Active XML: Peer-to-Peer Data and Web Services Integration *

Serge Abiteboul
INRIA
and Xyleme

Omar Benjelloun
INRIA

Ioana Manolescu
INRIA

Tova Milo
INRIA
and Tel Aviv U.

Roger Weber
ETH Zurich

1 Introduction

Data integration has been extensively studied in the past, in the context of company infrastructures. In the context of the web, data integration faces new problems, due in particular to the heterogeneity of sources and their non-interoperability. These issues have been recently addressed, and partially resolved by (proposed) standards like XML, RDF, SOAP and WSDL, see [10]. Peer-to-peer architectures seem to be a promising solution to approach other issues raised by data integration over the web, namely its large scale and the independence and autonomy of sources. Peer-to-peer architectures are becoming increasingly popular, as they provide a decentralized infrastructure, in sync with the spirit of the web and that scales well to its size, as demonstrated by recent applications such as [4, 5]. We believe that this technology, together with the aforementioned standards, form the proper ground for data and service integration over the web. But what is still lacking is the "glue", and this is what Active XML (AXML, in short) provides: a declarative framework for peer-to-peer data and service integration at the scale of the web.

The AXML framework is centered around *AXML documents*, which are XML documents that may contain calls to web services. When calls included in an AXML document are fired, the latter is enriched by the corresponding results. In some sense, an AXML document can be seen as a (partially) materialized view, integrating plain XML data and dynamic data obtained from service calls. Documents with embedded calls is an old idea. For instance, in Microsoft Office XP, Smart Tags within XML documents can be linked to Microsoft's *.NET* platform for web services [8]. However, to our knowledge, AXML is the first proposal that actually turns calls to web services embedded in XML documents into a powerful tool for data integration.

The content of an AXML document is dynamic, since it is possible to specify *when* a service call should be activated (e.g. when needed, every hour, etc.), and for how long its result should be considered valid. This simple mechanism thus allows to capture and combine different styles of data integration such as warehousing and mediation. To fully take advantage of the use of services, AXML also allows calling (i) *continuous* services (that provide streams of answers) and (ii) services supporting intensional data (AXML document including service calls) as parameters and/or result. The latter feature leads to powerful, recursive integration schemes.

The AXML framework allows to use arbitrary SOAP-based web services. It also allows to specify and use more elaborate services, that may query and update AXML documents.

Demonstration highlights To illustrate the main aspects of AXML and show how the framework may be used for easy development of distributed, data-oriented applications, we define a peer-to-peer auctioning system in AXML and run it using an AXML prototype developed at INRIA.

Two AXML peers will interact, and will also call a standard web service such as eBay.net. The demonstration will feature both manual and automatic bidding. Interactions between peers will be tracked by a distributed logging system and displayed to illustrate the system's underlying computation. We will also show how an effective web-based user interface can be created, so that end-users can use the auctioning application from a standard browser.

The remaining of this paper is structured as follows: Section 2 briefly describes the AXML data and service integration framework. Then, Section 3 illustrates AXML through the auction demonstration scenario.

*This project is partially supported by EU IST project DB-Globe (IST 2001-32645)

Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

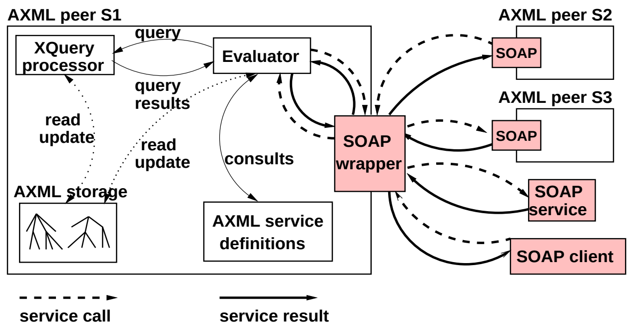

Figure 1: Outline of the AXML data and web services integration architecture.

2 Active XML

In this section, we describe the AXML integration architecture, and outline the main features of AXML data and services. More details on the syntax, formal semantics and an evaluation strategy can be found in [1].

2.1 AXML integration architecture

The AXML peer-to-peer integration architecture is shown in Figure 1. In general, we consider a *peer* to be any site that provides some XML data and/or web services and is able to call other peers. In particular, we distinguish *AXML peers* that contain AXML documents (which hold calls to other peers), and define AXML services. An AXML peer (like S_1 in Figure 1) thus contains an AXML storage module, and a catalog of the AXML services defined on this peer. The *Evaluator* module is responsible for activating the calls contained in the AXML documents. A call to a locally defined service entails a local computation, while a call to a service provided by another peer is encoded in a SOAP message by the SOAP wrapper, which transmits it to the service provider, and eventually receives the result as a SOAP message. In both cases, the service call result enriches the document, i.e., the document is updated to include the service call result.

Therefore, AXML peers are data and service integrators: They can integrate XML data obtained from various sources through service calls. But they are also service providers. The web services they define may be called by other XML peers as well as (with some restrictions) by peers that ignore AXML and simply know of SOAP. A call to an AXML peer is received by the SOAP wrapper, is decoded, processed, and the result (when obtained) is again sent via SOAP to the caller.

A first AXML peer prototype has been implemented in Java, relying on the Axis SOAP engine, and the Tomcat servlet engine, both by Apache [3]. For the purpose of the demonstration, we use a file-persistent DOM implementation as a repository and a simple home-made query processor [2].

2.2 AXML documents and services

AXML documents are XML documents that may include some special elements carrying the tag <sc> for *service call* [1]. The special interpretation attached to these elements is that they embed calls to web services. More precisely, an <sc> element encodes the URL of the site providing the service, the service name, the particular operation of the service that is invoked, and the appropriate parameters for the call. The latters are allowed to be XPath expressions. In this case, the XPaths expressions are instantiated in the document, and a service call is made for each combination of instantiated parameters. This feature turns out to greatly facilitate specifying integration. In the simplest case, the results of a service call are simply inserted in the document as siblings of the <sc> node. If unique identifiers (keys) exist in the document and in the service result, this insertion is done by the more elaborated mechanism of ID-based fusion. The details are omitted.

Defining services AXML allows not only to use existing web services in documents, but also to define new ones services on top of the enriched AXML documents. The definition of AXML web services relies on parameterized XML queries expressed in XQuery [11] extended with updates [9]. AXML services may query and update AXML or regular XML documents. The AXML service specification allows, in particular, the definition of continuous services, and of services with intensional input/output, which are detailed next.

Continuous services Simple services are similar to remote procedure calls. The service is called with some arguments, and eventually returns an answer. For the interesting class of *continuous* services, the interaction is more complex. Once a service call has been registered, a *stream of answers* is returned for this single service call. Streams of answers are encountered in many real-life applications: the stream of source updates for the maintenance of a data warehouse, the readings of a temperature sensor or surveillance system, answers returned by continuous queries or publish-subscribe systems like [6, 7], etc. Note that in the case of continuous services, the activation of the service call encapsulates a service subscription. The results received subsequently from the service are inserted in the caller document in a similar way as for simple calls.

Intensional service parameters and results In a perhaps more radical change to standard frameworks, we allow the parameters and result of a service call to contain service calls. A peer receiving a call with parameter containing service calls may have to activate the calls it includes before actually performing the service. Similarly, a service result may contain further service calls, which have to be activated by the

[1] The actual syntax relies on namespaces for clean separation, but is not used here for brievety.

site receiving the result. Thus, service call activations entail exchanging intensional data, and lead to a form of distributed computation.

The use of intensional parameters/results leads to security issues. For instance, a malicious user may force a site to perform a dangerous action by calling one of the site's services with an intensional parameter, that contains a call to a dangerous service. Besides security issues, the heterogeneous peer capabilities need to be taken into consideration. For instance, if the receiving site is a non-AXML peer (hence unable to activate the service calls embedded in the data) only fully extensional data must be sent. We address these two issues (security and source capability) using a simple model where each site publishes what it is willing to do (e.g., the services it is willing to call and the sites it is willing to serve). In the absence of such information as it is the case for non-AXML web services, we make conservative assumptions.

Controlling service call activation The moment when a service call should be activated is controlled by two special attributes of <sc> elements: mode and frequency. The frequency of a service call can be specified as a time interval (e.g., every week), a moment in time (e.g., on March 1st) or triggered by a change in the enclosing document. We say that a service call has *expired* when, according to its frequency attribute, it should be called again. The call mode may be either immediate, in which case the call is activated whenever it expires, or lazy, meaning that the expired call has to be activated when its result is needed (e.g., by a query over the AXML document, or by another service). The prototype we demonstrate only supports the immediate mode and time-related events.

Tightly related to the control of service call activation is the notion of life span of the returned data. Some data nodes in an AXML document are attached a special valid attribute, indicating for how long the data of this node remains valid (e.g. until March 1st, etc.) When the node becomes invalid, the entire subtree rooted at that node is removed from the document. The limit of the validity of the data may also be linked to the acquisition of new data, e.g., the data returned by a service call may be valid until this service call is re-executed.

3 Peer-to-peer auctions

We demonstrate the features of AXML by declaratively building a simple distributed auction system. Each AXML peer may get information about interesting items from his peers, place bids on others' auctions, and propose auctions for others to bid on.

An AXML auctions peer Each peer contains two AXML documents. The first one, called myAuctions.axml, is a standard XML document listing the auctions held by this particular peer and the current bids for these auctions. The second one, called knownAuctions.axml, is an AXML document that provides all the auctions that the peer is aware of. This document first contains this particular peer's auctions; and is augmented by auctions gathered from other AXML or non-AXML peers as specified by the document. This allows building a distributed, collective knowledge. In both documents, the auctions are grouped by categories. We assume that the category name and the auction id are unique identifiers for category and auction items, respectively. We will detail further on the structure of the two documents.

To allow for peers interaction, each peer provides a few web services: (1) getAuctions($c) gets as input a category name and returns the list of all auctions that the peer is aware of in this category; (2) getMyAuctions() simply returns the document myAuctions.axml; (3) placeBid($a,$b) places a new bid of amount $b for the auction $a of the receiving peer in the name of the calling peer; and (4) getHighestBid($a) gets as input the identifier of an auction held by the peer and returns the highest bid for that auction. Observe that a service call to (3) results in an update of myAuctions.xml. Note also that the service in (4) is *continuous*; once activated by a caller, it notifies the caller whenever a new higher bid is placed on this auction item. These services are defined by parameterized XQuery queries. We will see further an example of such a definition.

Gathering data The knownAuctions.axml document is defined and maintained as follows. First, the document lists the auctions categories that the peer is interested in. For each category, it gives the service calls that should be used to obtain auctions in this particular category. We will not detail how this list of service calls is constructed (It could typically be obtained as well from some web services.). For each call, the frequency of call activations, and the validity time of the acquired data are also specified. The document also contains a (local) call to getMyAuctions() to augment the above with the peer's own auctions.

A fraction of the document, at Peer "peer10", before the calls are activated, is given next. For brevity, we show only one category and omit the specification of the *frequency* and *validity* attributes.

```
<knownAuctions ID="peer10">
 <category name="Toys">
   <sc>eBay.net/getOffers("Toys")</sc>
   <sc>babel.org/translate( "Czech", "English",
        <sc>crystal.cz/getToys()< /sc></sc>
   <sc>peer25/getAuctions([../@name])</sc>
 </category>
   ...
 <sc>getMyAuctions()</ sc>
</knownAuctions>
```

The first <sc> element is a call to a standard, non AXML, eBay peer. The second <sc> element illustrates the use of intensional AXML parameters for a call: The getToys service of crystal.cz returns an answer in Czech. The translate service of babel.org translates from Czech to English. Its input file is specified here *intensionally* as an AXML document containing the call to getToys

Finally, the third <sc> element illustrates a call to another AXML auctions server. Rather than providing here explicitly the category name, the call parameter is given intensionally using a relative XPath expression. Observe that the called AXML peer also gathers its data from other peers, who themselves gather data from other peers, etc. So transitively, we potentially have access to all the reachable data. (The formal computation model is essentially a fixpoint computation). The last call in the document is the one that gets the peer's own auctions.

When the calls are activated, the document is enriched by the actual data they return. Note that due to the ID-based fusion mechanism, the common categories will be merged, resulting in a document having one entry per category name, containing both the local *and* acquired category auctions in that category.

As mentioned above, service call attributes are used to control the activation of the calls and the life span of the acquired data. This will be demonstrated by having one particular peer archiving all bids made on a distant auction of interest, whereas the other one will keep only the highest bid.

Automatic bidding The language features that control calls activation also allow us to easily build a simple automatic bidding mechanism. On the seller's site, the mechanism is the following. Once created, each auction lasts until a specific moment in time, after which the bidder that placed the highest bid is declared the winner. This winner is then notified and the auction is closed. To capture this, we use the following structure for the auction elements in the myAuctions.axml document. Each auction element contains the auction ID, the name of the peer holding the auction, a description of the item for sale, the list of the bids which have been placed for this item, and a call to a local service that will close the auction when the specified time has arrived.

```
<auction aID="1">
  <heldBy>peer10</heldBy>
  <item>Pink and yellow dragon...</item>
  <bid><who>peer5</who>
        <amount>$5</amount>
  </bid>
  <bid>...</bid>
  ...
  <sc mode="immediate">
      frequency="on March 1st">
      closeAuction([../])</sc>
</auction>
```

The service closeAuction gets as input an auction element, which is specified here using an XPath parameter ("../" enclosed in square brackets) that evaluates to the <auction> element itself. The role of closeAuction is the following: it selects the highest bid among the bids of the given auction, identifies who placed that bid, and appends to the auction element (i) a call to a notification service, (that will notify the winning bidder about the winning), and (ii) a new status element that declares that the auction is closed. The closeAuction service is defined by the following parameterized XQuery query:

```
let sc closeAuction($a) be
 for $b in $a/bid
 where $b/amount = max($a/bid/amount)
 return <sc mode="immediate" frequency="now">
            notifyWinner($b/who, $a/aID, $b/amount)
       </sc>
       <status> "closed" </status>
```

The return of the call to closeAuction includes in the document the status element and the service call to notifyWinner. Finally, the firing of the notifyWinner service call will result in the sending of the proper notification. The details are omitted. We omit as well here the definition of services for bidding and getting the highest current bid of an auction item.

References

[1] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active XML: a data-centric perspective of Web services. Submitted for publication, 2002.

[2] V. Aguilera. The X-OQL homepage. `http://www-rocq.inria.fr/ aguilera/xoql`.

[3] The Apache Software Foundation. `http://www.apache.org/`.

[4] The Kazaa Homepage. `http://www.kazaa.com`.

[5] The Morpheus homepage. `http://www.morpheus-os.com`.

[6] J. Naughton, D. DeWitt, and D. Maier et al. The Niagara Internet query system. In *IEEE Data Engineering Bulletin*, volume 24(2), 2001.

[7] B. Nguyen, S. Abiteboul, G. Cobena, and M. Preda. Monitoring XML data on the web. In *Proc. of ACM SIGMOD Conf.*, 2001.

[8] Jim Powell and Taylor Maxwell. Integrating Office XP Smart Tags with the Microsoft .NET Platform. `http://msdn.microsoft.com`.

[9] I. Tatarinov, Z. Ives, A. Levy, and D. Weld. Updating XML. In *Proc. of ACM SIGMOD Conf.*, 2001.

[10] The World Wide Web Consortium (W3C). `http://www.w3.org`.

[11] XQuery 1.0 : An XML Query Language. `http://www.w3.org/TR/xquery`.