_EID" arrows, e.g. the EMX nodes ("rectangle with thick line") *element*, *st*, *ct*, *attribute-group* and *modules* reference the "Schema" itself. If declarations or definitions are externally defined then the "parent_EID" is the *EID* of the corresponding *module* ("blue arrow"). The "Schema" relation is an EMX respectively the root of an EMX as already mentioned above.

[3]The EII <attribute> and <attributeGroup> are the same in the EMX, an *attribute-group* is always a container

The "Annotation" relation can reference every other relation according to the XML Schema specification. Wildcards are realized as an element wildcard, which belongs to a "Group" (i.e. EII <any>), or they can be attribute wildcards which belongs to a "CT" or "Attribute_Gr" (i.e. EII <anyAttribute>). Every "Element" relation (i.e. element declaration) has either a simple type or a complex type, and every "Element_Ref" relation has an element declaration. Attributes and attribute-groups are the same in an EMX, as mentioned above.

Moreover figure 3 illustrates the distinction between visualized ("yellow border") and not visualized relations. Under consideration of table 2 six relations are direct visible in an EMX: constraints, annotations, modules, groups and because of the *Garden of Eden* style element references and attribute-group references. Table 3 summarizes which relation of figure 3 belongs to which EMX node of table 2.

EMX Node	Relation
element	Element, Element_Ref
attribute-group	Attribute, Atttribute_Ref, Attribute_Gr, Attribute_Gr_Ref
group	Group, Wildcard
st	ST, ST_List, Facet
ct	CT
annotation	Annotation
constraint	Contraint, Path, Assert
module	Module

Table 3: EMX Nodes with Logical Structure

The EMX node *st* (i.e. simple type) has three relations. These are the relation "ST" for the most simple types, the relation "ST_List" for set free storing of simple union types and the relation "Facet" for storing facets of a simple restriction type. *Constraints* are realized through the relation "Path" for storing all used XPath statements for the *element information items* (EII) <key>, <unique> and <keyref> and the relation "Constraint" for general properties e.g. name, XML Schema id, visualization information, etc. Furthermore the relation "Assert" is used for storing logical constraints against complex types (i.e. EII <assert>) and simple types (i.e. EII <assertion>). Figure 4 illustrates the

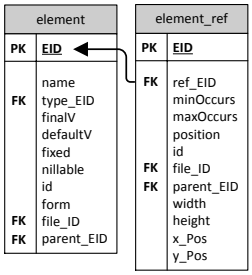

Figure 4: Relations of EMX Node *element*

stored information concerning the EMX node *element* respectively the relations "Element" and "Element_Ref". Both relations have in common, that every tuple is identified by using the primary key *EID*. The *EID* is one-to-one in every EMX as mentioned above. The other attributes are specified under consideration of the XML Schema specification [4], e.g. an element declaration needs a "name" and a type ("type_EID" as a foreign key) as well as other optional values like the final ("finalV"), default ("defaultV"), "fixed", "nillable", XML Schema "id" or "form" value. Other EMX specific attributes are also given, e.g. the "file_ID" and the "parent_EID" (see figure 3). The element references have a "ref_EID", which is a foreign key to a given element declaration. Moreover attributes of the occurrence ("minOccurs", "maxOccurs"), the "position" in a content model and the XML Schema "id" are stored. Element references are visualized in an EMX. That's why some values about the position in an EMX are stored, i.e. the coordinates ("x_Pos", "y_Pos") and the "width" and "height" of an EMX node. The same position attributes are given in every other visualized EMX node.

The edges of the formal definition of an EMX can be derived by knowing the logical structure and the visualization of an EMX. Figure 5 illustrates the allowed edges of EMX nodes. An edge is always a pair of *EID*s, from a source

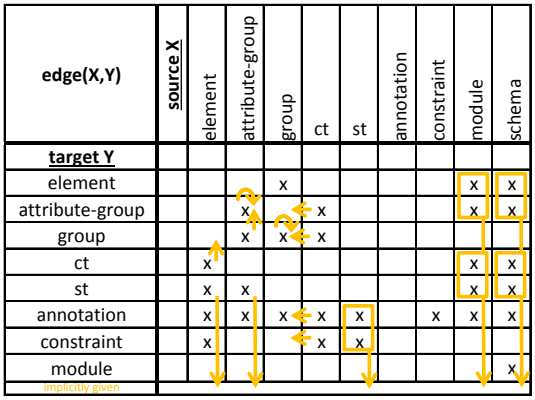

Figure 5: Allowed Edges of EMX Nodes

("X") to a target ("Y"). For example it is possible to add an edge outgoing from an *element* node to an *annotation*, *constraint*, *st* or *ct*. A "black cross" in the figure defines a possible edge. If an EMX is visualized then not all EMX nodes are explicitly given, e.g. the *type-definitions* of the *abstract data model* (i.e. EMX nodes *st*, *ct*; see table 2). In this case the corresponding "black cross" has to be moved along the given "yellow arrow", i.e. an edge in an EMX between a *ct* (source) and an *attribute-group* (target) is valid. If this EMX is visualized, then the *attribute-group* is shown as a child of the *group* which belongs to above mentioned *ct*. Some information are just "implicitly given" in a visualization of an EMX (e.g. simple types). A "yellow arrow" which starts and ends in the same field is a hint for an union of different nodes into one node, e.g. if a *group* contains a wildcard then in the visualization only the *group* node is visible (extended with the "blue W"; see table 2).

4. EXAMPLE

In section 3 the conceptual model EMX was introduced. In the following section an example is given. Figure 6 illustrates an XML Schema in the *Garden of Eden* modeling style. An event is specified, which contains a place ("ort") and an id ("event-id"). Furthermore the integration of other

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:eve="gvd2013.xsd" targetNamespace="gvd2013.xsd">
  <xs:element name="event" type="eve:eventtype">
    <xs:annotation/>
  </xs:element>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="datum" type="xs:date"/>
  <xs:complexType name="orttype">
    <xs:sequence>
      <xs:element ref="eve:name"/>
      <xs:element ref="eve:datum"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="ort" type="eve:orttype"/>
  <xs:attribute name="event-id" type="xs:string"/>
  <xs:complexType name="eventtype">
    <xs:sequence>
      <xs:element ref="eve:ort"/>
    </xs:sequence>
    <xs:attribute ref="eve:event-id"/>
    <xs:anyAttribute/>
  </xs:complexType>
</xs:schema>
```

Figure 6: XML Schema in *Garden of Eden* Style

attributes is possible, because of an attribute wildcard in the respective complex type. The place is a sequence of a name and a date (”datum”).

All type definitions (*NCNAME*s: ”orttype”, ”eventtype”) and declarations (*NCNAME*s: ”event”, ”name”, ”datum”, ”ort” and the attribute ”event-id”) are globally specified. The target namespace is ”eve”, so the *QNAME* of e.g. the complex type definition ”orttype” is ”eve:orttype”. By using the *QNAME* every above mentioned definition and declaration can be referenced, so the re-usability of all components is given. Furthermore an attribute wildcard is also specified, i.e. the complex type ”eventtype” contains apart from the content model sequence and the attribute reference ”eve:event-id” the *element information item* <anyAttribute>.

Figure 7 is the corresponding EMX of the above specified XML Schema. The representation is an obvious simplification, it just contains eight well arranged EMX nodes. These are the *elements* ”event”, ”ort”, ”name” and ”datum”, an *annotation* as a child of ”event”, the *groups* as a child under ”event” and ”ort”, as well as an *attribute-group* with wildcard. The simple types of the element references ”name” and ”datum” are implicitly given and not visualized. The complex types can be derived by identifying the elements which have no specified simple type but groups as a child (i.e. ”event” and ”ort”).

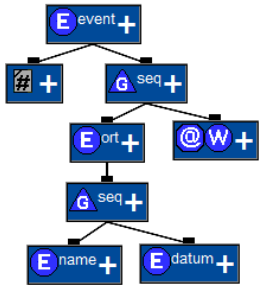

Figure 7: EMX to XSD of Figure 6

The edges are under consideration of figure 5 pairs of not visualized, internally defined *EID*s. The source is the side of the connection without ”black rectangle”, the target is the other side. For example the given *annotation* is a child of the *element* ”event” and not the other way round; an element can never be a child of an *annotation*, neither in the XML Schema specification nor in the EMX.

The logical structure of the EMX of figure 7 is illustrated in figure 8. The relations of the EMX nodes are given as well

Schema

EID	xmlns_xs	targetName	TNPrefix
1	http://www.w3.org/2001/XMlSchema	gvd2013.xsd	eve

Element

EID	name	type_EID	parent_EID
2	event	14	1
3	name	11	1
4	datum	12	1
5	ort	13	1

Annotation

EID	parent_EID	x_Pos	y_Pos
10	2	50	100

Wildcard

EID	parent_EID
17	14

Element_Ref

EID	ref_EID	minOccurs	maxOccurs	parent_EID	x_Pos	y_Pos	
6	2	1	1	1	75	75	event
7	3	1	1	16	60	175	name
8	4	1	1	16	150	175	datum
9	5	1	1	15	100	125	ort

ST

EID	name	mode	parent_EID
11	xs:string	built-in	1
12	xs:date	built-in	1

CT

EID	name	parent_EID
13	orttype	1
14	eventtype	1

Group

EID	mode	parent_EID	x_Pos	y_Pos	
15	sequence	14	125	100	eventsequence
16	sequence	13	100	150	ortsequence

Attribute

EID	name	parent_EID
18	event-id	1

Attribute_Ref

EID	ref_EID	parent_EID
19	18	14

Attribute_Gr

EID	parent_EID
20	1

Attribute_Gr_Ref

EID	ref_EID	parent_EID	x_Pos	y_Pos
21	20	14	185	125

Figure 8: Logical Structure of Figure 7

as the attributes and corresponding values relevant for the example. Next to every tuple of the relations ”Element_Ref” and ”Group” small hints which tuples are defined are added (for increasing the readability). It is obvious that an *EID* has to be unique, this is a prerequisite for the logical structure. An *EID* is created automatically, a user of the EMX can neither influence nor manipulate it.

The element references contain information about the occurrence (”minOccurs”, ”maxOccurs”), which are not explicitly given in the XSD of figure 6. The XML Schema specification defines default values in such cases. If an element reference does not specify the occurrence values then the standard value ”1” is used; an element reference is obligatory. These default values are also added automatically.

The stored names of element declarations are *NCNAME*s, but by knowing the target namespace of the corresponding schema (i.e. ”eve”) the *QNAME* can be derived. The name of a type definition is also the *NCNAME*, but if e.g. a built-in type is specified then the name is the *QNAME* of the XML Schema specification (”xs:string”, ”xs:date”).

5. PRACTICAL USE OF EMX

The co-evolution of XML documents was already mentioned in section 1. At the University of Rostock a research prototype for dealing with this co-evolution was developed: CodeX (Conceptual design and evolution for XML Schema)

[5]. The idea behind it is simple and straightforward at the same time: Take an XML Schema, transform it to the specifically developed conceptual model (EMX - Entity Model for XML-Schema), change the simplified conceptual model instead of dealing with the whole complexity of XML Schema, collect these changing information (i.e. the user interaction with EMX) and use them to create automatically transformation steps for adapting the XML documents (by using XSLT - Extensible Stylesheet Language Transformations [1]). The mapping between EMX and XSD is unique, so it is possible to describe modifications not only on the EMX but also on the XSD. The transformation and logging language ELaX (Evolution Language for XML-Schema [8]) is used to unify the internally collected information as well as introduce an interface for dealing directly with XML Schema. Figure 9 illustrates the component model of CodeX, firstly published in [7] but now extended with the ELaX interface.

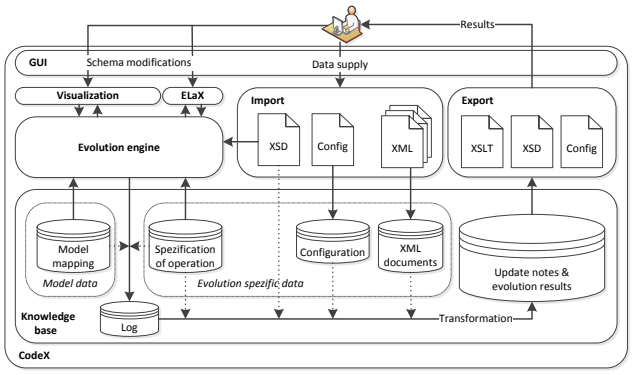

Figure 9: Component Model of CodeX [5]

The component model illustrates the different parts for dealing with the co-evolution. The main parts are an import and export component for collecting and providing data of e.g. a user (XML Schemas, configuration files, XML document collections, XSLT files), a knowledge base for storing information (model data, evolution specific data and co-evolution results) and especially the logged ELaX statements ("Log"). The mapping information between XSD and EMX of table 2 are specified in the "Model data" component.

Furthermore the CodeX prototype also provides a graphical user interface ("GUI"), a visualization component for the conceptual model and an evolution engine, in which the transformations are derived. The visualization component realizes the visualization of an EMX introduced in table 2. The ELaX interface for modifying imported XML Schemas communicates directly with the evolution engine.

6. CONCLUSION

Valid XML documents need e.g. an XML Schema, which restricts the possibilities and usage of declarations, definitions and structures in general. In a heterogeneous changing environment (e.g. an information exchange scenario), also "old" and longtime used XML Schema have to be modified to meet new requirements and to be up-to-date.

EMX (Entity Model for XML-Schema) as a conceptual model is a simplified representation of an XSD, which hides its complexity and offers a graphical presentation. A unique mapping exists between every in the *Garden of Eden* style modeled XSD and an EMX, so it is possible to representatively adapt or modify the conceptual model instead of the XML Schema.

This article presents the formal definition of an EMX, all in all there are different nodes, which are connected by directed edges. Thereby the *abstract data model* and *element information item* of the XML Schema specification were considered, also the allowed edges are specified according to the specification. In general the most important components of an XSD are represented in an EMX, e.g. elements, attributes, simple types, complex types, annotations, constrains, model groups and group definitions. Furthermore the logical structure is presented, which defines not only the underlying storing relations but also the relationships between them. The visualization of an EMX is also defined: outgoing from 18 relations in the logical structure, there are eight EMX nodes in the conceptual model, from which six are visualized.

Our conceptual model is an essential prerequisite for the prototype CodeX (Conceptual design and evolution for XML Schema) as well as for the above mentioned co-evolution. A remaining step is the finalization of the implementation in CodeX. After this work an evaluation of the usability of the conceptual model is planned. Nevertheless we are confident, that the usage is straightforward and the simplification of EMX in comparison to deal with the whole complexity of an XML Schema itself is huge.

7. REFERENCES

[1] XSL Transformations (XSLT) Version 2.0. `http://www.w3.org/TR/2007/REC-xslt20-20070123/`, January 2007. Online; accessed 26-March-2013.

[2] Extensible Markup Language (XML) 1.0 (Fifth Edition). `http://www.w3.org/TR/2008/REC-xml-20081126/`, November 2008. Online; accessed 26-March-2013.

[3] XQuery 1.0 and XPath 2.0 Data Model (XDM) (Second Edition). `http://www.w3.org/TR/2010/REC-xpath-datamodel-20101214/`, December 2010. Online; accessed 26-March-2013.

[4] W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. `http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/`, April 2012. Online; accessed 26-March-2013.

[5] M. Klettke. Conceptual XML Schema Evolution - the CoDEX Approach for Design and Redesign. In *BTW Workshops*, pages 53–63, 2007.

[6] E. Maler. Schema design rules for ubl...and maybe for you. In *XML 2002 Proceedings by deepX*, 2002.

[7] T. Nösinger, M. Klettke, and A. Heuer. Evolution von XML-Schemata auf konzeptioneller Ebene - Übersicht: Der CodeX-Ansatz zur Lösung des Gültigkeitsproblems. In *Grundlagen von Datenbanken*, pages 29–34, 2012.

[8] T. Nösinger, M. Klettke, and A. Heuer. Automatisierte Modelladaptionen durch Evolution - (R)ELaX in the Garden of Eden. Technical Report CS-01-13, Institut für Informatik, Universität Rostock, Rostock, Germany, Jan. 2013. Published as technical report CS-01-13 under ISSN 0944-5900.

[9] E. van der Vlist. *XML Schema*. O'Reilly & Associates, Inc., 2002.