Schema-driven Customization of Web Services

S. Abiteboul INRIA
B. Amann Cedric-CNAM
J. Baumgarten INRIA
O. Benjelloun INRIA
F. Dang Ngoc INRIA
T. Milo * INRIA

1 Introduction

XML is becoming the universal format for data exchange between applications. Recently, the use of XML documents, where some of the data is given explicitly, while other parts are defined by programs that generate the relevant data, started gaining popularity [7, 8, 3]. We refer to such documents as *intensional documents*. We call *Materialization* the process of invoking *some of the programs* included in an XML document and replacing them by their results. We are particularly interested here in documents where the intensional part is given by embedding calls to Web services inside the documents. Web services [11] are emerging as a standard mean of publishing and accessing data on the Web, by providing a standard XML-based protocol to invoke remote programs over the Web (SOAP), along with a standard language to describe them (WSDL). Consequently, they furnish a uniform framework for describing intensional XML data, and were adopted for instance in [7, 8, 3]. *The goal of this demonstration is to advocate for the exchange of such intensional XML documents between applications, and to illustrate the new possibilities and the great flexibility they bring to application design.*

The demonstrated software was developed in the context of Active XML [3, 1, 9, 2], a peer-to-peer system that natively supports intensional documents and the declarative specification of Web services using XQuery [12]. We propose a new approach that allows one to control and customize the exchange of intensional documents between Active XML peers, but the same principles can be employed for exchanges between arbitrary Web applications.

When exchanged between two applications, intensional documents have a crucial property: since Web services can be called from anywhere on the Web, data can either be materialized before sending, or sent in its intensional form, thus leaving to the receiver the freedom to materialize the data if and when needed. More generally, a hybrid approach can be adopted, where some data is materialized by the sender before the document is sent, and some by the receiver. This choice may be influenced by various parameters, such as performance, capabilities, and security considerations. For instance, if communication is expensive, deferring the materialization to the receiver is preferable. On the other hand, a particular receiver may not be capable of invoking some service calls due, for instance, to limited access rights or security considerations. Therefore, this particular portion of the data needs to be materialized by the sender.

For purely extensional data, schemas (like DTD's or XML Schemas) are used to specify the desired format of the exchanged data. Similarly, we proposed in [9] to control the exchange of intensional data, and in particular the invocation of calls, using an extension of XML Schema we introduced, called XML Schema$_{int}$. The novelty is that these schemas do not only describe the required structure of standard extensional XML data, but also entail information about which parts of the data are allowed to be intensional, and which service calls may appear where in the documents.

Before sending a document, the sender must check if the data, in its current structure, matches the schema expected by the receiver. If not, the sender must perform the necessary service calls to transform the data into the desired structure, if possible. We call the sequence of these invoked calls a *rewriting*. A set of novel algorithms that allow the sender to determine, statically or dynamically, the required rewriting sequence is provided in [9]. *We argue that this new, dynamic, data exchange paradigm allows for great flexibility in application design, and in particular in the customization and use of Web services.*

Indeed, rather than building for each client a particular Web service that fits its needs/capabilities, the same service can be adapted to different purposes by adjusting – automatically via rewriting and without additional programming effort – the intensional output (resp. input) parameters to the client's (resp. service provider's) requirements. The demo proposed here aims at showing the power of this new data exchange

* On sabbatical from Tel-Aviv University.

Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003

paradigm and, in particular, the role played by intensional XML documents, XML Schema$_{int}$ schemas, and our document rewriting algorithms, in the development of customizable Web services.

Demonstration highlights We will demonstrate a peer-to-peer *news publication and syndication system*, and show how Web services provided by various peers can easily be customized using the above concepts. We consider the following setting: A number of news sources (newspaper websites, but also individual "weblogs") regularly publish some news stories. They share this information with others in a standard XML format, called RSS [10]. Clients can *periodically retrieve* news from the sources they are interested in, or *subscribe* to news feeds. So called *aggregators* are a special kind of peers who know of many news sources and let other clients ask queries and/or discover new sources. They also relay queries to other news sources and/or aggregators to answer a query.

We will show that when the Web services provided by the news sources/aggregators exchange intensional documents (controlled by the schemas mentioned above), the *same* service, e.g. querying, can be easily customized to be used by distinct kinds of participants, e.g. various client types or aggregators, with different requirements on the type of the input/output. Interactions between peers (including the exchange and rewriting of intensional data) will be tracked by a distributed logging system and displayed to illustrate the system's underlying computation.

The demonstrated news application is implemented on top of the Active XML system [3], which was demonstrated in VLDB'02 [1] and *is not the focus of the current demonstration*. Indeed, the most significant module for our demonstration is a new *Schema Enforcement* module, that controls the rewriting of the intensional data exchanged between the defined Web services and permits their seamless customization.

The remaining of this paper is structured as follows. Section 2 briefly describes intensional documents, schemas, and rewritings. Section 3 illustrates, through the news syndication demonstration scenario, how Web services exchanging intensional data are customized. Section 4 describes the implementation.

This work is clearly related to topics such as view materialization and data conversion, (e.g. [6, 4]). A review of related work is omitted here for space reasons, but can be found in [9].

2 Exchanging intensional XML data

In this section, we briefly introduce intensional documents, and outline the role of schemas and rewritings in the exchange of such documents.

The following is a fragment of an intensional XML document containing information on news channels. It provides, for each channel, its title and a list of news items. The data format is inspired by RSS [10]. In the first channel, the news items are given *intensionally*: the `<sc>` element represents a call to the `getNewsAbout` service of source `scripting.com`. When this web service is invoked, the returned news items are inserted into the document, replacing the `<sc>` element. In the second channel, the news items are given explicitly, except for the news story which is defined by a call to the `getStory` service. The full syntax, omitted here for space reasons, uses a particular namespace to differentiate service calls from the rest of data, and provides all the necessary parameters to invoke the services using SOAP.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="0.91">
  <channel>
    <title>Scripting news</title>
    <items>
      <sc>scripting.com/getNewsAbout("tech")</sc>
    </items>
  </channel>
  <channel>
    <title>Slashdot</title>
    <items>
      <item id="cx_ah_0218">
        <title>Google goes Blog-Crazy</title>
        <pubDate>Feb 18, 2003 10:36:03 GMT</pubDate>
        <description>
          Google just acquired Pyra labs, the company
          that makes Blogger.
        </description>
        <sc>slashdot.org/getStory("cx_ah_0218")</sc>
      </item> ...
    </items>
  </channel> ...
</rss>
```

We assume here that `slashdot.org` returns a `<story>` element with a textual content, while `scripting.com` returns a sequence of `<item>` elements with the above structure. This type information is typically provided by the WSDL description of Web services.

The following is a fragment of the XML Schema$_{int}$ describing the structure of such a document. The syntax is a natural extension of XML Schema [12], allowing to specify what type of service calls can appear in the intensional document, and where.

```
<xs:element name="channel">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="items" type="ItemList"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name="ItemList">
  <xs:sequence>
    <xs:choice>
      <xsi:serviceCallPattern ref="News"/>
      <xs:element name="item" type="Item"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="Item">
  <xs:sequence>
    <xs:element name="title" type="xs:string"/>
```

```
    <xs:element ref="pubDate" type="xs:dateTime"/>
    <xs:element ref="description" type="xs:string"/>
    <xs:choice>
      <xsi:serviceCall to="slashdot.org/getStory"/>
      <xs:element name="story" type="xs:string"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:NMTOKEN"/>
</xs:complexType>
```

This schema states that a `<channel>` element has two sub-elements: `<title>` and `<items>`. Items are given either explicitly, as `<item>` elements or intensionally, by a service call matching the `News` *service call pattern*. The latter, whose definition is omitted here for space reasons, allows to use any service call that returns a sequence of news `<item>` elements of the correct type. The `<story>` element can also be given either explicitly or intensionally, as a call to the `getStory` service of `slashdot.org`. Observe the difference in the definition of the service calls in `Item` and `ItemList`. The former states that the `Item` element may only contain a call to a particular service call (`slashdot.org/getStory`). The latter allows the use of any service call obeying the referenced pattern.

The validation process of an intensional XML document w.r.t. an XML schema$_{int}$ is similar to that of regular XML file w.r.t an XML Schema. The main difference is that now the embedded service calls have to match their specification, in terms of service provider and name, and possibly of the input/output schemas (in the case of service patterns).

Now, assume that we want to send the above document to an application that expects data in the standard RSS format, where the news items must be given extensionally, with only the `<story>` element possibly being intensional. Namely, the received documents should conform to a similar schema, except that the `ItemList` complex type is replaced by a simpler `ItemList2`, where the `choice` construct is omitted and only `<item>` elements are allowed. To conform to it, the sender needs to invoke the first call before transmitting the document.

As another example, assume that the above file needs to be sent to a "naive" client, which cannot invoke service calls. The latter expects a fully extensional XML document with a schema similar to the one given above, but containing extensional `ItemList3` and `Item3` types, where both `choice` constructs are eliminated. To meet this schema, the sender needs to invoke all the service calls appearing in the document, plus the new calls that may appear in the answers of these calls (e.g. for the first call).

While the above examples are rather simple – one could determine by a simple observation of the document and the target schema which calls need to be invoked – things may in general be much more complex. This is mainly because the relationship between the document and the target schema may be involved, with the results of the embedded service calls (as well as their parameters) containing themselves embedded service calls, and so on, recursively. Novel algorithms to determine the appropriate sequence of rewritings to make a document match the target schema can be found in [9].

Web services with intensional parameters For standard Web services, WSDL is used to describe the types of their input/output parameters using XML Schema. In a similar way, for Web services exchanging intensional data, we propose WSDL$_{int}$, that relies on XML Schema$_{int}$ to describe the types of their intensional input/output parameters. The service customization relies on a *Schema Enforcement* module. Before calling a service, this module checks if the supplied parameters comply to the schema expected by the service provider, and if not rewrite them accordingly. Similarly, before answering a service call, the module checks whether the computed answer conforms to the schema expected by the client and, if not, tries to rewrite it to the required structure.

If the service provider knows in advance its potential clients (and their expected schemas), it can generate, with practically no programming effort, a special interface for each client, by simply chaining the given service with a rewriting of its output to the appropriate schema. When the client requirements are not known in advance, or may change with time, a dynamic customization can be employed: the caller may supply the expected schema as one of the input parameters (either explicitly or intensionally, e.g. by a look-up to a directory of possible schemas), and the above rewriting process is applied to the output w.r.t the supplied schema. *In this demo, both kinds of Web service customizations will be demonstrated.*

3 Demonstration scenario

We consider two different kinds of peers: (a) News sources, and (b) news aggregators. For each one of them, we present a set of basic Web services, with intensional output and input parameters, and show how they can be customized for different clients via schema-based rewriting. We first consider intensional output, then intensional input parameters.

3.1 Intensional output

News sources provide news stories, using a basic Web service named `getStory`, which retrieves a story based on its identifier, and has the following signature:

```
getStory(<xs:simpleType ref="xs:string">) ->
        <xs:element name="story" type="xs:string">
```

News sources also allow users to search for news items by keywords [1], using the following service:

[1] More complex query languages, such as the one proposed by [5] could also be used.

```
getNewsAbout(<xs:simpleType ref="xs:string">) ->
             <xs:complexType ref="ItemList2">
```

This service returns an RSS-style list of news items (of the type `ItemList2` defined in the previous Section), where the items are given extensionally, except for the story, which can be intensional. A fully extensional variant of this service, aimed for instance at PDA's that download news for off-line reading, is easily provided by employing the *Schema Enforcement* module to rewrite the previous output to one that complies to the fully extensional `ItemList3` type.

A more complex scenario allows readers to specify a desired output type at call time, as a parameter of the service call. If a rewriting of the output exists that matches this schema, it will be applied before sending the result, otherwise an error message will be returned.

Aggregators act as "super-peers" in the network. They know a number of news sources they can use to answer user queries. They also know other aggregators, which can relay the queries to additional news sources and other aggregators, transitively. Like news sources, they provide a `getNewsAbout` Web service, but allow for a more intensional output, of type `ItemList`. When queried by simple news readers, the answer is rewritten, depending if the reader is a RSS customer or a PDA, into a `Itemlist2` or `Itemlist3` version respectively. On the other hand, when queried by other aggregators that prefer compact intensional answers which can be easily forwarded to other aggregators, no rewriting is performed, with the answer remaining as intensional as possible, preferably complying to the type below, which *requires* the information to be intensional.

```
<xs:complexType name="ItemList4">
  <xs:sequence>
    <xsi:serviceCallPattern ref="News"/>
  </xs:sequence>
</xs:complexType>
```

Note also that aggregators may not all have the same capabilities. For instance, some may not be able to recursively invoke the service calls they get in intensional answers. This is captured by having them supply, as an input parameter, a precise type for the answer of `getNewsAbout`, that matches their capabilities (e.g. return me only service calls that return extensional data).

3.2 Intensional input

So far, we considered the intensional output of services. To illustrate the power of intensional input parameters, we define a *continuous* version of the `getNewsAbout` service provided by news sources and aggregators. The clients call this service only once, to subscribe to a news feed. Then, they periodically get new information that matches their query (A dual service exists, to un-subscribe). Here, the input parameter is allowed to be given intensionally, so that the service provider can probe it, adjusting the answer to the parameter's current value. For instance, consider a mobile user whose physical location changes, and wants to get news about the town she is visiting. The zip-code of this town can be provided by a Web service running on her device. A call to this service will be passed as an intensional query parameter, and will be called by the news source in order to periodically send her the relevant local information.

4 Implementation

We use the Active XML system [3] as an implementation platform for defining the data and Web services of the news syndication application. The novel *Schema Enforcement* module acts as an XML Schema$_{int}$ validator. It implements the static and dynamic rewriting algorithms described in [9].

Each news source/aggregator peer has a browser-based interface. For news sources, the interface allows the creation/modification of channels, news stories and headlines. For aggregators, it allows the peers to manage their known news sources and aggregators. Both interfaces also support fine-tuning of the input/output schemas of the provided Web services, and track down the issued calls/returned answers and the rewriting process, through a distributed logging mechanism.

References

[1] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active XML: Peer-to-Peer Data and Web Services Integration (demo). In *Proc. of VLDB*, 2002.

[2] S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo. Dynamic XML documents with distribution and replication. In *Proc. of ACM SIGMOD*, 2003.

[3] Active XML. http://www-rocq.inria.fr/verso/Gemo/Projects/axml.

[4] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. In *Proc. of ACM SIGMOD*, pages 509–520, 2001.

[5] Edutella. http://edutella.jxta.org.

[6] H. Gupta. Selection of views to materialize in a data warehouse. In *Proc. of ICDT*, pages 98–112, 1997.

[7] Apache Jelly. http://jakarta.apache.org/commons/jelly.

[8] Macromedia Coldfusion MX. http://www.macromedia.com.

[9] T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, and F. Dang Ngoc. Exchanging Intensional XML Data. In *Proc. of ACM SIGMOD*, 2003.

[10] RSS 1.0 Specification. http://purl.org/rss/1.0.

[11] The W3C Web Services Activity. http://www.w3.org/2002/ws.

[12] The W3C XML Activity. http://www.w3.org/XML.