# XML Schema Correspondences

Arne Handt, Joachim Quantz
Shinka Technologies AG
Tempelhofer Ufer 8/9
10963 Berlin
arne.handt@shinkatech.com, joachim.quantz@shinkatech.com

**Abstract:** In recent years, XML has become more and more accepted as the ideal format for information exchange between distributed real-world applications. XML Schema is the predominant standard used for describing the syntactic structure of the XML data to be exchanged. In an ideal world, all applications would use the same XML schemata. In reality, however, the applications to be integrated often use different Schemata, i.e. they use different syntactic formats for representing similar content.
Dynamic mediation of content and services in a semantic web therefore needs to overcome the difficulties of data format heterogeneity. In this article we introduce the notion of XML schema correspondences which describe the relations between the information models of two systems in a way that mediating between these systems can be efficiently automatized. The main idea is to explicitly model schema correspondences and to generate code from them which performs transformations between different XML formats on the fly.

## 1. Introduction

In recent years, XML has become more and more accepted as the ideal format for information exchange between distributed real-world applications. For describing and integrating these applications, the emerging family of WebServices-related technologies provides a starting point. Two important technologies in this context have been adopted recently:

- XML Schema [XSchema] is the predominant standard used for describing the syntactic structure of the XML data to be exchanged;
- WSDL (Web Services Description Language, [WSDL]) is the most important format for the definition of service interfaces and bindings of those interfaces to concrete protocols.

In a WSDL document, a service interface is described by means of operations which take input messages and produce output messages. This design follows the idea of applications which communicate by the exchange of business documents represented as XML messages. The types for both, input and output messages, are described using XML schemata. In an ideal world, all applications would use the same schemata. In

reality, however, the applications to be integrated often use different XML Schemata, i.e. they use different syntactic formats for representing similar content.

Dynamic mediation of content and services in a semantic web therefore needs to overcome the difficulties of data format heterogeneity. In this article we introduce the notion of XML schema correspondences which describe the relations between the information models of two systems in a way that mediating between these systems can be efficiently automatized.

The main idea is to explicitly model schema correspondences and to generate code from them which perform transformations between different XML formats on the fly. Thus, Application A sends out its information according to Schema A. Based on the Schema correspondence explicitly modeled between Schemas A and B this information is then transformed into information according to Schema B before being received by Application B.

The general concept of model correspondences is well-known in the domain of federated information systems. One particular objective of the work presented in this paper is to develop a complete process of efficiently automatized mediation which ensures consistency from correspondence declaration to message transformation. The requirements for this process include support for domain experts in modeling schema correspondences on an intuitive level without having to know the underlying technologies as well as a precisely defined mapping of schema correspondences to transformation rules, which can be efficiently be applied at runtime. For this purpose we decided to develop a new transformation language which is suited for the representation of transformation rules derived from schema correspondences and allows for the generation of efficient transformation code in a variety of target languages.

This article is structured as follows:

- Chapter 2 outlines the process of integrating heterogeneous systems by utilizing schema correspondences;
- Chapter 3 introduces XML schema correspondences and illustrates the applicability of this approach with a simple example;
- Chapter 4 describes a functional transformation language for XML which is used as intermediate representation for schema mappings;
- Chapter 5 gives a survey over the implementation of a transformation system for the Shinka Business Integration Platform based on XML schema correspondences;
- Chapter 6 summarizes related work;
- Chapter 7 gives an outlook on future work;
- Chapter 8 concludes.

## 2. Automating Mediator Development

The process of creating and performing transformations based on XML schema correspondences consists of the following steps (see Figure 1):

(1) declaration of schema correspondences;
(2) derivation of message transformations;
(3) generation of an executable transformer and finally
(4) execution of the transformer.

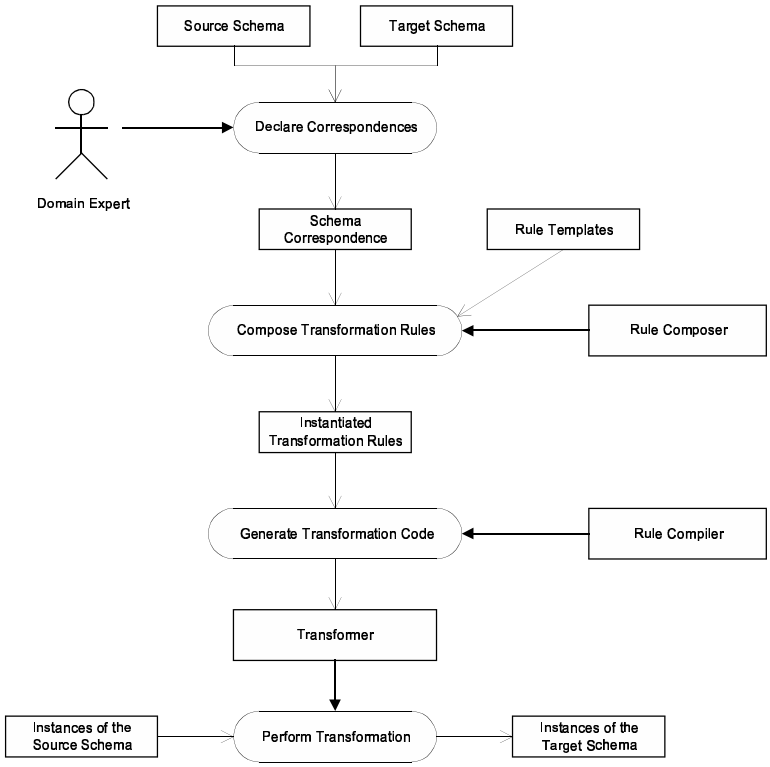


Figure 1: Process Overview

**Declaration of schema correspondences**
The objective of this task is to state the relations between the constituents of the two schemata which are concerned, so that a directed mapping between the two schemata can be derived from the resulting schema correspondence. In general, this task can't be performed automatically, since domain expertise is necessary to provide the

correspondences, but domain experts may be supported by software tools in different ways, e.g. by an automated correspondence recognizer suggesting candidate correspondences.

**Derivation of message transformations**
The schema correspondences declared in the first step are used to derive the transformations which have to be applied to instances of the source schema in order to translate them to an instance of the target schema. These transformations are represented in an abstract intermediate language which serves as a basis for the generation of concrete transformation code later on.

**Generation of an executable transformer**
In order to actually perform message transformation, executable code has to be generated. This may be code in a common programming language or in a XML-specific formalism, such as XSL transformations [XSLT] or the XML query language [XQuery], depending on the runtime environment the transformer is designed for.

**Execution of the transformer**
By plugging the generated transformer into a transformation framework at the transport layer of the underlying integration middleware, messages which are transmitted between the two concerned systems can be intercepted and transformed "on the fly" transparent to the application layer.

## 3. XML Schema Correspondences

The design of XML schema correspondences follows simple goals:

- » *specification of 1-to-1 correspondences* – each schema correspondence describes a mapping between exactly one schema and another.
- » *declarativity* – schema correspondences allow for the specification of mappings between XML schemata independent of a particular transformation language and thus without having to regard procedural aspects of the schema mapping. Furthermore they allow domain experts to abstract from underlying technical details.
- » *directedness* – by design, each schema correspondence is applicable for the transformation of XML messages from an source schema to a target schema. This is due to the fact that in most cases there will be a containment relation between both schemata, i.e. the information described by the source schema is a non-identical subset of the information described by the target schema, so a reverse mapping may not be possible.

Every schema correspondence consists of sub-correspondences relating the components of the respective schemata, i.e. elements and types. A type correspondence states that two

different types are used to express the same information. The relation between the types is substantiated by the declaration of correspondences between elements and attributes in these types.
Beyond these correspondences on the level of the XML structure, corresponding information may of course be represented using different formats for string literals, e.g. in attribute values. For this reason, conversions of corresponding simple types and simple-typed values can be specified based on the XML schema regular expression syntax.

By using schema correspondences, a semantical relationship between two systems can be established if the schemata are embedded in an ontological context, e.g. by using the Resource Description Framework [RDF] for attaching metadata to XML schemata. In this respect, schema correspondences provide connectors between the ontologies of these systems and thus can be regarded as semantic mapping rules.

The following example illustrates the definition of schema correspondences on a simple message type: Appointment information is sent as XML messages from one application to another. The data format for both applications is defined using the following XML schemata:

```
<schema targetNamespace="urn:Example/DateSchema"
        xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="Date">
    <complexType>
      <sequence>
        <element name="Description" type="string"/>
        <element name="Time" type="datetime"/>
        <element name="Participants">
          <complexType>
            <sequence>
              <element name="Participant" type="string"
                       minOccurs="1" maxOccurs="unbounded"/>
            </sequence>
          </complexType>
        </element>
      </complexType>
    </sequence>
  </element>
</schema>
```

Source Message Schema

```
<schema targetNamespace="urn:Example/AppointmentSchema"
        xmlns ="http://www.w3.org/2001/XMLSchema">
  <element name="Appointment">
    <complexType>
      <sequence>
        <element name="Name" type="string"/>
        <element name="Date" type="datetime"/>
        <element name="Attendee" minOccurs="1"
                 maxOccurs="unbounded">
      </sequence>
    </complexType>
```

```
  </element>
</schema>
```

Target Message Schema

The following example messages clarify the correspondences between these two schemata:

```
<Date>
  <Description>Weekly Executive Meeting</Description>
  <Time>2002-07-18T16:00:00-05:00</Time>
  <Participants>
    <Participant>J. Doe</Participant>
    <Participant>R. Funden</Participant>
  </Participants>
</Date>

<Appointment>
  <Name>Weekly Executive Meeting</Name>
  <Date>2002-07-18T16:00:00-05:00</Date>
  <Attendee>J. Doe</Attendee>
  <Attendee>R. Funden</Attendee>
</Appointment>
```

Example Messages

The transformation resulting from this example is a rather simple restructuring of a message which consists mainly in a renaming of elements. Other fundamental transformation operations which have to be regarded are joining and splitting of elements or attributes, conversion of simple-type values and addition and deletion of information.

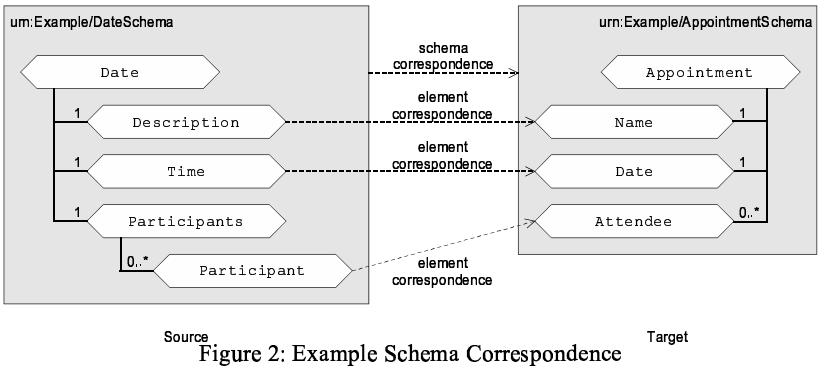


Figure 2: Example Schema Correspondence

Since schema correspondences can be regarded as assertions about schemata, the RDF is an appropriate formalism for the representation of correspondences. The schema correspondence in this example can be represented using the RDF as follows:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
  <rdf:Description about="urn:Example/DateSchema">
    <correspondence target="urn:Example/AppointmentSchema"
                    xmlns="http://www.shinkatech.com/schemas/XSC">
      <correspondence source="Date" target="Appointment"/>
      <correspondence source="Description" target="Name"/>
      <correspondence source="Time" target="Date"/>
      <correspondence source="Participants/Participant"
                      target="Attendee"/>
    </correspondence>
  </rdf:Description>
</rdf:RDF>
```

RDF Representation of a Schema Correspondence

However, a further investigation of how to embed schema correspondences in RDF assertions lies outside the scope of our work. Instead we focus on a intermediate representation which is suitable for the generation code in a variety of target languages. It is described in the next chapter.

## 4. Transformation Language

The purpose of the transformation language is to act as an intermediate representation of the transformations derived from an XML schema correspondence. This representation has to be appropriate to express complex transformation rules composed of the basic transformation operations join, split, add, delete and convert mentioned before. Furthermore it must be simple enough to allow for the generation of transformers in a broad variety of transformation or programming languages, as e.g. XSLT and XML Query. In this respect, it has to form the lowest common denominator of XML transformation languages. Furthermore, the intermediate transformation language serves the following purposes:

- » it is a conceptual aid for the development of the transformation system;
- » it allows for the reuse of how schema correspondences are mapped into XML transformations while abstracting the operational (i.e. specific to the target language) aspects;
- » it allows for optimization of XML transformations independent of the target language.

Based on these requirements the transformation language is designed similar to functional programming languages. It contains a set of fundamental transformation functions which can be combined to form complex transformation operations. The type system these functions operate on covers core XML structures, like node, element and attribute known from [DOM2], as well as datatypes, like string, pattern (a subset of XSLT patterns), NCName and lists.

The core functions of this language are divided into two groups:

» *accessors* provide input to the transformation process, e.g. by selecting particular elements from the input message;
» *constructors* create the output message and insert the input provided by accessors or other constructors.

| **Name** | **Signature** |
|---|---|
| selectDocument | URI → element |
| selectElement | node × pattern → list<element> |
| children | element → list<node> |
| selectAttribute | element × ncname → attribute |
| attributes | element → list<attribute> |
| value | node → string |

Table 1: Accessors (excerpt)

| **Name** | **Signature** |
|---|---|
| createDocument | element → document |
| createElement | list<node> × ncname → element |
| createAttribute | ncname × string → attribute |

Table 2: Constructors (excerpt)

Transformation on lists are realized by a higher order function *forEach* which takes a list and a transformation function on list items as parameters.
Recalling the appointment example, the transformation which can be derived from the correspondences would be expressed in the intermediate transformation language as follows:

```
createDocument(
  createElement(concat([
    createElement(
      [value(first(selectElement(
                        selectDocument("urn:Example/DateSchema"),
                        "Date/Description")))],
      "Name"),
    createElement(
      [value(first(selectElement(
                        selectDocument("urn:Example/DateSchema"),
                        "Date/Time")))],
      "Date")]),
    forEach(
      forEach(
        selectElement(
             selectDocument("urn:Example/DateSchema"),
             "Date/Participants/Participant"),
        value),
      createElement("Attendee"))),
   "Appointment"))
```

This example shows how recurring transformation patterns are translated into idioms in the transformation language, e.g. the realization of an element-to-element mapping by a

combination selecting the value of the source element and constructing an enclosing target element.
The *selectDocument* function is used to constrain the input to this transformation to messages which are instances of the specified namespace, namely the one defined by the source schema.

Based on this representation, the following XSLT stylesheet is generated by the transformer generator:

```
<stylesheet version="1.0"
            xmlns=" http://www.w3.org/1999/XSL/Transform">
  <template match="/Date">
    <element name="Appointment">
      <element name="Name">
        <value-of select="Description"/>
      </element>
      <element name="Date">
        <value-of select="Time"/>
      </element>
      <element name="Attendee">
        <value-of select="Participants/Participant"/>
      </element>
    </element>
  </template>
</stylesheet>
```

## 5. Implementation

A transformation system based on XML Schema correspondences has been implemented as extension to the Shinka Business Integration Platform [BIP], which provides a framework for WebServices-based integration of distributed applications. It is suited for integration in a HTTP-based request-response communication style as well as by using asynchronous message exchange over message queues.
Regardless of the underlying communication paradigm, the Shinka Business Integration Platform together with the transformation system permits the seemless integration of distributed, heterogeneous applications employing web-technologies as WSDL, XML Schema and SOAP.

The transformation system consists of the following components:

- a user interface for the declaration of schema correspondences between two schemata, currently a visual modeling tool for the definition of schema correspondences is under development;
- a rule composer which takes schema correspondences as input and derives a schema mapping represented in the intermediate transformation language;
- a transformer generator which translates the schema mapping to an executable transformer for the TraX API for Java [TraX]

» an extensible runtime transformation framework for the Shinka Business Integration Platform

## 6. Related Work

A common approach for the schema integration of federated information systems is to build mediators based on model correspondences. Given two or more information models, such correspondences state, which model components or submodels relate to each other (intensional aspect) and when correspondences between instances of these models can be established (extensional aspect). [Bu99b] proposes "a specification language for model correspondence assertions" based on OQL expressions.

The Web Service Modeling Framework [WSMF] aims at an amalgamation of semantic web and web service technologies towards so called "Semantic Web enabled Web Services". One aspect in this conext is the dynamic mediation of web services bypassing all kinds of interoperability problems. XML schema correspondences address the overcoming structural heterogeneity, which is one of those problems, and are thus suited for contributing to the vision of semantic web enabled web services.

XSL Transformations [XSLT] is a universal transformation language for XML documents. It comprises powerful transformation features applicable in a variety of contexts. XSLT stylesheets operate on a DOM-like tree data model of XML.

XML Query [XQuery], a query and transformation language for XML documents which is currently under development at the W3C. It has a scope comparable to XSLT but with more powerful query features. Like XSLT it is designed to be applicable for a broad range of contexts. The underlying data model of XML Query is similar to the intermediate transformation language although more complex.

## 7. Future Work

As mentioned before, the automatic recognition of correspondence by analyzing a given corpus of example data promises a significant simplification of the task of declaring schema correspondences. The effectiveness of this correspondence recognizer could be further improved if the schemata would be enriched by metadata describing the ontological context.
The development of a graphical user interface as a tool for modeling XML schema correspondences is another subject of ongoing development. Its main purpose is to allow the easy modeling of correspondences on a somewhat abstract, graphical level, which hides the syntactic aspects of XML schemata and message transformation. This is necessary since domain experts are usually not familiar with the underlying XML technology.

By facilitating type inference, such a tool could also automatically detect if a schema correspondence guarantees syntactic validity with respect to the given schemata, i.e. if the resulting transformation always transforms instances of the source schema into instances of the target schema. It is, however, a known problem (e.g. from the area of description logics) that performing type inferences can easily be NP-complete or even undecidable for a non-trivial type language.

## 8. Conclusion

In recent years, XML has become more and more accepted as the ideal format for information exchange between distributed real-world applications. In an ideal world, all applications would use the same schemata. In reality, however, the applications to be integrated often use different schemata, i.e. they use different syntactic formats for representing similar content.

Dynamic mediation of content and services in a semantic web therefore needs to overcome the difficulties of data format heterogeneity. For this purpose we introduced the notion of XML schema correspondences which describe the relations between the information models of two systems in a way that mediating between these systems can be efficiently automatized. The main idea is to explicitly model schema correspondences and to generate code from them which performs transformations between different XML formats on the fly.

One key to the success of this approach is the transformation language. It acts as an intermediate representation of transformation rules derived from schema correspondences and allows for the representation of transformation operations independent of a particular formalism, such as XSLT or XQuery. Nevertheless it is an appropriate basis for generation of transformers, e.g. of XSLT stylesheets.

This approach has been derived from real-world applications in the context of B2B integration and problems therein. Thus it is closely related to practical applications in this area and not primarily meant for striking goals related to the semantic web.

It is appropriate for different communication paradigms (point-to-point as well as publisher-subscriber) and integration styles (global/hierarchical as well as local/adaptor-like). Its added values with respect to other approaches are its strong relatedness to real-world applications, its high degree of abstraction and declarativity and its schema-driven approach.

## References

[BIP] *Business Integration Platform*, Whitepaper, Shinka Technologies, http://www.shinkatech.com/pdf/whitepaper.pdf

[Bu99a] S. Busse, U. Leser, R.-D. Kutsche, H. Weber: *Federated Information Systems: Concepts, Terminology and Architectures*, Forschungsberichte des Fachbereichs Informatik, TU Berlin, 1999.

[Bu99b] S. Busse: *A Specification Language for Model Correspondence Assertions*, Forschungsberichte des Fachbereichs Informatik, TU Berlin, 1999

[DOM2] J. Robie et al.: *Document Object Model (DOM) Level 2 Core Specification*, W3C Recommendation, 2000, http://www.w3.org/TR/DOM-Level-2-Core/

[RDF] O. Lassila et al.: *Resource Description Framework Model and Syntax Specification*, W3C Recommendation, 1999, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[SOAP] H. F. Nielsen et al.: *Simple Object Access Protocol Version 1.2*, W3C Working Draft, 2001, http://www.w3.org/TR/soap12-part0/

[TraX] Transformation API for XML, part of the Java-API for XML Processing, http://java.sun.com/xml/jaxp/

[WSMF] D. Fensel, C. Bussler: *The Web Service Modeling Framework*, Technology White Paper, Vrije Universiteit Amsterdam, http://www.cs.vu.nl/~dieter/wsmf/

[WG97] G. Wiederhold, M. Genesereth: *The Conceptual Basis for Mediation Services*, IEEE Expert, Vol. 12, No. 5, 1997

[WSDL] E. Christensen et al.: *Web Services Description Language 1.1*, W3C Note, 2001, http://www.w3.org/TR/wsdl.html

[XSchema] D. C. Fallside (Editor): *XML Schema, W3C Recommendation*, 2001, Part 1: Structures, http://www.w3.org/TR/xmlschema-0/

[XQuery] J. Robie et al.: *XML Query 1.0: An XML Query Language*, W3C Working Draft, 2001, http://www.w3.org/TR/xquery/

[XSLT] J. Clark (Editor): *XML Stylesheet Language Transformations (XSLT) Version 1.0*, W3C Recommendation, 1999, http://www.w3.org/TR/xslt