STUDIA INFORMATICA 2009
Volume 30 Number 2A (83)

Marcin MARCZEWSKI
Uniwersytet im. Adama Mickiewicza, Wydział Matematyki i Informatyki

THE ANALYSIS OF MATERIALIZATION PROCESS EFFECTIVENESS STRATEGIES IN AXML-BASED DISTRIBUTED SYSTEMS

Summary. Active XML (AXML) is a powerful framework for data integration in a distributed, peer-to-peer environment. In AXML documents information can be presented extensionally or intentionally by embedding calls to local or remote Web services. Invoking such a call to a Web service and replacing its call definition by the data returned from that service is called materialization.

In this paper various strategies for materialization processes and their impact on effectiveness of AXML systems are analyzed. Considerations are justified by examples.

Keywords: XML, materialization, Web Services, query processing

PRZEGLĄD TECHNIK WSPOMAGAJĄCYCH MATERIALIZACJĘ DANYCH W ROZPROSZONYCH SYSTEMACH AXML

Streszczenie. Aktywny XML (AXML) to framework wykorzystywany w procesie integracji danych w rozproszonym środowisku peer-to-peer. W dokumentach AXML dane mogą być prezentowane bezpośrednio lub przez wbudowane odwołania do usług sieciowych Web service. Realizacja tych odwołań przez wywołanie usług sieciowych i zastępowanie ich definicji uzyskanym rezultatem to materializacja.

W pracy tej omawiane są różne strategie realizacji procesu materializacji danych oraz ich wpływ na efektywność całego systemu. Rozważania te poparte są przykładami.

Słowa kluczowe: XML, materializacja, usługi sieciowe, wykonywanie zapytań

1. Introduction

Active XML (AXML)[1, 2, 4 and 5] is a powerful framework for data integration in a distributed, peer-to-peer environment.

In AXML information can be retrieved from distributed, heterogeneous sources which are able to provide knowledge using Web services technology. Those sources (known as AXML peers) are repositories of Active XML and XML[11] documents. They can act both as servers (by providing services which are usually defined as queries over AXML or XML documents stored locally) and as a client (by invoking calls to defined Web services [6 and 7]).

In AXML documents information can be presented in two ways: a) extensionally and b) intentionally by embedding calls to local or remote Web services. When invoking such a call to a Web service its call definition in an original AXML document is replaced by the data returned from that service.

Retrieved data sets could be provided in a text form, fragment of XML document or new piece of AXML code including deeper calls to other Web sources. In complex AXML-based systems multiple nesting calls to Web services or improper order of calls invocation can cause data retrieval to be inefficient in particular situations.

In this paper various strategies for materialization processes in a distributed, AXML-based environment are analyzed. Different approaches showing how they can affect effectiveness of AXML data integration processes are investigated. Finally, the use of various modifications of AXML peer architecture which could have an impact on choosing the most optimal data retrieval strategy is discussed.

The paper is organized as follows. Section 0 describes the main idea of Active XML together with the description how AXML documents are written and stored. It also provides short information about AXML peer architecture. In Section 3 the main focus is on materialization processes as well as on defining various strategies for AXML materialization. Section 4 describes and analyzes existing solutions which could have an impact on AXML data integration performance. Conclusions are defined in Section 5.Motywacja badań

2. Active XML

2.1. Active XML documents

An Active XML document is a usual, well-formatted XML file which, except local data, can include embedded Web service calls. Web services mentioned in AXML documents when executed return as an output data which is also presented in XML form. What is more, this

returned data can also include calls to locally or remotely shared Web services (be an AXML code itself).

Web service described in a service call nested between special tags in an AXML document is uniquely identified by two parts: name of the server which shares this Web service and the name of method of invoked Web service. In addition, this embedded Web service call can also define customized parameters [3] which can affect the behavior of the document before or after invoking those Web services. In the AXML implementation proposed in [12] such invocation is included between special tags: `<sc>` and `</sc>` (sc comes from service call):

```
<hotels>
  <county name="Poznan">
       <sc>poznan.com/HotelsPoznan()</sc>
  </county>
 </hotels>
```

The above presented piece of sample Active XML document includes an embedded service call to the Web service `hotelsPoznan()` which is shared by machine poznan.com.

2.2. Architecture of AXML peer

Active XML-based systems are based on a decentralized, pure peer-to-peer communication model. Every node in this environment is called an AXML peer. AXML systems are dynamic and volatile, i.e. peers can join and leave intermittently.

Every AXML peer can act as a client requesting data from remote peers (by calling remote Web services) and as a server (can be a source and can share data via its local Web services). All the information exchanged between various AXML peers is transferred using SOAP[14] (with characteristics adopted depending on the technologies used).

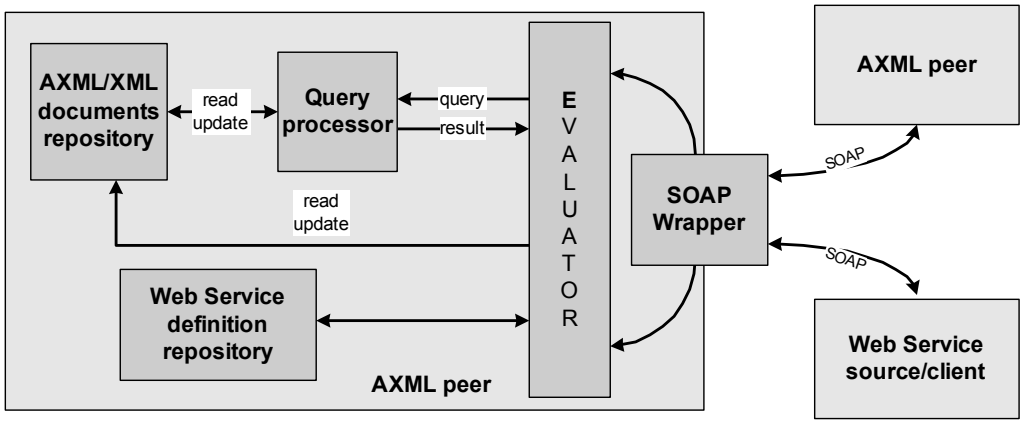

Fig. 0. AXML peer architecture
Rys. 1. Architektura węzła AXML

Main components of AXML peer architecture are (Figure 1) [3]: AXML/XML documents repository, Web services definition repository (called also Service descriptions repository), Query processor, Evaluator and SOAP Wrapper. Figure 1 shows relations between AXML

peer components. All those components perform specific roles (described in [3 and 19]) in AXML data integration processes.

3. Data materialization in AXML

Calling and executing Web services included in an AXML document is called **materialization**. Materialization means that a specific Web service is invoked with defined set of parameters, results of this are read and piece of original AXML document which includes tags describing service call is replaced by received results or those results are appended to the document. Such data results can also include calls to other Web services. What is more, in one AXML document multiple service calls can be defined. Finally, there can exist calls which consume results of one executed Web service as input parameters for another one (nested calls).

Materialization process, i.e. data retrieval by invoking Web services included in AXML documents is performed by participating AXML peers and transferred among them using means such as SOAP [14] and WSDL [13]. It also means that all the documents must be defined and processed according to the characteristics stated by those protocols [13 and 14].

Materialization is in the center of each AXML system, because it defines how data integration processes and information exchange can occur in the whole system. It also states what data (and when) is delivered to participating AXML peers.

Work in [19] presents how materialization tasks of one document, even in a small environment consisting of four AXML peers, can differ. Depending on a chosen strategy the result of materializing an AXML document is different.

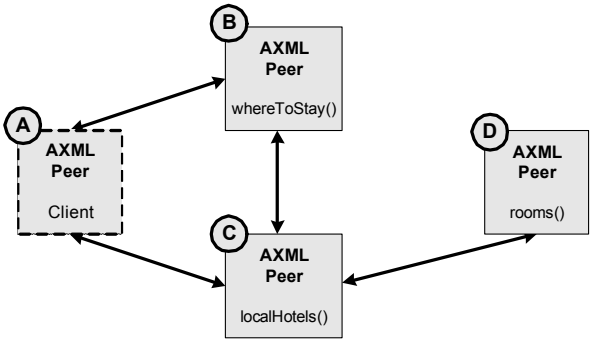

Fig. 1. AXML-based, tourist information system
Rys. 2. System informacji turystycznej oparty na AXML

To get a requested set of data an AXML peer (A), Figure 2, must materialize local AXML document which includes an embedded call to a remote Web service `whereToStay()` shared by other AXML peer (B). The service call is defined as following:

```
<dolnyslask>
      <sc>dolnyslask.com/whereToStay()</sc>
</dolnyslask>
```

Depending on the materialization strategy chosen by AXML peer (B) various results can be returned to AXML peer (A):

Example 1. AXML peer (B) to answer request from AXML peer (A) must call a Web service `localHotels()` shared by AXML peer (C). AXML peer (C) returns to AXML peer (B) the whole set of data records which are finally forwarded to AXML peer (A). Finally, AXML peer (A) stored the following document (after materialization):

```
<dolnyslask>
  <county name="legnicki">
       <hotel name="Hotel Wroclawski">
       <room person="2">7</room>
       </hotel>
       <hotel name="Hotel Centrum">
       <room person="2">20</room>
       </hotel>
  </county>
</dolnyslask>
```

Example 2. AXML peer (C) returns to AXML peer (B) a part of AXML code which includes a service call to a Web service shared by AXML peer (D). AXML peer forwards that piece of AXML code to AXML peer (A). After such a materialization process AXML peer (A) stores the following document which is only partially materialized:

```
<dolnyslask>
  <county name="legnicki">
      <hotel name="Hotel Wroclawski">
      <room person="2">7</room>
      </hotel>
      <hotel name="Hotel Centrum">
      <sc>hotelCentrum.pl/rooms(type,person)</sc>
      </hotel>
  </county>
</dolnyslask>
```

In this case AXML peer (A) must invoke another materialization process in order to get data sets delivered by AXML peer (D).

The above mentioned examples show that different materialization strategies can result in providing extremely different results and can require additional operations in order to get desired sets of data. That also means consuming additional system resources.

3.1. Materialization challenges

When analyzing materialization processes in a dynamic, AXML-based, distributed environment two aspects of data integration must be considered: processes which are affected internally i.e. those which could be optimized inside one AXML peer, and those which are controlled externally and are affected by more than one peer. The former includes not only architecture-related aspects of AXML peer, but also query optimization strategies as well as solutions used for storing and retrieving the data (mainly query processing [20]). The latter is affected by the

environment, i.e. how peers cooperate, how data sets are distributed and how they can be exchanged (data replication, distribution as well as data caching and re-use possibilities).

All the above define a set of areas and potential challenges which must be considered before or during materialization. These are:

- invocation order – i.e. the order of service calls execution in which Web services should be called and executed in order to provide the desired result;
- invocation dependencies – i.e. which service calls require another service calls results to be consumed as input parameters;
- dynamic community and its impact on the system – in P2P environment peers can join and leave dynamically. Moreover, more than one node can share the same or similar set of resources;
- search complexity – finding the most optimal way in which the request should be evaluated (which nodes and when should take part in the data integration process) can be very difficult and resource-consuming, especially in wide, dynamic environment with high-level of data distribution. Research made in [17 and 18] shows that finding the optimal execution plan in a volatile distributed environment is very complex and can take hundreds of years to just find such a plan;
- invocation deadlock – invoked Web services can return AXML code as a results. It means that other Web services shared by other peers must be invoked in order to get the pure, desired data. In a specific case, the root peer (that one initializing materialization of an AXML document) can get as resulting AXML code referring to its own Web service;
- lack of data and low data quality – in AXML-based systems peers can join and leave dynamically, sharing their data only in some periods of time. What is more, many peers can share the same or similar data sets and those with the highest quality should be chosen.

All the aspects presented above should be addressed by any or a combination of AXML materialization optimization strategies in order to provide efficient, high-quality way of data integration of AXML data.

4. Analysis of AXML optimization strategies

Since materialization in Active XML environment is a dynamic process and results of invoked service calls can be used as input for further invocations, the number of various alternatives which Web services should be called can be very high. A few researches were focused on finding the most optimal way of materializing AXML.

In this section various approaches which influence AXML materialization process are described and analyzed. Those include defining simple workloads for materialization, querying AXML data stored locally and externally, dynamic cost-based optimization algorithms for distributed data management, continuous AXML services, and caching mechanism.

4.1. Defining workflows

Finding an optimal execution plan for AXML materialization with multiple service calls defined in a wide AXML environment can be complex. A good way to limit the number of resources consumed to avoid invocation deadlocks is to define the order in which those service calls should be invoked and executed – i.e. defining workflows. Defining workflows can also help to define dependencies between Web services called during materialization process as well as to break into smaller pieces the whole process in order to use parallelism or delegation of tasks to external resources.

Workflows can be defined by using additional parameters in `<sc>` tags in AXML documents. Those parameters (`followedBy="serviceName"`) – define which Web service (local or remote) must be invoked and executed just after finishing materialization of the current one. An example of such an AXML document could be defined as:

```
<hotels>
  <county name="Poznan">
       <sc id="1" service="poznan.com/HotelsPoznan()" followedBy="2" />
       <sc id="2" service="stayhere.pl/stayPoznan()" followedBy="3"  />
       <sc id="3" service="stayhere.pl/stayHere()" />
  </county>
 </hotels>
```

In the example presented above, immediately after invoking and executing Web service `HotelsPoznan()` shared by node `poznan.com` Web service `stayPoznan()` is invoked too. Similarly after finishing working with `stayPoznan()` that peer will call `stayHere()`.

Presented approach is processed locally without any knowledge of materialization at other AXML peers. It also can negatively influence the overall processes, especially when some sub-services are unavailable and the materialization process must be stopped (invocation dependencies) or ignored (lack of data, low quality). On the other hand it can limit the risk of invocation deadlocks by proper service calls ordering.

4.2. Querying documents

Active XML documents are stored by each AXML peer in a document repository and can be accessed only by locally defined Web services. Those services can retrieve data using various XML query languages, for instance, using XPath [15] or XQuery [16].

When chosen Web service is invoked during materialization process an AXML Query processor is used. It is a component responsible for query execution and efficient data retrieval from files stored in the local documents repository. As a result of query execution some data collection can be returned. Moreover, some modification (updates) in source AXML files can be made as well.

Query processing and access plan optimization [20] is a crucial process, because its efficiency can directly affect the overall performance of the whole AXML-based system.

Another approach is a possibility to build queries over XML data using languages commonly used for relational data - such as popular SQL. Such approach could speed up processing of data and could provide an option to build joins on both XML and relational data which can make AXML more powerful. That can be done using repositories which can store XML data in a native format – such as pureXML technology [21] in IBM DB2 9 database. Implementation using such a repository is currently under development. That can limit the search complexity and provide requested data more effectively.

4.3. Dynamic cost-based optimization with XCraft

In [8] a dynamic cost-based optimization strategy for optimizing AXML materialization processes is proposed. Presented solution introduces an additional component in the Active XML peer architecture which takes the responsibility of the query optimizer (called as XCraft [8, 17 and 18]) and collects detailed information about up-to-date status of known peers.

XCraft provides a way of dynamic splitting of materialization process into small pieces which can be distributed between various peers cooperating at the time of query invocation. This enables the system to deliver at least partial results in more efficient way in a peer-to-peer environment where peers can leave and join dynamically.

In XCraft some important assumptions are made – this approach considers materialization process from workflow point of view. It means that Web service calls' invocations can relate each other and the order in which they are used must be properly defined. This is very important especially when materialization results of one or more service calls are used as input parameters in other invocations. In such cases sequencing those invocations as well as synchronizing them must be performed in order to provide relevant results.

XCraft adopts a cost-based algorithm to be used to generate an optimal query plan. It takes into consideration many aspects: materialization tasks delegation to other peers than the invoking one, parallel execution of service calls by many peers as well as ordering those tasks.

Described method of dynamic cost-based optimization addresses many challenges such as building calls invocation order, dependencies and limiting the search complexity and risk of

invocation deadlocks. Moreover, tasks delegation is a solution for providing missing data when a chosen peer is temporarily unavailable.

4.4. Continuous services

In many scenarios the data returned by invoking Web services can change dynamically on regular basis. In such cases root AXML peer (that one starting requesting the data) should repeat materialization process in shorter periods in order to maintain up-to-date version of data. That is the area where continuous services come.

The idea of continuous services [9] is to define Web service calls as a kind of subscription. In this approach the source AXML peer (that one delivering information) regularly sends information to the root AXML peer. That information includes updates with changes which occurred in the meantime. That also extremely limits the number of data sets needed to be send over the network among AXML peers, because source AXML peer does not need to send the whole information each time a Web service providing the data is called.

Continuous services work as kind of subscription. AXML peer (A) needed to be feed by data from Web service shared by AXML peer (B) asks it for subscription. AXML peer (B) in chosen, regular periods or when changes to data sets are introduced sends to AXML peer (A) a data set including up-to-date information. It also addresses issues such as invocation deadlocks or low quality of delivered data.

4.5. Caching mechanism

In [10] a simple caching mechanism for Active XML-based applications was presented. This materialization strategy addresses scenarios where data sets transferred through the network can overlap in many parties and this can lead to bottlenecks as well as propagation of unnecessary data sets.

The solution from [10] addresses the following requirements for AXML, where the number of querying hosts is high and the same data sets are sent among the peers:

- passing the same data sets between many nodes can lead to lower performance of the whole system (bottlenecks, inefficiency),
- data sources delivering information must work all the time,
- getting information with other parameters requires repeating (calling) Web services.

Based on the network and data parameters an AXML peer responsible for data caching is elected (kind of master peer). The analysis can be based on the following parameters: frequency of received and passed queries, network parameters, host localization, security issues etc. Elected AXML peer informs about enabling its caching facility all its neighbors. When any of these neighbors is hit by any other node requesting information then such

request with all parameters values is passed to caching machine (calling special Web service `getCached()`) which retrieves data and stores it locally.

The presented caching mechanism can limit the number of transactions performed among peers in order to retrieve requested data and have positive impact on the network performance. Moreover, it provides a possibility of data retrieval even in a case when the original data source is not working at the time.

Another interesting advantage of presented AXML caching solution is that caching machine may return approximate data sets according to the parameter range specified in the request – when the actual data is missing or data provider is not active. A disadvantage of caching approach is introducing a kind of centralization in the AXML system, which can also have a negative impact on the overall performance and serviceability.

4.6. Area for future work on materialization process effectiveness

The techniques for materialization o AXML data in a distributed environment provide a lot of flexibility and can work efficiently in small and large AXML-based systems, but there are still many research areas to investigate. AXML systems are large, peer-to-peer environments where peers join and leave intermittently. In such systems, materializing a lot of documents can result in invoking the same Web services at the same time or in small intervals. An interesting research area for such environment is to optimize not only single queries, but also sets of queries with shared sub-queries. Such an idea for Advanced Database Systems was presented in [22].

Another challenge for future work is to provide efficient mechanism for providing guaranteed high-quality results for AXML materialization. Contemporary techniques for distributed AXML-based systems do not focus on that area of problem.

5. Conclusion

AXML is a framework for distributed data management. In AXML systems many peers can exchange data in a dynamic, distributed manner. Since peer-to-peer systems are commonly used nowadays, together with popular technologies such as Web services, AXML can provide an efficient way of data distribution and integration. Moreover, AXML can integrate various environments which can deliver data sets using Web services which can access data stored in XML, AXML files as well as in relational databases.

In the paper focus on materialization process of AXML data is set. This work discusses various strategies which can have an influence on it how data integration processes in AXML-based environments occur. This paper shows materialization strategies and techniques such as defining workflows, dynamic cost-based optimization with XCraft query optimizer, continuous services or caching mechanism which can definitely improve the way Web services defined in AXML documents are materialized. Presented strategies can also be combined depending on the aspects which should be addressed.

BIBLIOGRAPHY

1. Abiteboul S., Manolescu I., Taropa E.: A Framework for Distributed XML Data Managements, EDBT, 2006, p. 1049÷1058.
2. Abiteboul S., Benjelloun O., Milo T.: The Active XML project: an overview, 2005.
3. The Active XML Team, Active XML User's Guide, http://www.activexml.net.
4. Abiteboul S., Amann B., Baumgarten J., Benjelloun O., Dang Ngoc F., Milo T.: Schema-driven Customization of Web Services, VLDB, 2003, p. 1093÷1096.
5. Abiteboul S., Benjelloun O., Manolescu I., Milo T., Weber R.: Active XML: Peer-to-Peer Data and Web Services Integration, VLDB, 2002, p. 1087÷1090.
6. Milo T., Abiteboul S., Amann B., Benjelloun O., Dang Ngoc F.: Exchanging intensional XML data, ACM Trans. Database Syst. 30 (1), 2005, p. 1÷40.
7. Abiteboul S., Benjelloun O., Milo T.: Positive Active XML, PODS, 2004, p. 35÷45.
8. Ruberg G., Mattoso M.: XCraft: Boosting the Performance of Active XML Materialization, EDBT, 2008.
9. Taropa E.: Continuous services in AXML, Ecole Polytechnique Tech. Report.
10. Marczewski M., Pankowski T.: Data caching in data integration systems based on AXML technology, DEXA - GRep workshop, 2007.
11. W3C, Extensible Markup Language (XML), http://www.w3.org/XML.
12. Active XML home page, http://www.activexml.net.
13. W3C, Web Services Definition Language (WSDL), http://www.w3.org/TR/wsdl.
14. W3C, Simple Object Access Protocol (SOAP), http://www.w3.org/TR/SOAP.
15. W3C, XML Path Language (XPath), http://www.w3.org/TR/xpath.
16. W3C, An XML Query Language (XQuery), http://www.w3.org/TR/xquery.
17. Ruberg N. Ruberg G., Manolescu I.: Towards cost-based optimization for data-intensive Web service computations, SBBD, 2004, p. 283÷297.
18. Ruberg G., Mattoso M.: XCraft: A dynamic optimizer for the materialization of active XML documents., COPPE/UFRJ Tech. Report, 2007.

19. Marczewski M., Pankowski T.: Integracja danych w środowisku Web z wykorzystaniem technologii AXML, BDAS, 2007.
20. Kossmann D.: The State of the Art in Distributed Query Processing, 2000.
21. Chen W., Sammartino A., Goutev D., Hendricks F., Komi I., Wei M., Ahuta R., Nicola M.: IBM DB2 9 Pure XML Guide, IBM Redbook, 2007.
22. Królikowski Z.: Optymalizacja wykonywania zapytań w zaawansowanych systemach baz danych, Politechnika Poznańska, Rozprawy, nr 326, 1998.

Recenzent: Dr inż. Michał Kozielski

Wpłynęło do Redakcji 1 lutego 2009 r.

Omówienie

Aktywny XML (AXML) to framework wykorzystywany w procesie integracji danych w rozproszonym środowisku peer-to-peer. W AXML informacje mogą być odczytywane z różnorodnych, rozproszonych źródeł, które są w stanie dostarczać wiedzę przy użyciu technologii usług sieciowych Web service. Źródła te (węzły AXML) są repozytoriami zarówno dokumentów Aktywnego XML, jak i XML. Mogą pełnić rolę serwera (przez dostarczanie usług, które zwykle zdefiniowane są jako zapytania do dokumentów AXML i XML przechowywanych lokalnie) oraz klienta (przez wywoływanie odwołań do lokalnych usług Web services). W dokumentach AXML dane mogą być prezentowane bezpośrednio lub przez wbudowane odwołania do usług sieciowych Web service. Realizacja tych odwołań przez wywołanie tych usług i zastępowanie ich definicji uzyskanym rezultatem to proces materializacji. Dane mogą być dostarczane zarówno w postaci tekstu, fragmentu kodu XML czy dokumentu AXML zawierającego dalsze odwołania do innych usług sieciowych Web services.

W złożonych systemach opartych na AXML wielokrotne zagnieżdżanie odwołań do usług sieciowych Web services lub niepoprawny porządek ich wywołania w pewnych sytuacjach może powodować nieefektywny odczyt danych (materializację).

W pracy tej omawiane są różne strategie realizacji procesu materializacji danych oraz ich wpływ na efektywność całego systemu. Dodatkowo, analizowane są różne rozwiązania umożliwiające efektywne zarządzanie danymi AXML, nawet w przypadku, gdy oryginalne źródło danych jest niedostępne w danym czasie. Rozważania poparte są przykładami.

Address

Marcin MARCZEWSKI: Uniwersytet im. Adama Mickiewicza, Wydział Matematyki i Informatyki, ul. Umultowska 87, 61-614 Poznań, Polska, Marcin.Marczewski@amu.edu.pl.