Active XML, Security and Access Control[*]

Serge Abiteboul[1 2] Omar Benjelloun[1] Bogdan Cautis[1] Tova Milo[3]

[1]INRIA-Futurs and LRI-Université Paris-Sud

[2]Xyleme SA

[3]Tel Aviv University

Abstract

XML and Web services are revolutioning the automatic management of distributed information, somewhat in the same way that HTML, Web browsers and search engines modified human access to world wide information. We argue in this paper that the combination of XML and Web services allows for a novel distributed data management paradigm, where the exchanged information mixes materialized and intensional, active, information.

We illustrate the flexibility of this approach by presenting Active XML, a language that is based on embedding Web service calls in XML data. We focus on two particular issues, namely security and access control.

1. Introduction

The field of distributed data management has centered for many years around the relational model. More recently, the Web has made the world wide (or intranet) publication of data much simpler, by relying on HTML as a standard language, universally understood by Web browsers, on plain-text search engines and on query forms. However, because HTML is more of a presentation language for documents than a data model, and because of limitations of the core HTTP protocol, the management of distributed information remained cumbersome. The situation is today dramatically improving with the introduction of XML and Web services. The Extensible Markup Language, XML [21], is a self-describing, semi-structured data model that is becoming the standard format for data exchange over the Web. Web services [24] provide an infrastructure for distributed computing at large, independently of any platform, system or programming language. Together, they provide the appropriate framework for distributed management of information.

Active XML (AXML, for short), is a declarative framework that harnesses these emerging standards for the integration and management of distributed Web data. An AXML document is an XML document where some of the data is given explicitly, while some portions are given only intensionally by means of embedded calls to Web services. By calling the services, one can obtain up-to-date information. In particular, AXML provides control over the activation of service calls both from the client side (pull) or from the server side (push). In AXML, all communications occur through service calls. Moreover, AXML encourages an approach to data exchange based on *active messages*, that is, messages that are AXML documents. The latter can be used both for the input parameters of service calls and for the returned results.

Choosing which parts of a should be given explicitly and which should be given in an active/intensional manner may be motivated and influenced by various parameters. These include logical considerations, such as knowledge enrichment or context awareness, and physical considerations like performance or capabilities.

[*]The research was partly funded by the RNTL Project e.dot and the French ACI MDP2P

In the present paper, we focus on the use of AXML for the management of security and access control in the context of distributed data exchange. Such aspects are typically considered in isolation from query processing. Indeed, the "historical" data models (e.g., the relational model) do not directly address such issues. This is perhaps acceptable in a centralized context, where a unique system is in charge of all data management aspects. It is much less so in the context of distributed data management, in particular when the various systems that are involved are autonomous. We will argue here that the model we recently proposed, namely Active XML, overcomes this separation.

It should be noted that the idea of mixing data and code is not new. Functions embedded in data were already present in relational systems [18] as stored procedures. Also, method calls form a key component of object-oriented databases [9]. In the Web context, scripting languages such as PHP or JSP have made popular the integration of (query) processing inside HTML or XML documents. Embedding calls to Web services in XML documents is just one step further, but is indeed a very important one. The novelty here is that since both XML and Web services are becoming standards, AXML documents can be universally understood, and therefore can be *exchanged*.

The rest of this paper is organized as follows. We first briefly recall some key aspects of XML, Web services (Section 2) and Active XML (Section 3). The following two sections informally discuss security and access control. The last section is a conclusion.

2. XML and Web services

In this section, we briefly discuss the context of our work, i.e., XML and Web services.

XML

XML is a new data exchange format promoted by the W3C [23] and widely adopted by industry. An XML document can be viewed as a labeled ordered tree, as seen on the example [1] of Figure 1. XML is becoming a lingua franca, or more precisely an agreed upon syntax, that most pieces of software can understand or will shortly do. Unlike HTML, XML does not provide any information about the document presentation. This is typically provided externally using a CSS or XSL stylesheet.

XML documents may be typed using a language called XML Schema [22]. A schema mainly enforces structural relationships between labels of elements in the document tree. For instance, it may request a *movie* element to consist of a *title*, zero or more *author*s and *reviews*. The typing proposed by XML Schema is very flexible, in the sense that it can describe, for instance, an HTML webpage, as well as a relational database instance, thus marrying the document world with the structured or semistructured world of databases. The presence of structure in XML documents enables the use of queries beyond keyword search, using query languages such as XPath or XQuery.

[1]We will see in the next section that this XML document is also an Active XML document.

```
<directory>
  <movies>
    <director>Hitchcock</director>
    <sc service="movies@allocine.com" >Hitchcock</sc>
    <movie> <title>Vertigo</title>
      <actor>J. Stewart</actor> <actor>K. Novak</actor>
      <reviews> <sc service="reviews@cine.com" >Vertigo</sc></reviews>
    </movie>
    <movie> <title>Psycho</title>
      <actor>N. Bates</actor>
      <reviews> <sc service="reviews@cine.com" >Psycho</sc></reviews>
    </movie>
  </movies>
</directory>
```

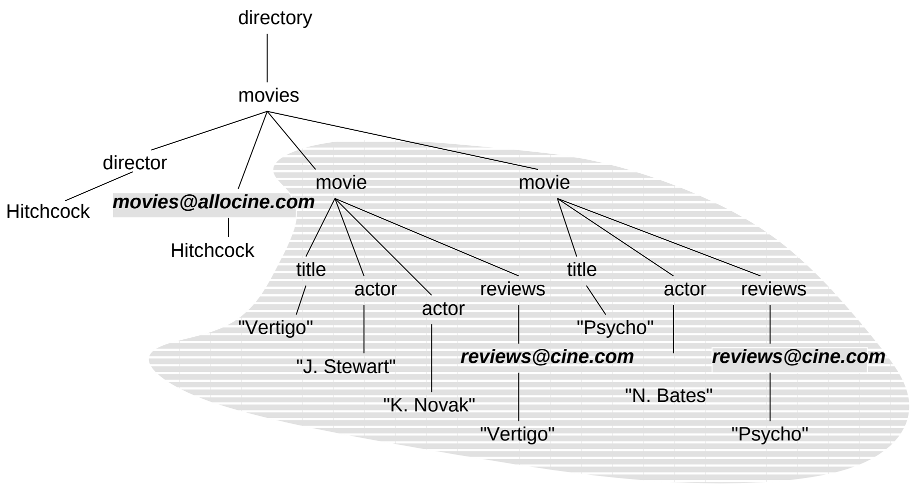

Figure 1: An Active XML document and its tree representation

Web Services

Web services form a major step in the evolution of the Web. Based on Web services, Web servers, that originally provide HTML pages for human consumption, become gateways to distributed resources. Although most of the hype around Web services comes from e-commerce, one of their main current uses is for the management of distributed information. If XML provides the data model, Web services provide the adequate abstraction level to describe the various actors in data management such as databases, wrappers or mediators and to manage the communications between them.

Web services in fact consist of an array of emerging standards. To find the desired service, one can query the yellow-pages: a UDDI [20] directory (Universal Discovery Description and Integration). Then, to understand how to interact with it, one relies on WSDL [25] (Web Service Definition Language), something like Corba's IDL. One can then access the service using SOAP [19], an XML-based lightweight protocol for the exchange of information. Of course, life is more complicated, so one often has to sequence operations (see Web Services Choreography [26]) and consider issues such as confidentiality, transactions, etc.

XML and Web services are nothing really new from a technical viewpoint. However, their use as a large scale infrastructure for data sharing and communication provides a new, trendy environment to utilize some old ideas in a new context. For instance, tree automata techniques regained interest as they best model essential manipulations on XML, like typing and querying.

This new distributed, setting for data and computation also raises many new challenges for computer science research in general, and data management in particular. For instance, data sources ought to be discovered and their informations integrated dynamically, and this may involve data restructuring and semantic mediation. Classical notions such as data consistency and query complexity must be redefined to accommodate the huge size of the Web and its fast speed of change.

3. Active XML

To illustrate the power of combining XML and Web services, we briefly describe Active XML, a framework based on the idea of embedding calls to Web services in XML documents. This section is based on works done in the context of the Active XML project [8].

In Active XML (AXML for short), parts of the data are given explicitly, while other parts consist of calls to Web services that generate more data. AXML is based on a P2P architecture, where each AXML peer is a repository of persistent AXML documents. It acts as a client, by activating Web service calls embedded in its documents, and also acts as a server, by providing Web services that correspond to queries or updates over its repository of documents. The activation of calls can be finely controlled to happen periodically, in reaction to some particular (in the style of database triggers), or in a "lazy" way, whenever it may contribute data to the answer of a query.

AXML is an XML dialect, as illustrated by the document in Figure 1. (Note that the syntax is simplified in the example for presentation purposes.) The `sc` elements are used to denote embedded service calls. Here, reviews are obtained from `cine.com`, and information about more Hitchcock movies may be obtained from `allocine.com`. The data obtained from the call to `allocine.com` corresponds to the shaded part of the tree. In case the relationship between data and the service call is maintained, we say that data is *guarded* by the call.

The data obtained by a call to a Web service may be viewed as intensional (it is originally not present). It may also be viewed as dynamic, since the same service call possibly returns different, data when called at different times. When a service call is activated, the data it returns is inserted in the document that contains it. Therefore, documents evolve in time as a consequence of call activations. Of particular importance is thus the decision to activate a particular service call. In some cases, this activation is decided by the peer hosting the document. For instance, a peer may decide to call a service only when the data it provides is requested by a user; the same peer may choose to refresh the data returned by another call on a periodic basis, say weekly. In other cases, the service provider may decide to send updates to the client, for instance because the latter registered to a subscription-based, continuous service.

A key aspect of this approach is that AXML peers exchange AXML documents, i.e., documents with embedded service calls. Let us highlight an essential difference between the exchange of regular XML data and that of AXML data. In frameworks such as Sun's JSP or PHP, dynamic data is supported by programming constructs embedded inside documents. Upon request, all the code is evaluated and replaced by its result to obtain a regular, fully materialized HTML or XML document. But since Active XML documents embed calls to Web services, and the latter provide a standardized interface, one does not need to materialize all the service calls before sending some data. Instead, a more flexible data exchange paradigm is possible, where the sender sends an XML document with embedded service calls (namely, an AXML document) and gives the receiver the freedom to materialize the data if and when needed.

More motivations for exchanging AXML data

We now briefly consider some extra motivations for peers to exchange active messages.

A first family of motivations concerns enabling a client to autonomously reuse pieces of information in the result of a service invocation without having to invoke the service again. Another main motivation is to provide dynamic information that adapts to changes over time. Also, an active answer allows the service to return directly some *partial* extensional information, along with some service calls to obtain more (in the style of answers of search engines). In general, active answers provides a more flexible paradigm for dealing with information such as summarization, context awareness, access to metadata, access to methods to enrich data, etc.

Furthermore, besides such *logical* motivations for using active answers, the notion of an active answer may be useful for guiding the evaluation of queries, when information and query processing are distributed. To see one aspect that typically plays an important role on the performance and quality of Web servers, consider the issue of *freshness*. Suppose some Web server S publishes a book catalog (as a Web service) and that a comparative shopping site S' accesses this catalog service regularly. The cache of S' will contain the retrieved data, but information such as book prices that change rapidly will often be stale. The Web server S may decide to make the answers of its catalog service active, e.g. by returning the price information intensionally (i.e. as service calls). Then, the book prices information can be refreshed by calling the corresponding services, without having to reload the entire catalog. This results in savings in communication.

Supporting techniques

We have briefly discussed XML and Web services and the advantages of exchanging active data. We presented AXML, that enables such a style of data exchange. To conclude this section, we mention three important issues in this setting, and recent works performed in these directions.

To call or not to call Suppose someone asks a query about the "Vertigo" movie. We may choose to call `cine.com` to obtain the reviews or not before sending the data. This decision may be guided by considerations such as performance, cost, access rights, security, etc. Now, if we choose to activate the service call, it may return a document with embedded service calls and we have to decide whether to activate those or not, and so on, recursively. We introduce in [15] a technique to decide whether some calls should be activated or not based on typing. First, a negotiation between the peers determines the schema of data to exchange. Then, some complex automata manipulation are used to cast the query answer to the type that has been agreed upon. The general problem has deep connections with alternating automata, i.e., automata alternating between universal and existential states [17].

Lazy service calls and query composition As mentioned earlier, it is possible in AXML to specify that a call is activated only when the data it returns may be needed, e.g., to answer a query. Suppose that a user has received some active document and wants to extract some information from it, by evaluating a query. A difficulty is then to decide whether the activation of a particular call is needed or not to answer that query. For instance, if someone asks for information about the actors of "The 39 steps" of Hitchcock, we need to call `allocine.com` to get more movies by this director. Furthermore, if this service is sophisticated enough, we may be able to ask only for information about that particular movie (i.e., to "push" the selection to the source). Algorithms for guiding the invocation of relevant calls, and pushing queries to them are presented in [1]. Some surprising connections between this problem and optimization techniques for deductive database and logic programming are exhibited in [7].

Cost model We describe in [5] a framework for the management of distribution and replication in the context of AXML. We introduce a cost model for query evaluation and show how it applies to user queries and service calls. In particular, we describe an algorithm that, for a given peer, chooses data and services that the peer should replicate in order to improve the efficiency of maintaining and querying its dynamic data. This is a first step towards controlling the use of intensional answers in a distributed setting.

Efficient query processing of structured and centralized data was made feasible by relational databases and its sound logical foundation [18, 6]. Deep connections with descriptive complexity have been exhibited [12, 6]. For the management of answers with active components, we are now at a stage where a field is building and a formal foundation is still in its infancy. Some recent results are presented in [3, 7]. The development of this foundation remains a main challenge for the researchers in the field.

4. Security

A main goal of the present paper is to show how AXML provides a uniform framework for addressing standard query processing issues as well as issues such as security for data management, that are typically considered separately. We believe that, in a Web context where various functionalities may be supported by different peers, it is of particular importance to provide an abstract model for distributed data management that captures these various viewpoints in a unique framework. Such a uniform model allows addressing issues such as data exchange protocol verification in a rigorous manner.

Security is a critical issue (arguably the most critical one) for Web applications. Not surprisingly, there is a lot of activity around XML, Web services and security. The W3C is promoting standardization efforts for security, e.g., for XML encryption [29], XML signature [27] or cryptographic keys [30]. The Security Assertion Markup Language [28] is an XML-based framework promoted by the Oasis consortium for exchanging security information.

Let us first illustrate by an example how basic security features may be supported in AXML. Suppose for instance that an AXML peer Bob wants to send a message to an AXML peer Alice, with a portion of the message encrypted. In AXML, the portion of the message to be encrypted is a sub-tree rooted at some particular node, say n. To encrypt it, Bob has to remove the children sub- trees of n , say t1, . . ., tm, and to replace them with their encrypted value.

In an AXML setting, encrypted XML data will be represented using the now standard XML Encryption with the following syntax:

```
<EncryptedData Id? Type? MimeType? Encoding?>
  <EncryptionMethod/>?
  <ds:KeyInfo>
    <EncryptedKey>?
    <AgreementMethod>?
    <ds:KeyName>?
    <ds:RetrievalMethod>?
    <ds:*>?
  </ds:KeyInfo>?
  <CipherData>
    <CipherValue>?
    <CipherReference URI?>?
  </CipherData>
  <EncryptionProperties>?
</EncryptedData>
```

We will rely on a public key encryption scheme. Each participant has a unique public/private key pair denoted PUK/PRK. As usual, the private key is private, while the public one is made accessible to the world, through the following service:

```
PublicKey@peer() -> string
```

A similar `PrivateKey@peer()` service exists, than can only be invoked by the peer itself. We also assume that each peer has available the following generic services, that respectively perform encryption and decryption.

```
encrypt@local(publicKey,data) -> encryptedData
decrypt@local(privateKey,encryptedData) -> data
```

Now, Bob first rewrites the data (to be sent) by replacing the children of n with:

`decrypt(PrivateKey@Alice()),encrypt(PublicKey@Alice(),t1,.,tm))`
Note that the resulting message has the same semantics (intensional content) as the original message. They only differ by the materialization of service calls. The exchange of information in AXML is guided by typing. The typing of the interface (typically the output type of a service) will specify that the information sent should not contain the service call ”encrypt”. Thus, before sending the data, the encrypt service call will have to be performed by Bob and Bob will send:

decrypt(PrivateKey@Alice(),E)

where E is the encrypted value of t1,.,tm, i.e. the result of:

encrypt(PublicKey@Alice(),t1,.,tm).

When receiving the message, Alice will have to decrypt it; indeed, she is the only one able to perform the decryption since she is the only one who can perform PrivateKey@Alice().

This is of course a simple setting. More complicated distributed exchange protocols may be supported by AXML. For instance, it is very simple to capture signatures (by switching public and private keys in the above example), authentication or delegation of privileges.

5. Access Control

Suppose we want to control the access to some resource, say F. Then we can simply hide F and let users access it via a controlled service G. So, for instance, to obtain F(a) for some input data a, a peer will call G(a,l) where l is some login information such as user name and password, possibly encrypted. The service G simply checks that the user has the proper access rights, eventually calls F, gets the result and returns it to the user. Note that this typically happens in a distributed setting with the user, the database and the access control manager residing on different peers. Note also that alternative strategies are also possible. For instance, the user may call the database that calls the access control manager to check that this particular user possesses the proper access rights.

To ensure the privacy of data, one may also choose to use a finer-grained control over the requests initiated by users. In [2], we adopted the GUP^{ster} approach [2], that unifies access control and source descriptions, by relying on a single query language to specify both. We use AXML and a single query rewriting mechanism to enforce them.

GUP^{ster} access control can be naturally incorporated into AXML documents by providing it as Web services. More precisely, we use *filtering services*, that enforce the access control rules on AXML data, given as their parameter. These services are used to protect some AXML data by filtering the queries that can be evaluated on them, according to a set of access control rules. Note that the protected data does not have to be sent extensionally as a parameter, but can be represented intensionally by a service call, and thus hidden from the filtering service.

Let us see in more details how this works. Suppose we want to evaluate a query on an AXML document for a particular user. This results in a call to some data guarded by a filtering service. The query is pushed to the filtering service. The filtering service rewrites it based on the access control rules defined for the particular user. The rewriting is performed by a GUP^{ster} service. Then, the resulting query is evaluated either directly (lazily) by the filtering service or via a request to the data data source. The choice of a specific evaluation strategy is controlled by the input/output types specified for the filtering services, using techniques introduced in [15].

6. Conclusion

The relational model has been an important breakthrough for data management in centralized information systems. It brought a cleanly formalized data model, with strong logical foundations. The SQL query language was essential for the adoption of this model, since it gave a syntax to define queries. With the advent of the Web, new data management paradigms are considered, to take advantage of what essentially is becoming an inherently distributed planet-scale information system.

Semistructured data, and its standard incarnation XML, are being recognized as the suitable model and language for data representation and exchange on the Web. XQuery, the query language for XML standardized by the W3C, is often advertised as the future "SQL" of the Web. However, XQuery is just one element of solution to the issue of "designing a language for Web data", since it primarily allows to query centralized collections of documents. In some sense, it misses to capture the distributed essence of the Web. With Active XML, we propose a first step towards a suitable data model and language for Web data management. The main contribution essentially consists in introducing intensional portions in semistructured documents, and enabling the exchange of this new kind of data. More precisely, we propose Active XML, a model for distributed data management, based on XML and Web services. In this model, AXML documents can integrate information from other Web sources, through embedded calls to Web services. AXML services open new perspectives for dynamic collaboration among systems on the Web, by enabling the exchange of AXML data.

The focus of the present paper was on aspects such as security or access control in an AXML setting. Typically, one would like to consider each *aspect* separately in the style of aspect-oriented programming [13]. For instance, one would prefer to ignore security issues when designing a data management application. We are currently working on extending AXML to do just that. The idea is to abstract some particular aspect such as security using rewrite rules. The application programmer may then ignore this aspect while designing the application. The rewrite rules are then in charge of automatically rewriting the data (for instance at the time it is exchanged) to meet the requirements of the aspect. This allows for a more modular approach. We have designed and implemented such an extension of AXML. We are currently validating it by considering various settings such as security management, access control, transaction processing or distributed query optimization.

References

[1] S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, N. Preda, Lazy Query Evaluation for Active XML, Sigmod 2004.

[2] S. Abiteboul, O. Benjelloun, B. Cautis, I. Fundulaki, T. Milo, A. Sahuguet, An Electronic Patient Record on Steroids : Distributed, Peer to Peer, Secure and Privacy Conscious (demo), VLDB 2004.

[3] S. Abiteboul, O. Benjelloun, T. Milo, Positive Active XML, In Proc. of ACM PODS, 2004., 2004.

[4] S. Abiteboul, P. Buneman, D. Suciu, Data on the Web, Morgan Kaufmann Publishers, 2000.

[5] S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, T. Milo, Active XML Documents with Distribution and Replication, ACM SIGMOD, 2003.

[6] S. Abiteboul, R. Hull, V. Vianu, Foundations of databases, Addison-Wesley, 1995.

[7] S. Abiteboul, T. Milo, Web Services meet Datalog, 2003, submitted.
[8] The AXML project, INRIA, http://activexml.net.
[9] The Object Database Standard: ODMG-93, editor R. G. G. Cattell, Morgan Kaufmann, San Mateo, California, 1994.
[10] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, Tata, Tree Automata Techniques and Applications, www.grappa.univ-lille3.fr/tata/
[11] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, T. D. Nguyen, Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities, Department of Computer Science, Rutgers University, 2002.
[12] N. Immerman, Descriptive Complexity, Springer 1998.
[13] G. Kiczales et al., Aspect-Oriented Programming, Proceedings European Conference on Object-Oriented Programming, 1997.
[14] M. Lenzerini, Data Integration, A Theoretical Perspective, ACM PODS 2002, Madison, Winsconsin, USA, 2002.
[15] T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, F. Dang Ngoc, Exchanging Intensional XML Data, ACM SIGMOD, 2003.
[16] M.T. Ozsu, P. Valduriez, Principles of Distributed Database Systems, Prentice-Hall, 1999.
[17] A. Muscholl, T. Schwentick, L. Segoufin, Active Context-Free Games, Symposium on Theoretical Aspects of Computer Science, 2004.
[18] J.D. Ullman, Principles of Database and Knowledge Base Systems, Volume I, II, Computer Science Press, 1988.
[19] The SOAP Specification, version 1.2, http://www.w3.org/TR/soap12/
[20] Universal Description, Discovery and Integration of Web Services (UDDI), http://www.uddi.org/
[21] The Extensible Markup Language (XML), http://www.w3.org/XML/
[22] XML Typing Language (XML Schema), http://www.w3.org/XML/Schema
[23] The World Wide Web Consortium (W3C), http://www.w3.org/
[24] The W3C Web Services Activity, http://www.w3.org/2002/ws/
[25] The Web Services Description Language (WSDL), http://www.w3.org/TR/wsdl/
[26] W3C, Web Services Choreography, http://www.w3.org/2002/ws/chor/
[27] W3C XML Signature specification, www.w3.org/TR/xmldsig-core
[28] The Security Assertion Markup Language, www.oasis-open.org/committees/security
[29] W3C XML Encryption specification, www.w3.org/TR/xmlenc-core
[30] W3C XML Key Management specification, www.w3.org/TR/xkms2