A Calculus and Algebra for Distributed Data Management*

Serge Abiteboul

INRIA-Futurs, Orsay & Univ. Paris 11
firstname.lastname@inria.fr

Abstract. The sharing of content by communities of users (e.g., scientists) in a P2P context remains cumbersome. We argue that main reasons for this is the lack of calculus and algebra for distributed data management. We present the ActiveXML language that extends the XML language with features to handle distribution. More precisely, ActiveXML documents are XML documents with a special syntax for specifying the embedding of Web service calls, e.g. XML queries such as XQueries. We also present ActiveXML algebra that extends ActiveXML notably with explicit control of data exchanges. ActiveXML algebra allows describing query plans, and exchanging them between peers.

1 Introduction

The field of distributed data management [17] has centered for many years around the relational model. More recently, the Web has made the world wide and intranet publication of data much simpler, by relying on HTML, Web browsers, plain-text search engines and query forms. The situation has also dramatically improved with the introduction of XML [22] and Web services [25]. Together, these two standards provide an infrastructure for distributed computing at large, independent of any platform, system or programming language, i.e., the appropriate framework for distributed management of information. However, the sharing of content by communities of users (e.g., scientists) in a P2P context remains cumbersome. We argue that main reasons for this is the lack of calculus and algebra for distributed data management and propose such languages based on Web standards, namely XML and Web services.

In [8], we propose the *data ring* that can be seen as a network analogue of a database or a content warehouse. The vision is to build a P2P middleware system that can be used by a community of non-experts, such as scientists, to build content sharing communities in a declarative fashion. Essentially, a peer joins a data ring by specifying which data (or services in general) are to be shared, without having to specify a schema for the data, load it in a store, create any indices on it, or specify anything complex regarding its distribution. The data ring enables users to perform declarative queries over the aggregated

* This work has been partially supported by the ANR Project WebContent and the EC project Edos [13] on the development and distribution of open source software.

data, and becomes responsible for reorganizing the physical storage of data and for controlling its distribution. Thus a primary focus of the data ring is *simplicity of use*. To achieve these goals, we identified a number of challenges:

Self-administration Since the users of the data ring are non-expert, the deployment of the ring and its administration should be almost effort-less. This means that a number of tasks such as the selection of access structures (indices) or the gathering of the statistics to be used by optimizers have to be fully automatic.

File management Since a large part of the data is going to reside in file systems, we need very efficient processing and *optimization* of queries over files, including for instance the automatic selection of specific access structures over file collections.

Query language To facilitate the exploitation of the ring by non-experts, the interfaces have to be mostly graphical and require the minimum expertise. They therefore must be based on declarative languages (calculus) in the style of relational calculus, rather than on languages such as Java or Ajax that require programming skills.

Query optimization Query optimization has, by nature, to be distributed and peers should be able to exchange query plans. This motivates adopting an algebra for describing distributed query plans interleaving query optimization, query evaluation, and possibly, error recovery and transaction processing.

We present ActiveXML, a declarative framework that harnesses XML and Web services for the integration and management of distributed data. An ActiveXML document is an XML document where some of the data is given explicitly, while other portions are given only intensionally by means of embedded calls to Web services, typically XML queries. By calling the services, one can obtain up-to-date information. In particular, ActiveXML provides control over the activation of service calls both from the client side (pull) or from the server side (push).

It should be noted that the idea of mixing data and code is not new, e.g., stored procedures in relational systems [19], method calls in object-oriented databases [10], and queries in scripting languages such as PHP. The novelty is that since both XML and Web services are standards, ActiveXML documents can be universally understood, and therefore can be universally *exchanged*.

We also present the ActiveXML algebra that extends ActiveXML in two main directions: (i) with generic services that can be supported by several peers (e.g., query services), (ii) with explicit control of the evaluation of ActiveXML documents (eval operator) and of data exchange (send and receive operators). The algebra can be used to describe query (evaluation) plans. Using rewrite rules, query plans may be optimized in a standard way. More fundamentally, the query plans are distributed and can be exchanged between peers. Thus the tasks of query evaluation and optimization can be distributed among the peers of the network.

The ActiveXML project has been going on for several years. A system is now available as open source [9]. In [16], a technique to decide whether or not

calls should be activated based on typing is introduced. The general problem has deep connections with tree automata [12] and alternating automata, i.e., automata alternating between universal and existential states [18]. Optimization issues in the context of ActiveXML are presented in [2]. In [5], a framework for managing distribution and replication in the context of ActiveXML is considered. Foundations of ActiveXML are studied in [3]. A preliminary version of the algebra appeared in [7].

We conclude this introduction by a brief discussion of XML and Web services.

```
<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>
```

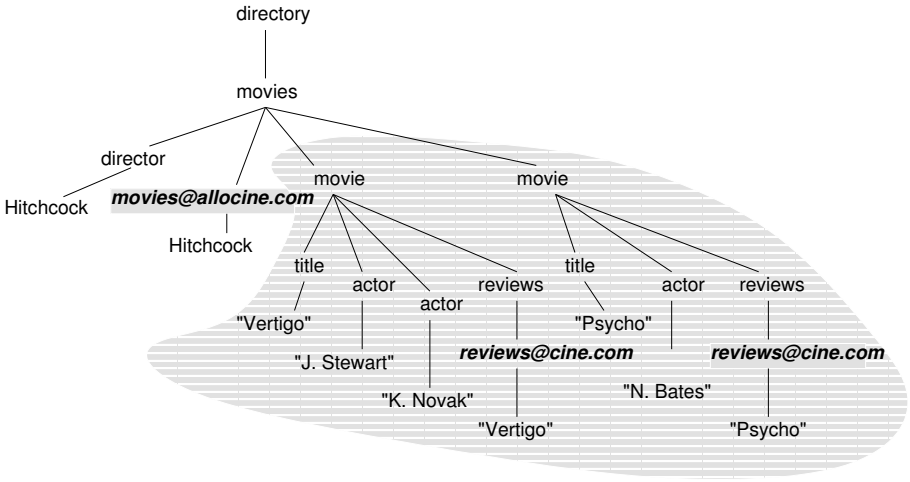

Fig. 1. An ActiveXML document and its tree representation

XML and Web services. XML is a semistructured data exchange format [4] promoted by the Word-Wide-Web Consortium and widely adopted by industry.

An XML document can be viewed as a labeled, unranked, ordered tree, as seen in the example[1] of Figure 1 (ignoring the grey area for now). Unlike HTML, XML does not provide any information about the document presentation. This is typically provided externally using a CSS or XSL style-sheet. XML documents may be typed, e.g., using XML Schema [23], and may be queried using query languages such as XPath or XQuery [24]. Web services consist of an array of emerging standards. For instance, to find a desired service, one can query a UDDI [21] directory. To understand how to interact with the service, one relies on WSDL [26], something like Corba's IDL. One can then access the service using SOAP [20], an XML-based lightweight protocol for the exchange of information.

The article is organized as follows. The calculus is discussed in Section 2 and the algebra in Section 3. The last section is a conclusion.

2 A stream calculus: ActiveXML

In this section, we briefly describe ActiveXML. Details may be found from [9] as well as papers on ActiveXML and the open-source code of an ActiveXML peer.

The success of the relational model essentially comes from the combination of a declarative language (relational calculus), an equivalent relational algebra, and optimization techniques based on rewrite rules. There have been a number of extensions such as object databases, but the classical pattern (calculus, algebra, rewrite rules) proved its robustness. It should also be adopted in the data ring context. However, the situation is essentially different (distributed vs. centralized, semi-structured vs. very structured) so requires a complete overhauling of the languages. We present a calculus for distributed semi-structured data in this section and an algebra in Section 3. In both cases, we insist on the features that we believe are fundamental for such languages.

We believe that to support effectively the loose integration paradigm of data, one essential aspect is the seamless transition between explicit and intentional data. One should not have to distinguish between extensional data (e.g., XML or HTML pages) and intensional data (e.g., access to a relational database provided by a Web service). As an example, consider the query "give me the name and phone number of the CEO of the Gismo company". Answering this query may require first finding the name of that CEO in an XML collection of company synopses, finding the service that exports the phone book of Gismo Inc, and finally calling this service with the name of this CEO. The query can be answered only (a) because we have a logical description of the resources, and (b) because based on that, we have derived a distributed query plan.

ActiveXML was designed to capture such issues. An ActiveXML document is an XML document where certain elements denote embedded calls to Web services. For instance, the company synopsis may contain the CEO phone number as a Web service call. The service calls embedded in the document provide intensional data in the sense of deductive databases [6]. Now suppose that the

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

phone number of the CEO changes, then the second time we call the service, the result changes. So, the home page of the company that includes this service call changes. Thus the embedding of service calls is also capturing active data in the sense of active databases [11].

Note that the use of intensional information is quite standard on the Web, e.g. in PHP-mySQL. It is also common in databases, see object or deductive databases. The main novelty is that the intensional data is provided by a Web service. Thus the corresponding service calls may be activated by any peer and do not have to be evaluated prior to sending the document.

In what sense can this be viewed as a calculus for distributed semi-structured data? First, we rely on some calculus for *local* semi-structured data. From a practical viewpoint, we can use the standard declarative language, XQuery. But one could use any calculus over XML as well. ActiveXML provides the support for handling distribution. The interaction with local queries is achieved by using query services. In some sense, the resulting language may be viewed as a deductive database language such as datalog [6] with XQuery playing the role of the single datalog rule and with ActiveXML acting as the "glue" between the rules, i.e., as the datalog program. Negation may be handled in any standard way [6]. Clearly, distribution introduces new issues with respect to evaluation and optimization, notably the detection of termination [1].

Henceforth, we assume that every peer exports its resources in the form of ActiveXML documents or Web services. The logical layer thus consists of a set of ActiveXML documents and services and their owning peers. The external layer will be dealing with the semantics (e.g., ontologies), but this aspect will be ignored here. A computation will consist in local processing and exchanging such documents.

ActiveXML is an XML dialect, as illustrated by the document in Figure 1. (Note that the syntax is simplified in the example for purposes of presentation.) 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 by a call to a Web service may be viewed as intensional, as it is not originally 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 returned is inserted in the document that contains the call. Therefore, documents evolve in time as a consequence of call activations. Of particular importance is thus the decision to activate a particular service call.

Two aspects are essential to the framework and motivate basing it on XML streams (as in ActiveXML) and not simply on XML documents:

Push vs. Pull In pull mode, a query-service is called to obtain information. But we are often interested on the Web in query subscription. The result of a subscription is typically a stream of answers, e.g., notifications of certain events of interest. A company synopsis may include such a service to, for instance, obtain the news of the company. Such a subscription feature is also essential for supporting a number of functionalities ranging from P2P

monitoring, to synchronization and reconciliation of replicas, or gathering statistics.

Recursion The embedded service calls may be seen as views in the spirit of those found at the core of deductive databases. In classical deductive databases, recursion comes from data relationships and recursive queries such as *ancestor*. In our setting, recursion kicks in similarly and also from the XPATH // primitive. But more fundamentally, recursion comes from the graph nature of the Web: site1 calls site2 that calls site3 that calls site1, etc. Indeed, the use of recursive query processing techniques in P2P contexts has been recently highlighted in several works in topics as different as message rooting on the Web [15] and error diagnosis in telecom networks [1]. Now, recursive query processing clearly requires the use of streams.

The basis of a theory proposed in [3, 1] makes two fundamental simplifying assumptions:

set-oriented The ordering in XML is a real cause of difficulty. We assume that the documents are labeled, unranked, *unordered* trees.

Query-services If the services are black boxes, there is little reasoning one can do about particular computations. We assume that the queries are defined logically (e.g., by conjunctions of tree pattern queries over the documents.)

Since documents contain intensional data (views), this result in a setting quite close to deductive databases. In [3], positive results are exhibited for limited query languages. They are obtained by combining techniques from deductive databases (such as Query-sub-Query) and from tree automata.

3 A stream algebra

Besides the logical level, our thesis is that a language in the style of ActiveXML should also serve as the basis for the physical model. In particular, the use of streams is unavoidable: see trivially, how answers are returned by Google or try to send 100K in a standard Web service without obtaining a *timeout*. As shown in a recent work [7], distributed query evaluation and optimization can be naturally captured using ActiveXML algebraic expressions, based on the exchange of distributed query execution plans. The expressions include standard algebraic XML operations and send/receive operators, all over XML streams. Note that these may be seen as particular workflow descriptions, very particular ones of a strong database flavor. Thus, we propose that the physical model be based on a the ActiveXML algebra [7].

The algebraic evaluation of queries is performed by collaborating query processors installed on different peers exchanging ActiveXML data in a streaming manner. Query optimization is performed also in a distributed manner by algebraic query rewriting. Standard distributed query optimization techniques can all be described using the proposed framework and simple rewrite rules in the language.

The ActiveXML algebra is an extension of the ActiveXML language with two main features: (i) generic data and services and (ii) a more explicit control of execution (e.g., eval) and distribution (send/receive). *Generic* data and services are data and services available on several sites, an essential feature to capture replication and the fact that a query service may be evaluated by any peer with query processing facilities (see [5]). We also provide the capability to explicitly control the shipping of data and queries, an essential feature to specify the delegation of computations (see [1]).

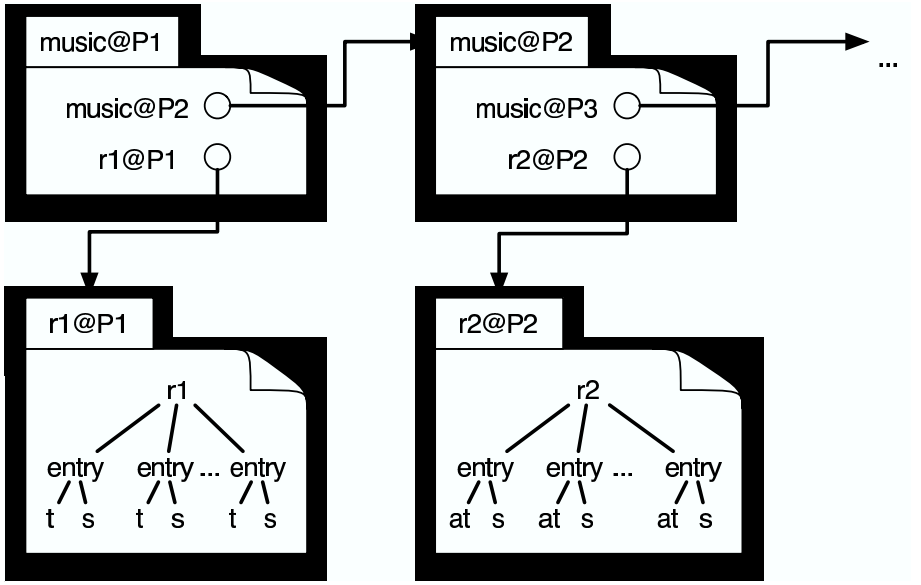

Fig. 2. A graphical representation of ActiveXML data.

An example will best illustrate this principle. Consider the data described in Figure 2. We use here a visual representation of ActiveXML documents. Peer $p1$ and $p2$ have their own collections of music with metadata described in relations $r1, r2$, respectively. Peer $p1$ knows about s(ingers) and t(itles), whereas $p2$ knows about s(ingers) and a(lbum) t(itles). Peer $p1$ also knows that $p2$ has some music; $p2$ knows that $p3$ (not shown here) has some; $p3$ knows $p4$, etc. The metadata of $p3, p4, p5$ are organized as that of $p1$. The actual texts underneath the tags s, t, at are not shown. Now suppose that $p1$ wants to get the titles of songs by Carla Bruni. Figure 3 shows three different query plans. Each box describes some peer computation. Query Plan (a) is the one that would result from an evaluation of the query without optimization, i.e., from applying the pure semantics of ActiveXML. Query plan (b) results from pushing selections, while Query plan (c) is obtained by also optimizing data transfers (cutting some middle persons

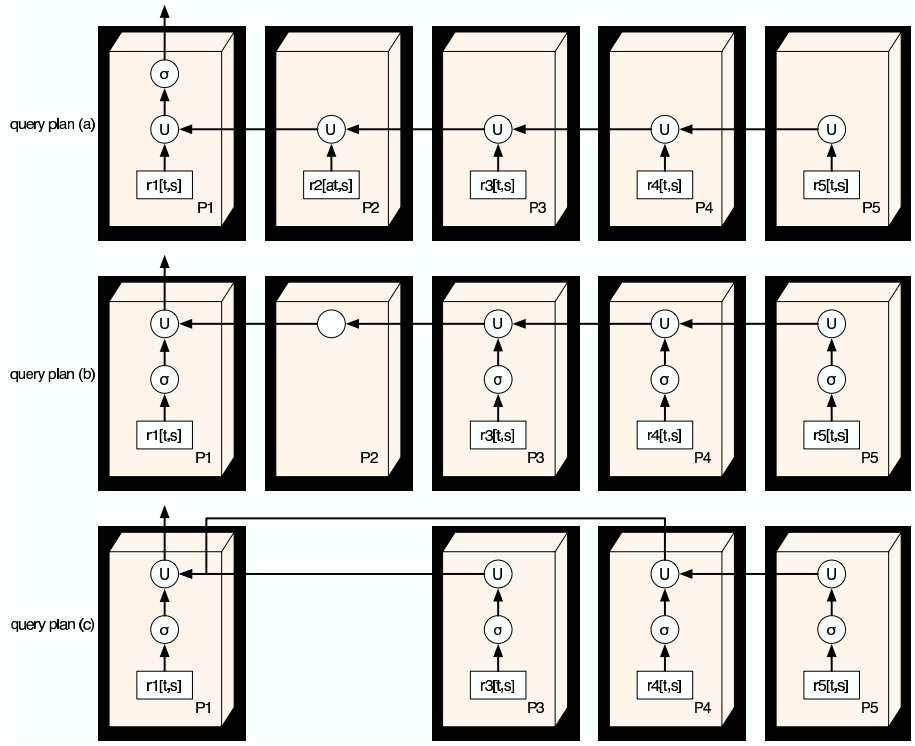

Fig. 3. Three equivalent distributed query plans.

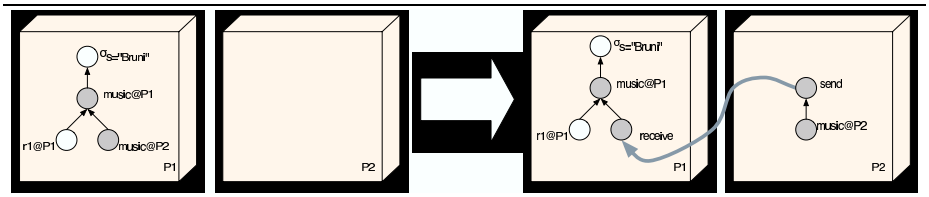

Fig. 4. An algebraic rewriting.

in data transmissions). One (particularly interesting) rewrite rule is illustrated in Figure 4. Consider only the shaded nodes. To perform the evaluation, an external service call is replaced by a receive node and remotely a computation is activated. It is requested that its result be sent to the location of the receive node. The communication is asynchronous.

We can make the following observations:

1. Peers 1 and 2 can already be producing answers, while Peer 3 is still optimizing the request it receives, while Peer 5 is still not even aware of the query. This is illustrating the need for streaming, Peer 2 can send answers to Peer 1 before obtaining the entire data she has to transmit.
2. Each peer separately receives a request and is fully in charge of evaluating it. (Some optimization guidelines may be provided as well.) For instance Peer 2 receives a query where she cannot really contribute and may decide to cut herself out of it to ask Peer 3 to evaluate its part and send the result directly to Peer 1.
3. We assumed so far that the peer cooperate to evaluate a query. Think now that the goal is to support a subscription. Then the same plans apply. Suppose a new song of Carla Bruni is entered in Site 3. Then it is sent to Site 1 (with Query Plan (c)), then produced as a new answer unless this title has already been produced.

In all cases, a query (subscription) for the songs of Carla Bruni (at the logical layer) is translated to a distributed plan (at the physical layer). Observe that the physical plan is essentially a workflow of Web services (i.e., an ActiveXML document), where the services encapsulate the different plan operators and the respective locations encode the distribution of computation and the flow of data. The main idea therefore is that the complete plan itself (or a portion of it), along with its current state of execution, can be described as an ActiveXML document, which in turn can be exchanged between peers in order to support query optimization and error recovery in a distributed fashion.

Another important element in the Figure 3 is the distinction between local query evaluation (inside each box) that is the responsibility of a local system, perhaps a relational system, and global query evaluation. The functional architecture of a peer query processor is shown in Figure 5. See the various components and in particular the local query optimizer and the local component performing global query optimization that collaborates with other peers to perform global query optimization. Essentially, this separation leads to physical plans that combine local query processing with distributed evaluation. Clearly, a collaboration between the two systems (local and global) is preferable but is unlikely to be widespread in the near future. This implies that we will have to view the local query optimizers as boxes with possibly different querying capabilities, in the same vein as mediation systems [14].

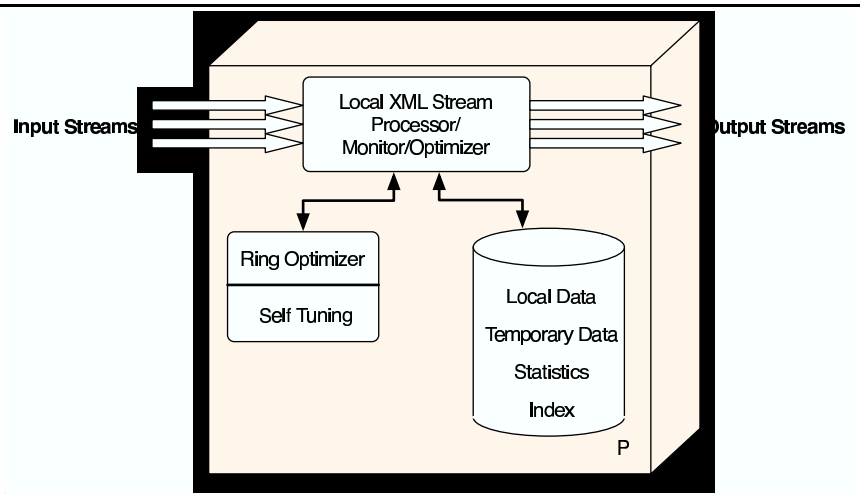

Fig. 5. Functional architecture.

4 Conclusion

It is not necessary to insist on the importance of distributed data management. Recent years have seen the arrival of a number of software tools that participate in such activity: structured p2p network such as Chord or Pastry, XML repositories such as Xyleme or DBMonet, file sharing systems such as BitTorrent or Kazaa, distributed storage systems such as OceanStore or Google File System. content delivery network such as Coral or Akamai, multicast systems such as Bullet or Avalanche, Pub/Sub system such as Scribe or Hyper, application platform suites as proposed by Sun or Oracle for integrating software components, data integration as provided in warehouse or mediator systems.

A formal foundation for distributed data management is still to come. The purpose of the present paper was not to advertise particular languages that close the issue, but rather to encourage researchers to work in this area. ActiveXML and ActiveXML algebra were used to illustrate aspects that, we believe, a calculus and an algebra for such a context should stress.

Acknowledgments The material presented in this paper comes from joint works with a number of colleagues from the projects that have been mentioned and most notably, Omar Benjelloun and Tova Milo for ActiveXML, Ioana Manolescu for ActiveXML Algebra, and Alkis Polyzotis for the Data Ring.

References

1. S. Abiteboul, Z. Abrams, S. Haar, and T. Milo. Diagnosis of asynchronous discrete event systems - Datalog to the rescue! In *ACM PODS*, 2005.

2. S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, N. Preda, Lazy Query Evaluation for Active XML, In Proc. of ACM SIGMOD 2004.
3. S. Abiteboul, O. Benjelloun, T. Milo, Positive Active XML, In Proc. of ACM PODS, 2004.
4. S. Abiteboul, P. Buneman, D. Suciu, Data on the Web, Morgan Kaufmann, 2000.
5. S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, T. Milo, Active XML Documents with Distribution and Replication, In Proc. of ACM SIGMOD, 2003.
6. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, Reading-Massachusetts, 1995.
7. Abiteboul, S., I. Manolescu, E. Taropa. A framework for distributed XML data management. In *Proc. EDBT*. 2006.
8. Serge Abiteboul, Neoklis Polyzotis, The Data Ring: Community Content Sharing In Proceedings of CIDR, 2007.
9. The ActiveXML project, INRIA, http://activexml.net.
10. The Object Database Standard: ODMG-93, editor R. G. G. Cattell, Morgan Kaufmann, San Mateo, California, 1994.
11. Sharma Chakravarthy, Jennifer Widom: Foreword: Special Issue on Active Database Systems. J. Intell. Inf. Syst. 7(2): 109-110. 1996.
12. 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/
13. The Edos Project, http://www.edos-project.org/
14. Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing Queries Across Diverse Data Sources. In *vldb97*, pages 276–285, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.
15. M. Harren, J. Hellerstein, R. Huebsch, B. Thau Loo, S. Shenker, and I. Stoica. Complex queries in dht-based peer-to-peer networks. In *Peer-to-Peer Systems Int. Workshop*, 2002.
16. T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, F. Dang Ngoc, Exchanging Intensional XML Data, In Proc. of ACM SIGMOD, 2003.
17. M.T. Ozsu, P. Valduriez, Principles of Distributed Database Systems, Prentice-Hall, 1999.
18. A. Muscholl, T. Schwentick, L. Segoufin, Active Context-Free Games, Symposium on Theoretical Aspects of Computer Science, 2004.
19. J.D. Ullman, Principles of Database and Knowledge Base Systems, Volume I, II, Computer Science Press, 1988.
20. The SOAP Specification, version 1.2, http://www.w3.org/TR/soap12/
21. Universal Description, Discovery and Integration of Web Services (UDDI), http://www.uddi.org/
22. The Extensible Markup Language (XML), http://www.w3.org/XML/
23. XML Typing Language (XML Schema), http://www.w3.org/XML/Schema
24. An XML Query Language, http://www.w3.org/TR/xquery/
25. The W3C Web Services Activity, http://www.w3.org/2002/ws/
26. The Web Services Description Language (WSDL), http://www.w3.org/TR/wsdl/