published in: Proceedings of 2nd International Conference, EC Web, Springer Verlag, ISBN 3-540-42517-9,
pp 71-80

Composition of XML-Transformations

Johann Eder and Walter Strametz

University of Klagenfurt
Department of Informatics Systems
eder@isys.uni-klu.ac.at, walter.strametz@infologs.com

Abstract. Electronic commerce seeks improvements of business processes by aggressively exploiting the enormous increases in information exchange offered by digital telecommunication systems. XML is seen as an important step to overcome the problems of heterogeneity of data exchange between different systems, albeit the structural as well as the semantic heterogeneities are not even touched by this standard: The same information is encoded quite differently in XML by different information systems. Therefore, to let these information systems communicate and interoperate, it is necessary to transform XML documents.
We propose a new way to generate such transformations based on the XSLT language which was originally developed for rendering XML documents. We aim to improve the way XSLT transformations are developed by binding XSLT transformers to the document type descriptions of source and target documents and introducing and exploiting the concepts of composition and specialization for DTD as well as for transformers in XSLT, resulting in highly improved efficiency and quality.

Keywords: XML, heterogeneous information systems, e-commerce

1 Introduction

Electronic commerce in all its forms and variants depends on the electronic exchange of information - in interactive form, and/or by exchange of electronic documents. While interactive form is still frequently used in business-to-consumer (B2C) applications, in business-to-business (B2B) applications the exchange of electronic documents is the preferred way. This allows to overcome costly media breaks in business processes and enables IT-systems to interoperate with different IT-systems of business partners [5].

In such a scenario, an IT-system has to be capable of accepting electronic documents from various sources or generating electronic documents for various receivers. The necessity of adapting to formats of electronic documents defined by others depends on the market power of the organizations.

There are no generally accepted standards for electronic business documents, although many attempts have been made (e.g. EDIFACT). Exchange of documents in electronic commerce suffers from heterogeneity on several levels: from the choice of the code, the structure of documents up to semantic differences.

For the lower levels of electronic document exchange XML, the extended markup language [6], is developing as generally accepted standard. It can be soon taken for granted that it is possible to send XML documents to a communication partner and this communication partner is equipped with software to process XML documents. However, it is not at all sure that this communication partner also understands the documents. For successful communication it will be necessary to negotiate the form of XML documents and/or to transform XML documents into different XML representations.

For an example: There are numerous electronic bookshops on the web. So it should be easy to write an application to get the best bid for a given book (including taxes and shipment costs). However, all the book-outlets have different interfaces for costumers and even if a request for quote could be received by sending an XML document, all the document forms are probably different. So for writing the application sketched above, it is indeed necessary to transform the XML document into different forms.

In this paper we report on an approach to facilitate the development of XML-transformers. We use the widely available language XSLT [3, 7, 6, 8, 11] as transformation language. To overcome some shortcomings of this language we first bind XSLT transformers to the DTD (document type definition) of the source and destination document types. This makes it easier to search for appropriate transformers when a new one has to be developed. Then we introduce the notions of composition and specialization of XML documents as well as of XSLT transformers and provide a meta structure for storing this information. Then we can use this structure for composing new XSLT transformations from already available component transformers.

We will also briefly describe our prototype implementation of a transformation system named CoX (component-based XML transformer), based on these concepts.

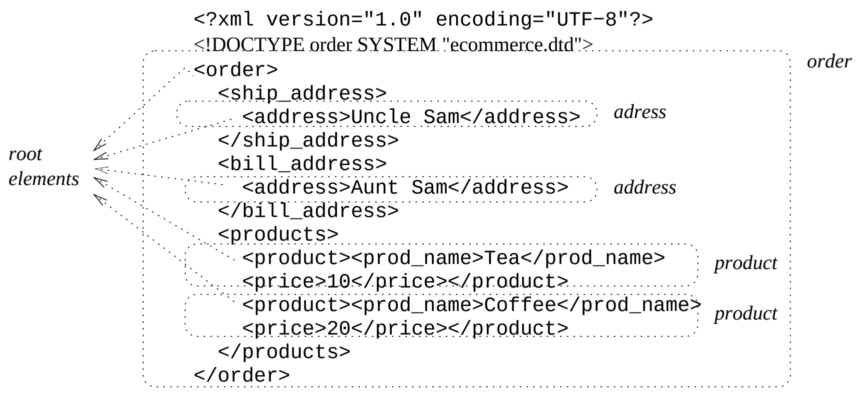
2 XML and XSLT

In this section we revisit the basic notions of XML and XSLT to introduce the concepts and terminology we need in the following sections to make the paper more self-contained.

2.1 XML and DTDs

Semi-structured data [2, 1, 9] can be represented with the XML language [3, 10] standardized by the W3C. Semi-structured data (documents) are modelled in form of trees, whereby nodes contain data and the named edges (*tags*) describe the nodes (*elements*). The labels are interpreted as schema information and thus the tree contains the schema information of the document. For an example, Figure 1 shows a sample document of a simplified order document.

The tree-representation of an XML document such as an order can be constrained and further documented by a *Document Type Definition* (DTD), which



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE order SYSTEM "ecommerce.dtd">
<order>
  <ship_address>
    <address>Uncle Sam</address>
  </ship_address>
  <bill_address>
    <address>Aunt Sam</address>
  </bill_address>
  <products>
    <product><prod_name>Tea</prod_name>
    <price>10</price></product>
    <product><prod_name>Coffee</prod_name>
    <price>20</price></product>
  </products>
</order>
```

Fig. 1. An order-document in XML. The dotted rectangles indicate the components of the document and the arrows the root elements of the components.

is part of the XML language. In Figure 2 the DTD of the order document of Figure 1 is shown. A DTD is a context-free grammar for the structure of an XML document. An XML parser validates the conformance of a document with a DTD. For example, in the order-document a product-tag inside an address-tag should not be accepted. The DTD is an optional part of an XML document but in this paper we only consider documents with DTD - syntactically expressed in the DOCTYPE clause of an XML document. E.g. the clause `<!DOCTYPE order SYSTEM ''ecommerce.dtd''>` binds the DTD "ecommerce.dtd" to the document of Figure 1.

```
<!ELEMENT order (ship_address, bill_address?, products)>
<!ELEMENT bill (bill_address, products)>
<!ELEMENT ship_address (address)>
<!ELEMENT bill_address (address)>
<!ELEMENT address ANY>
<!ELEMENT products (product)*>
<!ELEMENT product (ANY)>
```

Fig. 2. The DTD of the XML document in Figure 1.

As a DTD may feature many choices and optional parts, a particular DTD can describe several disjoint sets of document schemas - for example it is possible that one single DTD can validate an order-document as well as an invoice. The DTD in Figure 2 uses this property to validate an order as well as a billing document. Thereby, the root-element of the document is used as an entry-point to the DTD to put together the grammar for validating the schema-tree. As you can see the `bill` element shares structure (`bill_address`, `products`) with the `order` element. So parts of the DTD can be reused with the consequence that documents share structural parts for particular subtrees of the XML document. The main idea of our approach is to reuse the transformations of shared structures among different documents which are based on the same DTD.

2.2 Transforming Documents with XSLT

XSLT (*eXtensible Stylesheet Language for Transformations*) [3, 7, 6, 8, 11] is a language for transforming XML documents. A transformation in the XSLT language is expressed as an XML document and can therefore be validated and parsed by XML parsers. A transformation expressed in XSLT consists of rules for transforming a source tree of an XML document into a result tree, which are executed by an XSLT processor. The XSLT standard does not specify how an XSLT transformation is associated with an XML document. As a consequence a system has to keep track which transformations must be applied on a document in order to get the desired output. When transforming various document types this task becomes difficult to manage.

As a solution to this we wanted a transformation system which supports document types: A transformer (consisting of a set of XSLT transformation programs) should be simply invoked by providing the source document and a target type of the output document. The transformation process itself should be transparent to the user. Our solution to this problem is to bind the XSLT transformation to the DTDs of source and target documents. The CoX transformation system uses this information to find and apply the proper transformers for the specific instance of a document type. We extended this basic idea and added support for composition and specialization to further simplify and accelerate the process of creating document transformations.

3 Composition and Specialization of Documents

3.1 XML Components

A document can be seen as an aggregation of components. For example, in a document for ordering goods we can identify the shipping and billing address, the products, etc. as components of the ordering document. In an XML document the tags can be interpreted as delimiter of the components of the document. In the tree model thus we define a component of an XML document as a subtree identified through its root element.

The type of a component is then defined as the elements reachable from the type name of its root element in the DTD of the document. A component can in this way be validated against its DTD and type using conventional XML parsers.

In principle, every element defined in a DTD can be considered as a component. However, we recommend to restrict to those elements which represent semantically meaningful units (entities) of the problem domain.

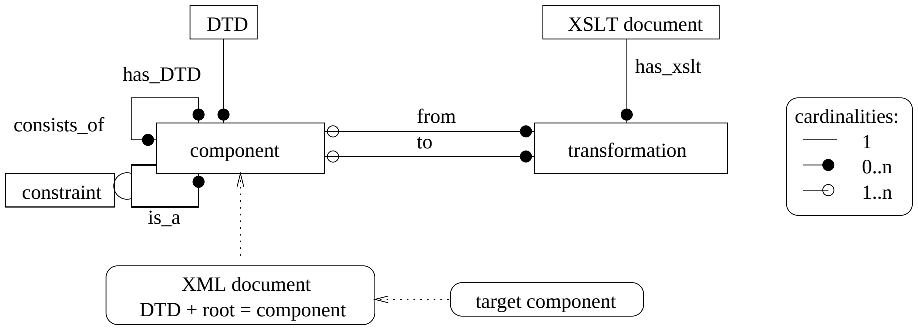


Fig. 3. Type information stored within the CoX transformation system (metaschema).

In the metaschema we represent the aggregation hierarchies of documents and of document types. Fig. 3 shows the realization of the definitions above with the associations from component to the entities DTD and root: A component is related to exactly one DTD and has exactly one root element. Defining components only makes sense if there are several components for one DTD. Later, when we introduce specialization it will be clear that having multiple components which have the same DTD and the same root element is perfectly reasonable.

We can generalize the meaning of a component in a way that also a whole document can be seen as a component. So a document is a component with a certain base type (i.e. a DTD together with a root element). Like a document can have components also a component may be composed of components in turn. The *consists_of*-relation of Figure 3 reflects that a component may consist of several components. The model also shows that every component is assigned to a DTD and to an element name. Given that information an XML parser can validate a component.

3.2 Specialization of XML-Components

So far we presented the aggregation structure of XML documents and XML types. In good modeling tradition, we also introduce the notion of specialization. This concepts covers three different phenomena:

1. specialization through environment
2. specialization through restricted structure
3. specialization through restricted instances

The first type of specialization allows to distinguish between components with the same type but different semantics due to different environment. As an example for this kind of specialization we look at the components shipping and billing address of the example above. Both are addresses and we can't distinguish

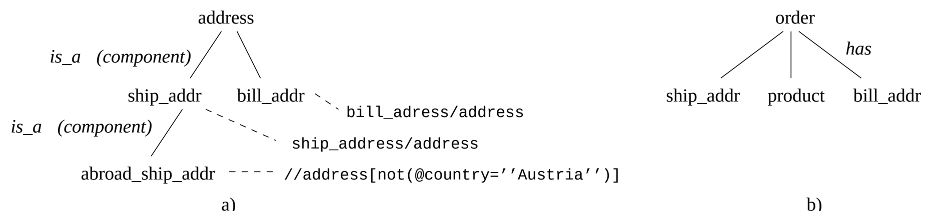


Fig. 4. a) is an example of a specialization tree of the address component. b) shows the component tree of an order document.

between them by type and therefore we can bind only one transformation to the addresses.

The second kind of specialization considers the fact that a document type definition may be very generic and allows the validation of many different structures of documents. We define a specialization as a DTD which restricts the structural genericity of a DTD. For an example, it might be required that the shipping address is a physical address with street and number and not merely a post-box address. So the physical address restricts the DTD of address and it would validate only a subset of valid address documents.

The third way of specialization restricts the possible instances of a type by restricting values. As an example, certain values can be defined as mandatory. The type "domestic-address" would require that the country component of an address is always filled with Austria, in our example.

For all three kinds of specialization we found a uniform way of representing them in our metaschema: We define a component A as specialization of component B, if every document which is valid for A is also valid for B, and additionally satisfies the specialization constraint. The relationships between components defined through specialization is represented through the *is_a* hierarchy in our metaschema where the condition attribute of the *is_a* relationship represents the specialization constraint.

The specialization constraints are expressed in the language XPath [3, 12] which is part of XSLT and thus can be processed by an XSLT processor. XPath provides a way to select a subtree of a document by selecting a specific element. If the selected subtree is identical to the instance of the specialized component the constraint is satisfied. The identity is checked by comparing the root element of the component instance with the result of the XPath expression. If both elements are equal then the test succeeds. The test is also called an "element test".

XPath allows to specify the path to an element of a document. The syntax of a sample XPath constraint for the shipping address is: "`ship_address/address`" which matches any `address` element with an `ship_address` parent (see also Figure 4a). With this functionality it is possible to distinguish between the billing and the shipping address of the sample document in Figure 1: When the transformer recognizes an address it executes an element test for the root element (`<address>`) of the component. It evaluates the XPath expression which is assigned to a specialized component. If the result of the XPath expression contains the root element of the component instance then the test is successful and we can infer that the document is valid for the specialization (i.e. the actual type of the document is the subtype considered). [7, 6, 12] provide details of XPath and the expressive power of this language.

3.3 Specialization and Transformation

The specialization allows to provide different transformations for documents with the same DTD based on the their dynamic subtypes, i.e. on the specialization whose specialization conditions they satisfy.

In our metaschema (Figure 3) the *is_a* relationship represents the specialization for components. A component can be a specialization of an other component, whereby all assoziations and attributes are inherited from the parent. The assoziations except *has_DTD* can be overwritten by the specializing component. Thus the base type of a specialization hierarchy is the same for all components.

Figure 4a shows an example of the specialization hierarchy of the address component and its constraints. The root of the tree is the most general component with the most general transformation. Each specialization adds a constraint to the constraints inherited by its preceding components. To meet the criteria for a specialized component all constraints must be checked successfully. In the example the *abroad_ship_addr* component has to satisfy two constraints - it is a shipping address (environment specialization) and it does not contain country code "Austria" (instance specialization). The transformation of this component can be highly specialized in transforming "abroad shipping addresses" and the transformation program can rely on the constraints defined in the specialization hierarchy. If no transformation is defined for this component then the transformation of the *ship_addr* component will be applied. It is guaranteed that all possible specializations of *ship_addr* meet the criteria to be a *ship_addr* component and, therefore, all transformations defined for *ship_addr* can be applied.

4 Generating Transformations

4.1 The Process of Transforming a Document

In this section we discuss how a document is transformed with the CoX transformation system. Assume that some XSLT transformations are already programmed and the type information is stored within the system. We start with an XML-document and the target type. With the DTD and the root element of the document the principal component can be found. Now the type of the source and the type target document is known. The CoX transformer will only apply transformations which lead to the DTD of the target component.

To determine the dynamic type of the document the transformer analyzes the source document and determines the base type by using the DTD and the root element of the component. The *is_a* relation is searched depth-first to apply the constraint expressions on the document. So a document is decomposed and the most specialized subcomponents are identified by checking the constraints given in the specialization hierarchy.

When a component is transformed it is important that the transformation has access to the component only. Otherwise an XSLT transformation could have undesirable side effects on the whole document. To obtain a scope for each component the CoX transformer extracts the XML subtree representing the from the document. Each component is transformed as it was a separate document.

For the transformation, therefore the document is decomposed into it's components which are in turn transformed and afterwards (re-)assembled to obtain the target XML document.

The transformer uses the *consists_of* association to recursively determine the smallest components for which a transformation program to the target DTD has been registered. For each of the components identified in this way, the *is_a* hierarchy is descended to identify the most specialized dynamic type of each component in a depth-first manner. A component is thus transformed if its *consists_of* relation is empty or if all of its components are already transformed. If no transformation is found the search continues at the next higher component in the specialization hierarchy.

4.2 Transformation of a Sample Document

On the sample document of Figure 1 we want to show the principle of how a transformation of a document is generated.

First, the transformation system has to make sure that the schema information is correct. For example the component and specialization trees must be checked if they are correct trees and have no circular links. Another requirement is that all components in the component tree have the same DTD, and so on. The data for the metaschema of Figure 3 is stored in an external XML document which is read at startup time of the transformation system.

With the CoX transformation system it is possible to transform one document to many other types and DTDs simply by naming the target component or target DTD. In the following example a document which represents an order document (see Figure 1) should be transformed into an HTML representation. In Figure 4b you can see a fragment of the type information of the order document which is stored within the CoX transformation system. The specialization tree of the address component (see Figure 4a) is also stored within the system. For the transformation of the components there are five transformations defined[1]: *t_order*, *t_addr*, *t_ship_addr*, and *t_prod* which are assigned to the according components. Note that there is no transformation for the billing address.

With the DTD and the root element of the document the transformation system selects the proper component tree of the order document. From there the transformer learns to look for an `address` component. Two address components

[1] The transformations may be implemented with one XSLT document or even more than five documents using `import` and `include` statements.

are found which are processed consecutively. As there are specializations for an address it searches the specialization tree, evaluates the XPath constraints and determines the dynamic subtype, which is *ship_addr*. This leads to the *t_ship_addr* transformation which is applied to the subtree (starting an the first `<address>` tag). The second address is processed similar but the dynamic subtype is *bill_addr*. This component has no transformation assigned and therefore the transformation is taken from the *address* component. In other words the billing address inherits the transformation from the address component. Then the transformations for the products are applied. The final transformation is the order transformation which is the only transformation having access to the whole document. This way also a final rearrangement or the merging of components can be carried out. So the transformer executes the following sequence of XSLT transformations in their particular scope: *t_ship_addr*, *t_addr*, *t_prod* and *t_order*. The transformation of the whole document is composed of a set of transformers for parts of this document and can be generated from these partial transformers.

4.3 Creating New Transformations

The more transformations and different DTDs are employed in a system the more important it gets to manage the transformations. The CoX transformer helps to organize documents and transformations in reusable pieces. A major goal for the development of CoX was to support the programmers of transformation programs to increase the efficiency of the process for creating transformations and to achieve quality improvements by employing certified well tested component transformers.

The process of creating a new transformation starts with searching the meta-structure whether transformations for the DTD of the document have been defined already. If no suitable transformation is found, the search is continued for the components of the DTD. If transformation programs for components are found, they can be ready used as part of the transformation of the new document. The specialization hierarchy also supports the creation and application of transformations which reflect the actual structure (dynamic type) and content of a document, which increases the flexibility for handling semi-structured data in a flexible yet reliable way.

4.4 Implementation

The previous sections introduced the concepts of composition and specialization for XML transformations. The ideas are applicable for the transformation of an XML document and do not depend on any transformation language. To keep the implementation simple we have chosen XSLT as the transforming language. The CoX transformer is written in the Java language and is based on the XSLT/XPath processor "Xalan" which is available on http://www.apache.org. The architecture of CoX makes extensive usage of the factory pattern [4] which makes it possible to exchange the XSLT and the XPath processors. The type information about components, DTDs and specialization trees is stored in an XML document which is parsed and evaluated by the CoX transformer. For that it uses the "Xerces" XML parser which is also exchangeable. The transformation system is accessible via a Java, a commandline and a graphical interface.

5 Conclusions

We presented the XML transformation systems CoX and its underlying methodology. CoX was developed to support the process of developing transformations of XML documents from one format to another. The main contributions of this approach are the following:

– Recording source and target DTDs of XSLT transformations.
– Composition and specialization of XML documents.
– Composition and specialization of XSLT transformations.

The information obtained by decomposition and specialization is documented in a metastructure which is then used in the process of identifying available transformations.

The methodology introduced in the CoX system is intended to increase the efficiency of the development process for creating transformations between XML documents. It greatly increases the productivity by promoting reuse of already developed transformations (for components) and thus reduces the necessity of developing new transformations from scratch.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web.* Morgan Kaufmann Publishers, 2000.
[2] P. Buneman. Semistructured data. Tutorial in Proceedings of the 16th ACM Symposium on Principles of Database Systems, 1997.
[3] World Wide Web Consortium. W3C. http://www.w3c.org/, 2001.
[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software.* Addison-Wesley, 1996.
[5] H. Groiss and J. Eder. Workflow systems for inter-organizational business processes. *ACM SIGGROUP Bulletin*, Dec. 1997.
[6] E. R. Harold. *XML Bible.* IDG Books Worldwide, 1999.
[7] D. Martin, M. Birbeck, M. Kay, et al. *Professional XML.* Wrox Press, 2000.
[8] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. In *Proceedings of the Nineteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2000, Dallas, Texas)*, 2000.
[9] D. Suciu. An overview of semistructured data. *ACM SIGACT News*, 1998.
[10] D. Suciu. Semistructured data and XML. In Proceedings of International Conference on Foundations of Data Organization, Kobe, Japan, 1998.
[11] P. Wadler. A formal semantics of patterns in XSLT. Markup Technologies, 1999.
[12] P. Wadler. Two semantics for XPath. http://cm.bell-labs.com/cm/cs/who/wadler/, 2000.