Managing Distributed Workspaces with Active XML

Serge Abiteboul **Jérôme Baumgarten** **Angela Bonifati** **Grégory Cobéna**
Cosmin Cremarenco **Florin Drăgan** **Ioana Manolescu** **Tova Milo** **Nicoleta Preda**

INRIA, Gemo group
BP 105, 78153 Le Chesnay Cedex, France
firstname.lastname@inria.fr

1 Introduction

The tremendous evolution of the Web has brought the need for platforms allowing to easily deploy distributed data management applications. The current trend goes towards the de-centralization of such platforms, and in particular to peer-to-peer architectures. In this spirit, the Active XML system [1, 2, 3, 7] (AXML, for short) provides a peer-to-peer data integration platform, based on Web standards such as XML, and Web services [8]. The system is centered around AXML documents: XML documents where parts of the content is explicit XML data, whereas other parts are dynamically generated by calls to Web services on the same or on other peers. By including Web service calls, AXML documents already have an inherent form of distributed computation.

A higher level of distribution that also allows (fragments of) AXML documents and (some of the) Web services that they use to be distributed and/or replicated over several sites is highly desirable in today's Web architecture, and was addressed in [2]. *The goal of this demo is to illustrate the power of this novel distribution and replication paradigm for easy scalable development of Web applications, via a particular example - the management of a collaborative workspace.*

To explain this, we now describe in more details what AXML is, the new issues brought by the distribution and replication of AXML documents and services, and why this novel paradigm is particularly attractive as a foundation for the distributed management of collaborative workspaces.

The basic AXML model. As mentioned above, Active XML is a peer-to-peer platform for data and Web service integration. An AXML document is an XML document having a static part (XML data) and a dynamic part, consisting of calls to Web services. Such calls may contain actual XML *parameters*; the parameters are wrapped into an input message for the remote service. The XML data received in the service output message is inserted into the AXML document, which thus evolves through Web service call activation.

Each peer has some (A)XML documents and/or provides web services; communication among peers takes place through the invocation (on one peer) of the Web services provided by another peer, as shown in Figure 1(a). Regular non-(A)XML peers may also participate as long as they provide some Web services (like p_3 in Figure 1), and/or use some Web services provided by (A)XML peers. We distinguish AXML services, like s_1, s_2, s_3 in Figure 1, defined by parameterized, *declarative* XQuery queries, from *opaque* ones, provided by other peers and whose implementation is unknown, as, for example, s_4 in Figure 1. For opaque services, the only available information is their signature, provided, e.g., by a WSDL description. The AXML platform was demonstrated in [1].

Distributing and replicating AXML documents. In a recent work [2], we have considered the distribution and replication of AXML documents among several peers. The dynamic character of AXML documents, and the presence of declarative Web services, yields a set of new, complex possible replication scenarios. In a nutshell, replicating a fragment of AXML including a call to a declarative service may lead to replicating also the definition of the service, which in turn may lead to replicating the data used (queried) by this service. A sample AXML distribution and replication scenario is depicted in Figure 1(b). Document Doc_1 is distributed among p_1 and p_2; a *data link* still connects the two parts. Depending on the application needs, parent-child data links can be mono- or bi-directional, or may not appear at all. In Figure 1(a), p_1 has also taken a local copy of the declarative service s_2, as well as part of the document Doc_2 consulted by s_2. The local replica of s_2 has been *adapted* (re-formulated in XQuery) so that running on the local, partial Doc_2 replica, it produces the same results as the remote s_2 on the original document.

In the presence of data distribution and replication, the evaluation of a query over an AXML document may span several peers. For example, a query over Doc_1 at p_1 may cross the data link leading to the remote fragment at p_2. In a peer-to-peer setting like

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

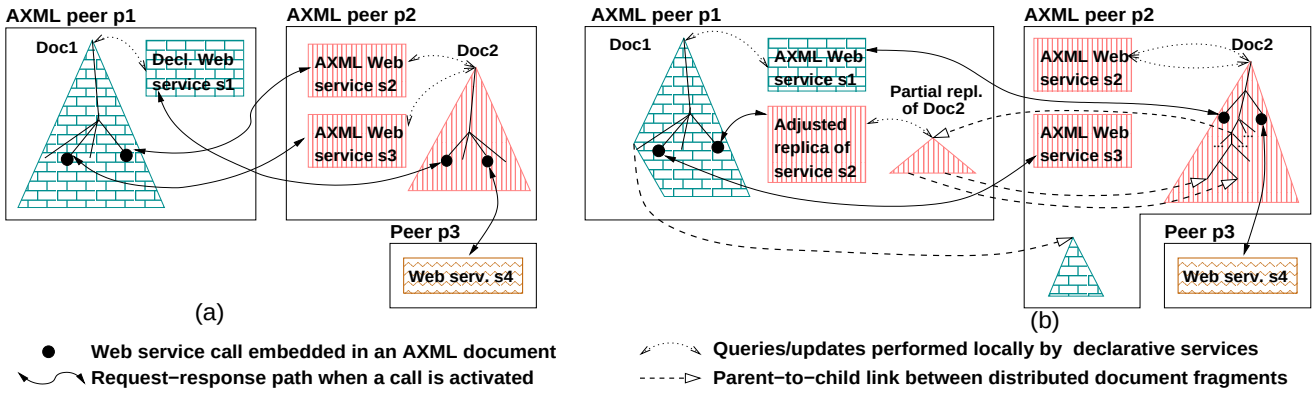

Figure 1: Peer interactions in the AXML model (a); influence of distribution and replication (b).

ours, no peer has global knowledge about which data or service is replicated where, and no single peer can impose some query processing to the others. Therefore, in [2] we have proposed a cost-based strategy for query evaluation, based on collaboration. Each involved peer attempts to minimize its own costs, taking into account its local information about data placement. For example, when processing a query over Doc_2, p_2 may realize that using the partial replica at p_1 would entail crossing data links back and forth, and may prefer to use only the original document.

To reduce the evaluation costs of queries/services often used by a peer, we provided in [2] an algorithm for automatically recommending to a given peer the data fragments and/or web services that the peer should replicate, given a set of constraints related to storage space, bandwidth etc. The partial replica of Doc_2, as well as the adjusted replica of s_2 in Figure 1, are a possible result of the recommendation algorithm run at p_1. In this case, the replication of s_3 and its related data was not recommended, since it is not found profitable enough.

We denote the AXML platform, empowered with distribution and replication capabilities, AXML^{DR}.

Managing collaborative workspaces with AXML. The purpose of this demonstration is to illustrate the usage of AXML^{DR}, for building a specific class of applications: collaborative management of a distributed workspace. Similar tools have long been investigated in the field of Computer-Supported Collaborative Work (CSCW) and Groupware. Our aim here is to demonstrate: *(i)* the flexibility, declarativeness, and ease-of-use of AXML^{DR} for specifying such applications (besides reducing the coding effort, declarative specification also allows for multiple optimization opportunities), and *(ii)* the unique data and service replication opportunities brought by AXML^{DR} in the context of such applications.

This paper is organized as follows. The next section presents our target application class: distributed workspaces. Section 3 details the proposed demonstration scenario: implementing a distributed information system for a real-estate agency, using AXML^{DR}. Section 4 describes some aspects of the implementation, and concludes. For space reasons, we delegate to [2] a more complete comparison with existing related work.

2 Managing distributed workspaces

In this section, we present the class of applications that we address in this demonstration: management of collaborative workspace, and illustrate the strength of the AXML^{DR} framework for these target applications.

In our setting, a *workspace* consists of a set of Active XML documents, and a set of (possibly declarative) Web services. A workspace is meant to be exploited by several users, co-operating to achieve a given common task. To that purpose, different users may:

- issue declarative queries on the documents;
- invoke the given Web services;
- trigger the execution of Web service calls embedded within Active XML documents, which amounts to an implicit document update;
- explicitly update the documents (through editing or declarative update).

Both implicit and explicit updates may modify the set of service calls contained in a document.

Usage scenario for a focused workspace. At workspace user, at any moment, is typically using only *part of* the cooperative workspace content. We denote by *focused workspace* of a user the portion of the workspace that the user is interested in. This may consist of: *(i)* a subset of the workspace services, possibly restricting the value ranges of some service call parameters, plus *(ii)* some partial replicas of documents that the user wants to query, or of documents used by *(iii)* the services in *(i)* above, or *(iv)* some service call embedded in the documents in *(ii)* or *(iii)* above.

To reduce the processing costs associated to the queries, updates, and web service calls included in the user's focused workspace, it may be useful to replicate the ingredients of this workspace on the user's machine. Note however that, due to various constraints

such as storage space, it might be the case that not all the ingredients can be replicated. Therefore, the evaluation of a query, update, or service call is distributed among the peer where it originated, and other peers.

To summarize, the life cycle of a focused workspace consists of the following steps:

Creation: the user selects, from the workspace documents and services, those that she wants to work with. Useful data fragments may be specified by declarative XQuery queries and/or by calls to existing services.

Configuration parameters: the user may provide some parameters describing the conditions under which she intends to work on her device. For example, she may specify that she intends to work disconnected from the network, or that outgoing communications from her device are expensive and should be avoided, or that CPU is a scarce resource because the device uses a battery etc.

Automatic replication: having declaratively specified her workspace, the user asks the system to automatically recommend data and/or declarative Web services to replicate locally for reducing the processing costs, and selects among the suggested configurations (if more then one is offered).

Usage: On the resulting configuration, the user now perform the processing associated to her task. This entails querying and modifying the local documents (through service calls or declarative updates) as well as distributed query processing and service invocation (if not all data/services were replicated).

Check-in: at some point, the user may want to check back her modified workspace fragment in the central repository. This may entail conflict resolution between the user's workspace version, and the repository. Many conflict resolution policies have been described in previous work on distribution and replication [4, 5, 6]. We implement a few simple policies [2], and allow for more complex ones to be plugged in our architecture as user-provided synchronization web services.

3 Application Scenario

This section illustrates the scenario we intend to demonstrate: the collaborative management of a real estate agency workspace. The components of the workspace are depicted in Figure 2. Three distinct documents contain information regarding properties for sale, clients seeking to buy a property, and a status summary for all properties currently managed by the agency. Calls to Web services embedded in the documents are represented by special <sc> elements. For lack of space, we use here a somewhat simplified syntax. The full syntax 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 [1]. In Figure 2, we grouped the workspace Web services under the documents they read/write. Each service in Figure 2 is easily defined

```
<properties>
  <property propID="344" type="studio">
    <status><sc>getStatus(344)</sc></status>
    <propID> 344 </propID>
    <assignedTo>Alice</assignedTo>
    <location>Paris</location>
    <picture>344.gif</picture>
    <descr>Magnificent view on Montmartre</descr>
    <ownerMail>tr678@myhome.com</ownerMail>
    <maybeClients><client><name>T.Jones</name>
        <addedBy>Bob</addedBy></client>...
    </maybeClients>
  </property>...
</properties>
```

Services accessing the <properties> document:
getProperties(assignedTo, Type, Location, Price);
getPicture(propID); **addClient**(propID, clientDescr, by)

```
<requests>
  <request>
    <type>Villa</type> <maxPrice>300,000</price>
    <handledBy>Jimmy</handledBy>
    <clientMail>you45@myhome.com</clientMail>
    <offers>
      <sc>getProperties(ANY,Villa,Paris,300,000)</sc>
    </offers>
  </request>
</requests>
```

Services accessing the <requests> document:
getRequests(reqDescr); **putRequest**(reqDescr, handledBy)

```
<allStatus>...
  <propStatus propID="344" updated="3/2/03"
    status="open"/>...
</allStatus>
```

Services accessing the <status> document:
getStatus(propID); **updStatus**(propID)

Figure 2: Workspace for the real estate agency

as an XQuery query (omitted for brevity). Clearly, the workspace documents may contain calls to other useful services, e.g., an estimateValue(propertyDescr) provided by a French government site, a map service, etc.

We demostrate the distribution of the workspace described in Figure 2 into many focused workspaces. In our scenario, on a daily basis, every real estate agent creates on his mobile device (laptop, PDA, or mobile phone) the focused workspace he/she is going to use that day. Let us track the lifecycle of the focused workspace belonging to real estate agent Alice, according to the steps defined in Section 2.

Focused workspace creation. Alice's focused workspace should reflect the tasks she intends to perform at the given day - namely the services she will use to accomplish the tasks and the needed data. Starting from a menu listing the available services and documents, Alice marks the ones she intends to use, restricting them to the relevant scope. Assume that Alice intends to work mainly on Parisian properties for sale assigned to her. She marks the getProperty service, restricting its parameters to getProperty("Alice", _,"Paris", _), thus indicating that her further calls will

have varing price and type parameters, but will regard only her assigned Parisian properties. Alice also checks addClient(_, _,"Alice") as she intends to attach new potential clients to Parisian properties, getRequest("Paris",) indicating she will use the service to look for requests matching the properties assigned to her, and finally getStatus(_) and updStatus(_) to check property status, resp. to register a done deal.

Configuration. Alice specifies here if she is to work on-line or disconnected, whether she prefers to always use data from the central agency repository (considered up-to-date) or when possible from her own machine (possible less up-to-date but cheaper and faster to obtain), whether during the day her device will work on battery etc. AXML^{DR} converts this information into local cost parameters. In general, a peer assigns his own set of *weights* to costs incurred by query processing on different peers; Alice may assign, e.g., a weight of 1 to costs incurred at the central agency peer and on her device, and 0 to all other weights.

Automatic replication. The algorithm running on Alice's peer analyzes the costs incurred by the Web service calls and queries Alice wants to make, and makes recommendations as follows. First, all Paris properties assigned to her are replicated on her peer. The addClient and getProperty services can now be run on Alice's partial replica of <properties>, therefore they are copied on her peer, and adapted to the copy. In our case, this consists of erasing the "Alice" and "Paris" selection predicates, since they have already been applied when taking the replica. Such adaptation is a limited form of view-based query rewriting. The data needed by getRequest is found too large to be replicated; Alice will have to query it remotely, from the real estate agency peer. Finally, assuming Alice wishes to use only the master copy of the status data, she will access this data only through the services getStatus and updStatus, provided by the central agency peer (not replicated on Alice's).

The demonstration displays the resulting configuration, and show how the replication decisions are affected when the cost weights, or the agent needs vary.

Usage. Alice can now use her focused workspace by running the services described above. While the details of peer-to-peer negociation and query execution are not interesting for Alice, we plan to show them in our demonstration, involving, for the sake of illustration, a few more peers. For example, assume Alice's colleague Bob takes in his focused workspace the master version of all status information for flats in Paris: status-related services invoked by Alice will now have to traverse Bob's focused workspace, too.

Check-in. Finally, Alice concludes her work and checks-in the data; we show the resulting workspace.

Besides the demonstration for the *agent* workspace, we also plan to show other roles in the real estate process, e.g., agency's *owner* or *client* workspaces.

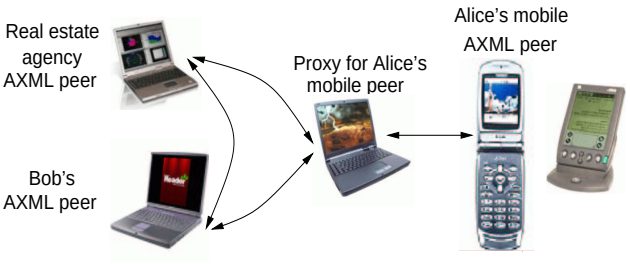

Figure 3: Outline of the demonstration configuration.

4 Implementation

The AXML^{DR} platform is built on top of the existing AXML one, described in [1]. The novel AXML^{DR} components are: an extension to our XML query processor enabling it to *optimize and evaluate distributed queries*; a module implementing the *automatic replication* algorithm; and a set of *AXML-specific Web services* for replicating data and services, intra-peer communication during query optimization and evaluation, and synchronization. Furthermore, we are adding to an AXML peer a *monitoring* capability, which proves useful for visualising the interactions among peers.

We plan to demonstrate the collaborative workspace on three physical devices: two laptops, for the AXML peers belonging to the real estate agency and to Bob, and a third small device for Alice's peer (Figure 3). The implementation of a "light" AXML peer for devices of the CLDC category (*C*onnected *L*imited *D*evice *C*onfiguration, e.g., mobile phones or PDAs) is ongoing. The limited storage, communication, and computation capabilities of such devices led us to introduce in the architecture an AXML proxy (either standalone or within an AXML peer), communicating with the phone via kXML-RPC, and with the other AXML peers via SOAP. On the CLDC device: *(i)* We store (simplified) AXML documents in the provided Record Management Store. *(ii)* We found no generic all-purpose XPath processor small enough; we are currently considering delegating all XPath processing to the proxy, or building a small, strongly simplified path query processor.

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 the Int'l VLDB Conf.*, 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 Conf.*, 2003.

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

[4] L. Chen and E. A. Rudensteiner. ACE-XQ: A CachE-aware XQuery Answering System. In *WebDB*, 2002.

[5] O. Kapitskaia, R. Ng, and D. Srivastava. Evolution and revolutions in LDAP directory caches. In *EDBT*, 2000.

[6] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. XMiddle: A data-sharing middleware for mobile computing. *Journal on Personal and Wireless Communications*, 2002.

[7] T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, and F. Dang Ngoc. Exchanging intensional XML documents. In *Proc. of ACM SIGMOD Conf.*, 2003.

[8] The W3C Web serv. working group: www.w3.org/2002/ws.