Review

Active XML (AXML) research: Survey on the representation, system architecture, data exchange mechanism and query evaluation

Binh Viet Phan, Eric Pardede*

Department of Computer Science and Computer Engineering, La Trobe University, Melbourne, Australia

Abstract

Active XML (AXML) is an extension of XML to exploit the powerful computation ability of peer-to-peer network and Web services technologies. AXML is considered a distributed XML DBMS which extends the capability of XML by embedding intensional XML data inside XML documents. The management of intensional XML and XML data together in XML documents raises issues such as representation for intensional XML data, AXML–XML data exchange and AXML data query processing. This paper will study these issues, comparing as well as discussing the current solutions to AXML systems.

Contents

1. Introduction ......................................................................................................348
2. AXML as intensional XML data .......................................................................................349
3. AXML representation ...............................................................................................350
   3.1. Gemo's AXML representation ..................................................................................350
   3.2. ARAXA AXML representation ...................................................................................352
   3.3. LTU's AXML representation ....................................................................................352
4. AXML systems architectures . . .......................................................................................354
   4.1. Gemo's AXML system architecture . . ............................................................................354
   4.2. ARAXA's AXML system architecture . ............................................................................354
   4.3. LTUAXML system architecture ..................................................................................356
5. AXML data exchange ...............................................................................................356
   5.1. Gemo's proposal for exchanging intensional XML data ..............................................................357
   5.2. LTU's technique for intensional XML data exchange .................................................................359
6. AXML query processing .............................................................................................360
   6.1. Gemo's AXML Systems query evaluation ...........................................................................360
   6.2. ARAXA's AXML Systems query evaluation ........................................................................361
   6.3. LTU's AXML system query evaluation ...........................................................................362
7. Future research issues ..............................................................................................363
8. Conclusion .......................................................................................................364
References ..........................................................................................................364

1. Introduction

Traditional database systems are well-suited to manage and organize structured data. They are also used as one of the foundations for building and publishing web sites over centralized network architectures and the internet. The strength of
these database systems is derived from strict requirements of structured data as well as their own data formats. However, there is a considerably high cost implication for data exchange and integration between traditional database systems and between applications and web sites built on top of these database systems. Moreover, along with the developments of the internet as well as diversified data formats, there is a high demand for exchanging and storing unstructured data which is not well-suited to be stored inside traditional database systems. These problems are solved by the emergence of eXtensible Markup Language (XML).

XML (Bray et al., 2008) is a flexible and scalable text-based language. In the last decade, it has been very popular for data representation and exchange over the internet. XML is applied as a means to communicate between applications over networks, regardless of different platforms. XML can be used as a repository for unstructured data as well as for data exchanges. In the early development stage of the internet, client-server architecture was a typical model for web applications. Nevertheless, centralized network models do not support the various properties needed in current web applications as well as trends of current technology such as exchangeability, heterogeneity, data scalability, autonomous systems and Service-Oriented trends. Therefore, peer-to-peer architectures are proposed as an alternative solution. The development of peer-to-peer architecture is also supported and facilitated by the emergence of web services technologies that help to remove the antagonism of diversified systems and platforms.

Web services are technologies that have been rapidly developed. This technology has been creating and contributing to fundamental infrastructures for Web applications. Web services are expected to be efficient sources of data, which are dynamic, distributed and interoperable. Hence, Web services can be considered as data objects and can be manipulated like data in database systems. Moreover, along with the developments of P2P architecture because of its capability to share computing power between peers in the networks.

For end-users, AXML performs like normal centralized XML DBMSs. Additional computations such as invocation of Web services, materialization of AXML data and querying AXML data are executed by additional modules along with XML DBMSs, which is transparent to the end users.

We provide several scenarios to demonstrate how AXML systems can be used in our daily life.

**Scenario 1: Newspaper Web Content.** A homepage of a newspaper usually comprises: (1) explicit data such as the newspaper’s name, the current date and time etc.; and (2) implicit data such as the weather forecast and local events. It is easy to see that the information about weather and local events should not be given as explicit and static data because this information only makes sense at a specific time. This information is usually retrieved by Web service calls.

**Scenario 2: Collaborative Management for a Real Estate Agency.** It is common for a real estate agency to design a database that can manage, facilitate and provide collaborative and mobile workspace for their staff as well as information to customers. A staff member can access their related documents, create new information and insert these into documents from their portable devices via the internet.

The most suitable solution is using intensional data on P2P architecture because it reduces the load on the central server and each peer can have their own initiative and independency. As described in (Abiteboul et al., 2003; Goldman and Widom, 1997), the agency can divide the database into three separate documents; namely, properties, requests and status (see Table 1). These documents contain comprehensive information about the properties managed by the real estate company. This information will be updated on the central repository. Staff can also examine this information on the central database. All information will be exchanged and organized between peers and the central repository by intensional information via Web services.

**Scenario 3: Electronic Patient Record Management.** Electronic Patient Record (EPR) is a document under the control of a number of peers such as hospitals, patients, insurance companies, and the

---

### Table 1
AXML Documents and Web services for a Real Estate Agency.

<table>
<thead>
<tr>
<th>AXML Documents</th>
<th>Contain</th>
<th>Web services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties.xml</td>
<td>A list of properties</td>
<td>getProperties(assignedTo, type, location, price); getPicture(propertyID); addClient(propertyID, clientDescr,by)</td>
</tr>
<tr>
<td>Requests.xml</td>
<td>Requests</td>
<td>getRequests(requestDescr); putRequest(requestDescr, handleBy)</td>
</tr>
<tr>
<td>allStatus.xml</td>
<td>Status of properties</td>
<td>getStatus(propertyID); updateStatus(propertyID)</td>
</tr>
</tbody>
</table>
health department. These peers, which can act as the provider and the user of information, exist on remote mobile devices, computers and servers. Patients can add or remove their information. Doctor and nurses can access this information with some restrictions. Moreover, all data can be controlled and monitored by hospitals, insurance companies and the health department (Abiteboul et al., 2004a, 2004b, 2004c, 2004d; Kossmann, 2000). The data distributed in the system can be arbitrarily updated by peers, and thus, the system will gain benefit if it uses intensional data. Peers will contribute to the most up-to-date and correct patients’ information via Web services.

We identified several research issues that we have to address to fully employ the benefit of AXML data. The first issue is the representation of AXML data. The second issue is the architecture of the AXML system. The third issue is the AXML data exchange mechanism. Finally, the fourth issue is the AXML query processing. We will discuss each of these issues in the next four sections. In each section, we identify the existing work and follow up with our new initiative.

3. AXML representation

AXML was introduced by Milo et al. (2003) who applied the eXist native database system. Ferraz et al. (2010) identified the shortcomings of storing and processing AXML data in a native XML database and employed Object-Relational DBMSs as the repository instead. A later improvement of AXML representation was proposed by Phan and Pardede (2011). AXML documents represented by these three groups are valid XML documents and comply with XML W3C standards (Ray et al., 2008). This section discusses and compares the three AXML representations.

3.1. Gemo’s AXML representation

Intensional XML data is not real XML data but representative of XML data. Particular representations are needed to distinguish them from normal XML data in AXML documents. To represent intensional XML data, (Abiteboul et al., 2004a, 2004b, 2004c, 2004d; Milo et al., 2003) proposed special elements called service calls (sc), which contain information to invoke Web services to retrieve XML data. AXML documents are modeled as ordered trees with two types of nodes: (i) XML nodes and (ii) intensional XML nodes sc (Milo et al., 2003; Abiteboul et al., 2008; Team, 2005). We discuss the second type of node in detail as follows.

Intensional nodes are the heart of AXML documents. They are special elements to process intensional data. These elements are denoted by the (sc) tag with axml prefix. These elements contain children elements and attributes to define the Web service to invoke, the invocation methods and the parameters and methods for handling returned results (Milo et al., 2003; Team, 2005; Ruberg and Mattoso, 2008).

Web service calls stored inside intensional elements can be classified into three groups. The first is the Web service call to any Web service on the web. An example is the invocations to Web services provided by Amazon. The second is the Web service call to a P2P network. The third is the continuous Web service call that can deliver a stream of data including requests, parameters and returning results. All three types of embedded Web services calls can be provided by third parties, or can be declared by XML Query Languages inside AXML documents (Abiteboul et al., 2002, 2004a, 2004b, 2004c, 2004d; Milo et al., 2003).

The service calls element (sc) complies with standard regulations to declare XML elements. However, sc elements contain some special attributes regarding Web service information (Tatarinov et al., 2002): (i) ServiceURL is the endpoint URL of the service; (ii) serviceNameSpace is used for the body of the SOAP message; (iii) methodName is the name of the operation to invoke the service; (iv) signature is the URL of WSDL file for the service; and (v) useWSDL is a boolean value that stipulates whether or not to perform type validation.

There are an additional six properties that can specify and control service invocation (Tatarinov et al., 2002): (i) id is the unique value of the service call; (ii) name assigns the service call with a name; (iii) frequency stipulates the time interval for Web service invocation; (iv) callable is a boolean value to specify whether or not to allow another AXML peer to activate the service; (v) lastCalled stores the time of the last service activation; and (vi) followedBy indicates the next service call being activated after activating the current Web service call.

To manage returned results, AXML systems employ two attributes (Team, 2005): (i) mode to indicate how to store results; and (ii) doNesting is the history of the previous results (except text nodes) inserted.

The parameters of the Web service are sc’s child elements. If a Web service does not have any parameters, the child elements will be declared empty. Parameters can be explicit values or an XPath expression. If there are many parameters for a service call, these parameters must be specified in a certain order.

As mentioned before, specific representation for intensional data are used to distinguish between XML and intensional XML data so that AXML systems can recognize and process intensional data. When an intensional node is being requested, XML data received by materializing this node will replace or append intensional node as the siblings (Abiteboul et al., 2004a, 2004b, 2004c, 2004d; Milo et al., 2003; Ruberg and Mattoso, 2008). Figure 1 is an example of an AXML document for a local newspaper (Milo et al., 2003; Abiteboul et al., 2008).

The representation proposed by Gemo (Milo et al., 2003; Gemo, 2007) is the foundation for AXML systems. The representation of intensional data is used to distinguish between intensional XML and XML data so that AXML system can handle and process intensional XML during query evaluations and data exchange. Moreover, the representation will decide how intensional data are stored, processed and queried.

However, Gemo’s representation has several shortcomings. Each of the shortcomings will be illustrated by using a hotel management example, as shown in Fig. 2.

3.2. AXML improvements and limitations

a. Waste storage space. As it can be seen in Fig. 2, every hotel element can contain Web services such as getNearbyMuseum, so similar information on these Web services are repeated in each hotel element. It is easy to see that when the number of hotel elements is large, there is a considerable amount of repeated information on the Web services.

b. Repetition of service call and data nodes. After invocation of a Web service call, a parent node might be filled by repeated information. In Fig. 3, the first invocation of node A results in node 1, node 2...node n, and service B. The second invocation of node A will again return node 1, node 2...node n, and service B. These repeated Web services can return exactly the same results when they are invoked in the future. This will result in bad performance and it is also difficult to manage. To avoid repetition of service calls in parent nodes, current AXML needs to search and compare all Web services.

c. Service call management. It is difficult to disable Web service calls in AXML documents that are sometimes needed for various reasons, such as for security reasons. Currently, attribute callable with a “false” value is used to counteract Web service calls in a sc node. It affects only the Web service in that particular sc node. However, the service call can appear...
anywhere in AXML documents (Abiteboul et al., 2004a, 2004b, 2004c, 2004d, 2005; Milo et al., 2003; Ghitescu and Taroza, 2008), so inactivating all relevant nodes can take time and computing resources.

d. Lacking in a facility to materialize Web services in AXML documents. To work offline or to work with devices of limited abilities, AXML documents can be converted to pure XML data. Moreover, AXML results from query evaluations against AXML...
documents can be requested in the form of XML data. In these cases, all Web services in AXML documents or AXML fragments need to be materialized. Current AXML systems must always traverse and search all Web services in whole documents, determine relationships among them, then invoke these Web services. This can be time consuming, particularly in large AXML documents.

e. Ineffectiveness to find and specify dependencies among existing Web services. To indicate dependencies among services, additional algorithms such as the Data Guides (Bray et al., 2008; Abiteboul et al., 2003) can be applied. However, these algorithms can have an adverse effect on the performance of AXML systems. Furthermore, it is even more costly when new service calls are needed (Goldman and Widom, 1997). The current representation is not helpful to determine or to classify Web services in terms of what should be accepted as intensional XML data and what should not. This is because service calls appear everywhere in AXML documents so these have to be searched before invocation or classification.

f. Inability to deal with some characteristics of P2P networks. When materializing an intensional node sc, there is only one choice to invoke a Web service, which is stored in the sc. This is problematic in P2P architectures due to the possibilities of unavailability of peers and replication of data. If we only store one URL and method name that refer exactly to a concrete Web service as the current representation, many queries related to that Web service will not be evaluated when the peer providing the Web service is not available. In P2P networks, every peer can join or leave the network at any time, so dealing with the availability of service calls is an essential issue for AXML systems.

g. Problem choosing the most suitable Web services to invoke. In P2P network and distributed database systems, the data and Web services provided should be repeated at some peers, to assist in ensuring service providers and data are always available. This means that the same Web services or different Web services providing the same functions should be stored in different peers. Moreover, when we need to invoke a Web service, we have only one service choice to invoke, which is the Web service indicated in sc regardless whether the peer is busy, the call is costly, the bandwidth is low or even if the service is unavailable.

h. Not utilizing XML schemas. By employing XML schemas with suitable AXML representation, we can specify the position of Web service calls easily, reducing the cost for searching for Web services and avoiding the need to apply complex algorithms to indicate relationships as well as the dependence of Web services. However, XML schemas have not been applied in current AXML representation.

To overcome the limitations when storing multiple documents, ARAXA employs Document (id, doc name).

In ARAXA, intensional elements of an AXML document are stored in two separate relations, Service call (id, path id, dewey, doc id, serviceURL, methodName, serviceName, useWSDLDefinition, signature, callable, frequency, lastCalled, followed, mode, doNesting) and Parameter (id, service id, path id, type, name). These relations contain information regarding the location of intensional elements in AXML documents. The Service call relation stores Web service call catalogs, and the Parameter relation stores parameters for Web service invocations.

By employing ORDBMS, ARAXA can exploit the mature technologies of OR DBMSs during query evaluation and Web services invocation. There are two improvements in ARAXA for attribute reconstruction and mapping multiple documents (Ferraz et al., 2010). Finally, improvement is also made by reducing the repetitions of Web service calls because all Web service calls are stored in the Service call relation.

The weakness of this proposal is the high cost for the mapping and reconstruction of AXML documents. Moreover, original AXML documents before being mapped to ORDB and the AXML document after reconstruction still employ the representation proposed by (Abiteboul et al., 2004a, 2004b, 2004c, 2004d; Milo et al., 2003). Therefore, AXML in ARAXA inherits all the advantages and disadvantages from the previous representations.

3.3. LTU's AXML representation

Phan et al. (2011) proposed a new AXML representation to solve the shortcomings of the current representations. While this proposal maintains the attributes and elements of sc in (Milo et al., 2003), it divides them into different locations. In addition, new attributes are added to link these locations and to optimize data exchanges and query evaluation.

The first part of an AXML document contains a catalog of Ids of intensional nodes and Web service calls. The catalog includes essential information to invoke the Web services. Web services and information associated with them are labeled with ws tags. All ws elements are stored under an element called webServiceInfo. All ws elements have reference attributes to the ids of the sc elements, which are located in the second part of the AXML document.

This second part of the AXML document contains XML data and intensional elements sc. This new structure and organization of intensional data is applied to reduce the cost of searching during invocation and repetition of intentional XML data. Additional subQuery elements are added in sc (Phan et al., in press) which are queries imposed on sc. These elements are automatically generated whenever a query is executed on AXML documents. These elements are only applied to the sc elements where they are located. They cannot be transferred to other peers when the sc is being requested from other peers. The maintenance of subQuery in sc efficiently supports the exchange and the materialization of intensional XML data.

The elements and attributes of webServiceInfo, ws and sc are listed in the following Tables 2 and 3.

This new proposal offers several advantages as follows.

a. Avoids employing AXML engines to process XML documents. In the real world, AXML systems do not only work with AXML documents but also with pure XML documents. Therefore, when evaluating a query against XML documents, we can use XML engines without additional expenditure from the algorithm attached inside AXML. To indicate whether a document
Table 2
Elements and Attributes in webServiceInfor.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Attribute and Child Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>webServiceInfor</td>
<td>This is a catalog of Web services and intensional nodes. It is employed to manage information about Web services ws and intensional nodes sc</td>
<td>elementName</td>
</tr>
<tr>
<td>elementName</td>
<td>Representative for elements in data part. It is applied to group and manage: (1) all equivalent Web services, which provide XML data for elements specified in attribute name and (2) references to intensional nodes sc related to these Web services</td>
<td>Name, ws, node</td>
</tr>
<tr>
<td>name</td>
<td>This attribute specifies the name of elements which will be provided as XML data when their relevant Web services are invoked</td>
<td></td>
</tr>
<tr>
<td>node</td>
<td>Contains ID of sc node, and node to materialize the node</td>
<td></td>
</tr>
<tr>
<td>scID</td>
<td>It is id of sc node, This Id will be employed to refer to corresponding sc node in data part</td>
<td></td>
</tr>
<tr>
<td>wsID</td>
<td>It is used for referring to corresponding Web services</td>
<td></td>
</tr>
<tr>
<td>frequency</td>
<td>Indicate frequency to materialize the sc node</td>
<td></td>
</tr>
<tr>
<td>ws</td>
<td>Contain information regarding Web services</td>
<td></td>
</tr>
<tr>
<td>namespace</td>
<td>be used in SOAP message to invoke Web service (Abiteboul et al., 2008; Vidal et al., 2008)</td>
<td></td>
</tr>
<tr>
<td>serviceURL</td>
<td>be used in SOAP message to invoke Web service (Abiteboul et al., 2008; Vidal et al., 2008)</td>
<td></td>
</tr>
<tr>
<td>method Name</td>
<td>to invoke Web service (Abiteboul et al., 2008; Vidal et al., 2008)</td>
<td></td>
</tr>
<tr>
<td>signature</td>
<td>to validate Web service (Abiteboul et al., 2008; Vidal et al., 2008)</td>
<td></td>
</tr>
<tr>
<td>useWSDLDefinition</td>
<td>to validate Web service (Abiteboul et al., 2008; Vidal et al., 2008)</td>
<td></td>
</tr>
</tbody>
</table>

Table 3
Elements and Attributes in sc.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Attribute and child elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc</td>
<td>These elements are placed in data part, at positions, where their corresponding XML data will appear. sc contains information of intensional data</td>
<td></td>
</tr>
<tr>
<td>scID</td>
<td>Id of sc node so that this sc node can be linked to other information in webServiceInfor part</td>
<td></td>
</tr>
<tr>
<td>parameters</td>
<td>be applied to organize parameters for Web services invocations (Milo et al., 2003; Abiteboul et al., 2008; Vidal et al., 2008)</td>
<td></td>
</tr>
<tr>
<td>para</td>
<td>Parameter for Web services (Abiteboul et al., 2008; Vidal et al., 2008)</td>
<td></td>
</tr>
<tr>
<td>subQuery</td>
<td>subQuery enclosed to this sc nodes. This subQuery is employed to filter results during Web service invocation</td>
<td></td>
</tr>
</tbody>
</table>

is an AXML or XML document, the existence of webServiceInfor element is a quick check.

b. **Saves time for Web services invocation.** This representation assists to indicate, determine and collect intensional elements quickly by looking for sc under webServiceInfor instead of traversing whole documents. For example, consider an XML query in an AXML document shown in Fig. 2 “List all hotels in Sydney with their ratings”. Using the existing representation, whole documents have to be traversed through to find the “getRatingHotel” Web service because it can appear in literally any location of the AXML document. By using the new AXML representation, the system only needs to traverse the webServiceInfor element to find and materialize the intensional nodes and thus, the search scope and time can be reduced.

c. **Saves time and effort to determine the dependency among Web services.** With the new AXML representation, the dependency of Web services can be identified without the use of additional algorithms such as the Data Guides used in the previous system (Goldman and Widom, 1997; Abiteboul et al., 2004a, 2004b, 2004c, 2004d). The element webServiceInfor is organized as a hierarchical tree so dependencies among Web services will be indicated by their positions in webServiceInfor. For example (see Fig. 2), Web services getRatingsHotel and getHotel are Web services. With the new representation, these two Web services must be located as children of the hotel node under webServiceInfor. With the same reasoning, it is easy to see that getNearbyRestaurants and getNearbyMuseums are independent Web services. Moreover, if we use some simple labeling algorithm, dependencies of Web services can be indicated even more easily.

d. **Flexibility to choose suitable Web services for invocation.** All Web service calls are organized into groups and stored under the same element names. This helps the system to select the needed Web services at peers which are not busy, at the nearest distance or which incur the lowest bandwidth cost. For example, the Sydney tourism board stores data and services provided by different AXML peers (AXMLPeer1, AXMLPeer2, AXMLPeer3). We assume AXMLPeer3 is disconnected, AXMLPeer2 is too busy, and AXMLPeer1 is free. If there is a request from Melbourne for information on hotels in Sydney, using the new representations, equivalent Web services from these peers are stored under the same elements so Web services in AXMLPeer1 will be chosen, since this is the one that is free. In the current AXML representation, there is only one URL endpoint to AXMLPeer1, so every request to this peer may not always be processed.

e. **Avoids repetitions of Web services invocation.** In current AXML representation, the same Web services may appear at many sc nodes (Abiteboul et al., 2004a, 2004b, 2004c, 2004d; Milo et al., 2003). This repetition causes storage waste and management problems in searching, materializing, or disabling a Web service. For example, assume we notice that getRatingHotel is suspected of dangerous behavior, so this Web service must be disabled. With our representation, we will find getRatingHotel in webServiceInfor then update this getRatingHotel once. Using the current AXML representation, we must search this Web
service in the whole document and update it many times, whenever the Web service is found.

f. Avoids unnecessary updates. A previous proposal found in (Abiteboul et al., 2004a, 2004b, 2004c, 2004d; Milo et al., 2003; Ferraz et al., 2010) is costly to control updates on AXML documents. Each sc that can be located throughout the AXML document has storeMode attributes which will be applied immediately when Web services in that sc are executed. In the new representation, storeMode is gathered at suitable places under webServiceInfor. Therefore, it is easy to control updates on AXML documents during the materialization processes.

g. Faster detection of data exchange format. WebServiceInfor in this proposal acts like a schema for AXML documents. Hence, whenever intensional nodes are materialized, the structure of these nodes will be quickly provided by webServiceInfor fragments without the need to traverse the whole document to find the desired data exchange structures.

The three AXML representations show different ways to depict intensional data: Gemo’s representation stores information for intensional data inside sc elements; ARAXA still employs the same representation, with additional mapping of AXML documents into several relations in ORDB; and the LTU representation locate the Web service calls and the data separately. Even though this last proposal increases the complexity of the structure of the AXML document, (Phan et al., 2011) claim that the new representation can enhance the performance of AXML systems. Table 4 below briefly compares Gemo (and ARAXA) and LTU’s representations.

With respects to AXML representations, Gemo and ARAXA keep all intensional data at the location where their corresponding XML data will be appeared. This solution make AXML document look simple but it results in expensive costs for query evaluations and document maintenance, redundant information of Web services and expensive AXML data exchanges (see Section 5. AXML data exchange). The AXML representation by LTU seems a bit more complex because of webServiceInfor fragment in every AXML document. However, this representation is more efficient for further implementation of AXML systems, which will be seen in AXML Data Exchange and AXML Query Processing sections.

4. AXML systems architectures

There are three architecture proposals for AXML systems, each of them built and based on their representation and algorithms for data exchange and query evaluations. The first two subsections briefly describe the two existing architectures and the last one is the new initiative.

4.1. Gemo’s AXML system architecture

The initial AXML system relies on current technologies including XML, Web service, SOAP and WSDL. The system deploys four software programs: Tomcat 5.5, eXist, Axis2 and AXML service calls execution engine (Phan et al., 2011; Ghitescu and Taroza, 2008). The system is configured with a P2P network in three possible scenarios (see Fig. 4).

Each AXML peer is a repository of AXML documents. In the system, each peer performs three functions as a Web server, a client and an engine. As a Web server, each peer provides various services including query and materialization services. As a client, each peer is equipped with a SOAP client and web interfaces to assess, optimize and access AXML documents. As an engine, each peer includes a database access layer, a document manager for organizing AXML documents and a materialiser that provides information on how to evaluate AXML documents (Phan et al., 2011; Ghitescu and Taroza, 2008).

Figure 5 depicts the basic architecture of Gemo AXML systems, where each peer can collaborate with other peers by exchanging messages. A peer can send requests to other peers and receive returning results that are conveyed by SOAP messages.

The AXML system is equipped with essential Web services, some of the most important ones listed as follows:

- Receive: to obtain and manage returning results from a service call.
- Send: to pass data to a specified address.
- New Node: to create and install the new AXML data peer’s repository.
- Materialize: to contain various operations such as evaluate (to materialize the document in a depth first manner), evaluateNode (to materialize the document starting at a specified node), activate (to activate a specified service call), etc.
- GenericQueryService: to receive a query declaration and parameter streams and answers XQueries over the database.
- DummyStreamService: to test the streamed-back results of a specified query.
- OptimaxService: to act as a distributed query optimizer.

4.2. ARAXA’s AXML system architecture

Each AXML peer in the ARAXA project consists of two main modules: Control Module and Integration Module (see Fig. 6). The Control Module is composed of four modules: (i) a Service Call Catalog to keep service calls which are updated after each service invocation; (ii) a Service Manager to manage and execute service calls; (iii) a Result Manager to manage and materialize obtained

Table 4 Comparison of three AXML representations.

<table>
<thead>
<tr>
<th>CRITERIA</th>
<th>Gemo AND ARAXA</th>
<th>LTU</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Represent intensional data</td>
<td>Yes</td>
<td>Yes</td>
<td>All three representations are able to describe and embed intensional XML data in XML documents.</td>
</tr>
<tr>
<td>Data encapsulation</td>
<td>Yes</td>
<td>Yes</td>
<td>All representations are able to encapsule intensional data.</td>
</tr>
<tr>
<td>Simplicity</td>
<td>Fine</td>
<td>Complex</td>
<td>LTU representation is more complex than the others.</td>
</tr>
<tr>
<td>Stability</td>
<td>Yes</td>
<td>Yes</td>
<td>Noted that ARAXA requires more time and resource for mapping and re-construction AXML documents from XML to Object Relational DBMSs</td>
</tr>
<tr>
<td>Support performance improvements</td>
<td>No</td>
<td>Yes</td>
<td>LTU representation enables more efficient node searching and efficient mechanism to determining relationships of intensional nodes with other nodes in AXML documents.</td>
</tr>
<tr>
<td>Support intensional data</td>
<td>No</td>
<td>Yes</td>
<td>LTU representation is more efficient for checking the correctness of arguments, signatures and other parameters of Web services</td>
</tr>
<tr>
<td>management aspects</td>
<td>Hard</td>
<td>Easy</td>
<td>The gathering of intensional data in webServiceInfor makes LTU’s representation more efficient when intensional data are evolved or changed.</td>
</tr>
</tbody>
</table>
results; and (iv) a Monitor Agent to observe the system clock during the activation of service calls. The Integration Module is a client application. It contains the XML-Relational Mapper that is responsible for mapping and storing AXML documents. In the mapping process, service calls will be identified and stored in the Service Call Catalog of the Control Module. The Query Translator is used for translating XQuery or XPath to SQL as well as to reconstruct the returned results to the XML data.

ARAXA use relations/tables for service call storage (Ferraz et al., 2010; Ruberg and Mattoso, 2008). These service calls are processed using two execution modes: lazy mode and continuous mode. With the lazy mode, the service calls are observed during the query processing and the obtained results will be returned as the result of a query. With the continuous mode, the service calls are monitored at all times and are activated regularly. ARAXA introduces a new method, execute service (), to activate Web services. This method operates independently from the database systems. The class diagram (see Fig. 7) depicts the association of ARAXA infrastructures with the DBMS (Ferraz et al., 2010; Ruberg and Mattoso, 2008).
4.3. LTUAXML system architecture

Based on earlier work (Phan et al., 2011; Phan and Pardede, 2011; Phan et al., in press), a new initiative for an AXML architecture has been proposed (see Fig. 8). This AXML system is built on top of an XML database system. The main block comprises AXML processing modules. The modules are responsible for receiving requests, analyzing and processing intensional data before passing it through to the XML DBMS for evaluation. The modules are also responsible for assembling results arriving from XML DBMS and from other peers to finalize results.

Figure 9 depicts the main modules in the proposed architecture: Structure Examiner, Intensional Node Detector, Query Decomposer, Task Manager And Distributor, Materializer, and Result Assembler.

Structure Examiner compares desired data structures between requesters and providers if needed. The data structures are provided in a SOAP message and by extraction from webServiceInfor of AXML documents in the provider. If the data structures are matched, Intensional Node Detector will analyze and find intensional nodes related to the query from requesters. This module and Query Decomposer will indicate a sub-query for intensional nodes found by Intensional Node Detector.

Task Manager and Processor decide which intensional nodes related to the query will be materialized and sends them to Materializer. The Task Manager and Processor also specifies queries that will be evaluated against AXML documents in this peer and send these queries to the XML DBMS for evaluation. This module also collects results from XML DBMS and Materializer and sends them to Result Assembler for results finalization.

Materializer invokes Web services from other peers (this peer acts as a client side) or from the Web service existing through itself (playing a role as a Web services provider). Results from this module can be used to update AXML documents or be directly sent to Result Assembler module. Note that the Materializer will use webServiceInfor to find the most suitable Web services for each invocation, if possible.

Result Assembler gathers and assembles all results from other modules. During assembly, Result Assembler will consult Structure Examiner to find acceptable results to be returned to requesters. If the requesters require more materialization of intensional nodes, Result Assembler will require Task Manager and Processor to again process those nodes.

5. AXML data exchange

In AXML systems, XML and intensional XML data are streamed between peers. Each peer can request data from other peers, as well as providing data for other peers. Some intensional data needs materialization but others do not. Due to this difference in
treatment of intensional data, a mechanism to control intensional data exchanged between peers is required, especially since the cost to materialize and convey a large amount of real XML over the network is high.

In the three proposed AXML systems, there are two trends to control data exchanges in AXML systems. Gemo (Milo et al., 2003) and ARAXA (Ferraz et al., 2010) apply the same algorithm. The algorithm finds intensional data related to requests for data exchanges, establishes schemas for data exchanges, and then materializes these intensional data. In the LTU’s algorithm proposed by (Phan et al., in press), most intensional data remain in their original form. However, there are subQueries, which are automatically generated from requests for data exchanges, attached in these intensional data. This section will examine the data exchange algorithms in AXML systems.

5.1. Gemo’s proposal for exchanging intensional XML data

Frequently, when a master peer queries an AXML document, it wants its intensional nodes to be materialized. In order to achieve this, the query and the desired structures of results will be sent to a slave peer. According to (Abiteboul et al., 2004a, 2004b, 2004c, 2004d), sub-queries will be extracted from the query sent by the master peer to filter results. This implementation applies ideas in distributed database systems, which is query shipping instead of data shipping (Kossmann, 2000). Results from the invocation of Web services inside intensional nodes after filtration will be compared with the desired data format—XML schema_int (see (Milo et al., 2003) given by master peers (Milo et al., 2003).

If data and desired format are not matched, the intensional data inside the invocation results will be materialized to find data which complies with the desired structure. This process is called rewriting documents (Milo et al., 2003) or is known as materializations of intensional nodes. If these rewritten documents cannot find matches between data and schema_int, failure reports will be returned to requesters.

The schema to control the exchange of intensional data is an XML schema extension called schema_int (Milo et al., 2003). It is not used only to specify data format but also to control what nodes in the exchanged data can be intensional. The determination for allowing intensional data exchange in schema_int is based on statistics of performances such as workloads and costs of communication, capabilities of peers, security issues and functionalities such as confidentiality (Milo et al., 2003).
The existing technique poses several problems, derived from the controls of the data exchange, as well as the existing representations and query evaluation of AXML systems.

- In addition to the XML schema\textsubscript{int} proposal, (Milo et al., 2003) proposed extensions of WSDL. These extensions increase the complexity of validating extensional XML schema\textsubscript{int}. It also causes potential issues in the further development of AXML systems and AXML-based applications. For example, schema\textsubscript{int} cannot be applied when data providers are not AXML peers. In addition, natural features of exchanging data via Web services in P2P networks are loose-coupled, anonymous, stateless, and heterogeneous. Hence, applying schema\textsubscript{int} into AXML systems to control data exchange via Web services can cause interventions of natural features of Web services or can make the implementation more complex.

- In AXML systems, there are two ways to materialize intensional data exchange between peers: the automatic activation of Web services in intensional nodes; and the passive activation of Web services to process query evaluations (Abiteboul et al., 2004a, 2004b, 2004c, 2004d). In both these cases, schema\textsubscript{int} cannot be applied to improve the performance of the system. For example, schema\textsubscript{int} cannot be used to reduce the time to search the intensional nodes and specify the relationships of those nodes with others in AXML documents.

Applying schema\textsubscript{int} to determine intensional nodes can be problematic to the query evaluation performance. When intensional nodes are instances of nodes in predicates of XML queries, they should be materialized immediately to evaluate other nodes related to them. This is a rather complex process and can cause performance issues. Currently, there is no mechanism to recognize intensional nodes which belong to predicates when evaluating XML queries (Abiteboul et al., 2004a, 2004b, 2004c, 2004d; Milo et al., 2003; Abiteboul et al., 2005). Hence, it is impossible to use schema\textsubscript{int} to stipulate which nodes will be intensional nodes.

- With the current AXML system, intensional XML data are unchangeable during data exchanges because these intensional nodes are the smallest units of intensional data in AXML systems. Thus, it is impossible to return and manage these intensional nodes in requester peers. For example (see Fig. 10), peer P1 requests peer P2 to materialize intensional data along with a sub-query Root/A/B[C=="1"] (assumes that P1 accepts all intensional nodes in the position of nodes A). Results from P2 contain XML data and an intensional node A along with the query A/B[C=="1"] for intensional node A. However, intensional node A cannot remember the query applied on the intensional node. Therefore, intensional node A must be materialized before sending back to requester P1 even though this node was expected to be returned as intensional data.

Current mechanisms can exchange intensional data which are not related to queries only. Intensional data cannot be returned to requesters if requesters’ queries only ask for sub-parts of data provided by Web services. This can be seen in algorithms applied to query evaluations in both pushed and non-pushed query versions (Abiteboul et al., 2004a, 2004b, 2004c, 2004d). In this algorithm, the intensional nodes and newly arrived intensional nodes from service invocations will be repeatedly materialized until there is no intensional node associated with the query.

Currently, the cooperation needed to control data between schema\textsubscript{int} and query algorithms is not efficient. For example, after receiving a request to materialize intensional data, a peer invokes Web services and evaluates sub-queries indicated by requesters trying to look for all intensional nodes related to the sub-queries. If the results do not match with the desired schema\textsubscript{int} from requesters, intensional nodes in the results will be continuously materialized, regardless of the expense associated with materialization. Nevertheless, the results returned to requesters are not always guaranteed to exactly match with the desired schema (Milo et al., 2003).

- Current intensional data exchange techniques might result in redundancy, that is, data which is out of date, and is obliquely transferred between peers.

Generally, there are two main materialization goals in intensional data exchange between peers. The first is to serve query evaluations (Abiteboul et al., 2004a, 2004b, 2004c, 2004d) and the second is to match AXML data with the desired schema\textsubscript{int} (Milo et al., 2003). For example (see Fig. 11), peer P1 requests the materialization of node A at peer P2. Results from peer P2 contain intensional nodes B, C and D. The intensional nodes B and C must be materialized for a sub-query evaluation against the result; intensional node D is materialized to match with a schema\textsubscript{int}. AXML data from these materializations will be sent from peer P3, P4 and P5 to peer P2. Then, all the results are returned to P1. In this simple scenario, AXML data are
indirectly transferred from peer P3, P4 and P5 through P2 to peer P1.

These materialization processes are problematic because they contradict the original ideas of AXML systems, which allow users to materialize intensional data only when needed (Milo et al., 2003). In addition, the results of materialization may not be needed by requesters. Furthermore, materializations can result in data not being up-to-date in cases where data sources are changed frequently. To avoid these unexpected effects, it is therefore necessary to preserve intensional data through exchange and update processes as much as possible.

The shortcomings of current intensional data exchange techniques affect query response times in two main ways. The first is the high cost of checking and processing matches between schema and the results of invocations, which is not only time consuming but also complicated (Abiteboul et al., 2005) and causes others materializations and updates (Milo et al., 2003). The second is the limited ability to store and exchange intensional data due to current AXML representations and query evaluation. Query evaluation steps must be delayed until there are no intensional nodes relating to the query existing in the AXML documents being evaluated. As presented in (Abiteboul et al., 2004a, 2004b, 2004c, 2004d), all pre-processes for evaluations are executed at requester peers, and in query runtime. Unfortunately, these processes consume large amounts of time because of travel, searches, updates and invocations.

- Although Gemo’s AXML system employed Dewey codes to create F-Guides data structures (Abiteboul et al., 2004a, 2004b, 2004c, 2004d; Goldman and Widom, 1997), the cost of adjusting and updating Dewey codes for nodes in AXML documents is not mentioned. This needs to be studied more for comparison between prototypes of AXML systems.

5.2. LTU’s technique for intensional XML data exchange

In the previous technique, intensional data related to a request will be materialized and sent back to the requester and the results will be checked to determine if they comply with expected data formats identified by the requester. This is the shortcoming of the previous technique. An improved technique should be able to keep intensional data, which are being exchanged, as much as possible. The structure validation of XML data corresponding to the intensional data should not be enforced during exchange in order to avoid forced materialization of intensional data.

An initiative for improving intensional XML data exchange is proposed in (Phan et al., 2011). When a master peer requests another peer to exchange AXML data, an additional query Q is sent to the slave peer to filter results before returning back to the master peer. This query will be decomposed for intensional nodes at the slave. To enable this, the new technique allows intensional nodes to retain sub-queries imposed on them by including an element subQuery under sc element. If intensional nodes in the results at the slave peer satisfy requirements including performance, capability, security, functionality (Milo et al., 2003) and do not belong to predicates of Q, then these intensional nodes will be kept in intensional format and sent to the master peers.

The structures of the possible results will be validated against the expected structures from the master peer. It is noted that structures of data will be found in webServiceInfor of the master and slave peer. These structures are also used to find subQuery for intensional nodes in those peers during query evaluation processes.

This new technique offers several advantages as follows.

a. This technique can work effectively without any extension of XML schemas and WSDL. Desired data structures indicated by requester peers and data structures of results will be provided by webServiceInfor. Hence, this proposal can avoid the unexpected side effects of modifying standard XML schemas and WSDL in future developments. Moreover, using adopted standards of XML schemas and WSDL allows other XML peers to communicate with AXML peers via Web services.

b. Element subQuery added inside each intensional node allows the flexible and efficient exchange of intensional data.

c. Element webServiceInfor is not only employed to control data exchange between peers, but also to help simplify the copying
and exchange of intensional data for AXML documents, hence, it is not necessary to look for and copy their separate schema_int during copying and exchanging AXML documents. In addition, webServiceInfor contributes to the enhancement of the performance of AXML systems by reducing the time to search intensional nodes in documents (Phan et al., 2011).

b. This technique reduces the amount of indirect data transferred. In the previous technique, a large amount of data is indirectly transferred through intermediate peers. This practice wastes bandwidth and time. Particularly, the costs of bandwidth and time for data transfers are more expensive when the amount of data exchange is large and there are more nested invocations.

c. The technique reduces computation resources for matching data exchanges between peers. It compares structures of data instead of comparing large amounts of results data with desired data structures, as shown in (Milo et al., 2003). The number of nodes being matched is reduced significantly. As stated in (Milo et al., 2003; Abiteboul et al., 2005), the complexity of matching algorithms in current AXML systems is exponential, whereas the complexity of the LTU’s technique (Phan et al., 2011) is at most $O(n \times m)$.

d. The previous technique needs to travel and compare a large amount of nodes, and also materializes other intensional nodes in query evaluations and tests of matching data with schemaext. Furthermore, the AXML data which have been exchanged are updated in requesting peers. This incurs expensive overheads for AXML systems and is the cause of a long query response time.

This section briefly surveys techniques to exchange intensional XML data. In Gemo data exchange, there are chains of materialization so the amount of intensional nodes being materialized can be large. Intensional data exchanged in Gemo’s prototype (Milo et al., 2003) are mostly matched between requesters and senders by comparing schemaext, however ARAXA relies completely on the preceding algorithm. The new proposal (Phan et al., 2011) is implemented in the opposite way. It tries to postpone materializing intensional XML data until they are needed.

### 6. AXML query processing

One task of AXML systems is to process XML queries against AXML data. Since AXML documents contain both normal and intensional XML data, querying AXML documents is different to querying XML documents. For XML documents, all data in the nodes are explicit so XML data that satisfies the conditions in the queries are selected and extracted as the final results of the queries. In AXML documents, XML nodes and intensional nodes can contribute to the final results to answer XML queries. It is noted that XML data corresponding to intensional nodes are stored by other peers. Therefore, to request XML data for those nodes, Web services, which are stored in those intensional nodes, will be invoked.

Therefore, to evaluate XML queries against AXML documents, some questions need to be answered. Such questions include: does any intensional node need to be materialized?; which intensional nodes should be materialized?; and what are the desired data that should be returned? As AXML query evaluation is accompanied with Web service invocation, it is easy to see that there are three possible solutions to process XML queries against an AXML document: (i) materializing all intensional nodes in AXML documents before querying so that XML queries are evaluated normally; (ii) materializing intensional nodes encountered during the evaluation of XML queries; and (iii) materializing only intensional nodes related to queries before evaluating the XML query as normal (Abiteboul et al., 2004a, 2004b, 2004c, 2004d).

The first solution is the simple way to evaluate queries against intensional XML data. However, it is also the most naive solution because this solution can invoke many Web services which are not used and are unnecessary for query evaluations. This solution causes time wastage, high computing resource usage, high bandwidth, as well as outdated results. However, it can be useful in some cases, for example, if users want to work off-line or if the data is on resource-limited mobile devices.

The second solution seems to be easy, simple and natural but it can cause interventions to query engines. Whenever intensional nodes are encountered, query evaluation processes are postponed until materialization has finished. Moreover, this solution cannot apply optimization mechanisms from XML query engines if the original document is frequently changed.

The third solution can be divided into two steps. The first step is finding intensional nodes related to the query being evaluated, then materializing these nodes. In the second step, the XML query engine will evaluate the query over intensional nodes as normal. This solution is applied in all current prototypes of AXML systems because it can reduce the amount of Web services being invoked. However, the search to locate intensional data related to the query evaluation is still a large concern in the existing AXML systems.

#### 6.1. Gemo’s AXML Systems query evaluation

In the Gemo system, a superset of intensional nodes related to a query is specified, materialized and updated into the AXML document. Finally, the input XML query is evaluated against the evolved AXML documents.

The system performs a pre-query evaluation that guarantees all data in the AXML documents needed for that query to be extensional. However, there is no result for the input query if any Web service invocation fails or if there are infinite Web service invocations.

This query evaluation technique satisfies the requirements of dealing with static and dynamic data. It takes full advantage of existing query optimization algorithms in the XML database systems of each peer. It does not intervene with XML query engines and it is possible to apply parallel invocations and computing in query evaluation processes.

However, this technique has several shortcomings as follows:

a. **Query optimizations issues.** The lack of a global schema causes anonymous peers to participate in query evaluation. In Gemo’s query evaluation, the queries are shipped to query other peers (sites) to filter the results of Web service invocations (Abiteboul et al., 2004a, 2004b, 2004c, 2004d). This acts similarly to implementations in distributed DBMSs so the lack of global schemas is an issue in AXML systems. Moreover, shipping queries or shipping data as well as joining data between peers (Web service providers) is not been fully studied as yet.

b. **Inflexible workload shares between master and slave peers.** Master peers take most of the responsibilities and computations in the arrangement and management of Web service invocations. Master peers are responsible for creating and re-evaluating F-Guides structures (Abiteboul et al., 2004a, 2004b, 2004c, 2004d) and Node-focused queries (NFQ) (Abiteboul et al., 2004a, 2004b, 2004c, 2004d), to search whole documents, to arrange and organize temporal results returned from invocations and so forth. These workloads require resources
and time and should be shared with other peers with strong computation capabilities. Hence, it is hard to apply this technique on devices with limited resources.

c. **Overhead of extra data structures.** It is expensive to build additional data structure F-Guides for AXML documents based on Data-Guides (Goldman and Widom, 1997) when they are not reusable. In addition, F-Guides are created and frequently updated at the time of querying so it increases the query response time. This situation is worse because frequent updates for F-Guides during query evaluation processes are unavoidable.

d. **F-Guides are not effective for determining the relationships of intensional nodes.** F-Guides store the paths of each single intensional node. However, it is not useful to specify the relationships between intensional nodes. Moreover, F-Guides are not able to determine and eliminate irrelevant intensional nodes based on invocations or based on related extensional nodes.

e. **NFQ Algorithm** (Abiteboul et al., 2004a, 2004b, 2004c, 2004d) cost during query evaluation. In the pre-query evaluation, NFQs will be created and executed against AXML documents to find the intensional nodes related to the query and divide them into layers for further materialization. Each NFQ execution needs a whole document search, which is very costly. Moreover, after invocations for each layer, a re-evaluation of NFQ is needed. Hence, it can be seen that many document searches are needed to create and re-evaluate NFQs during query evaluation.

f. **Management of temporary results.** Currently, AXML systems have to spend a large amount of resources and computing power to manage the temporary results from invocations before actually evaluating the queries. During the materialization of intensional nodes, the results returned from the materialization will be updated into AXML documents. This can be expensive, particularly when there are large numbers of invoked Web services because the system also needs to frequently manage, update and delete the temporary results.

g. **Cost of AXML document updates.** Updates of AXML documents which depend on mode attributes (Milo et al., 2003; Abiteboul et al., 2008) can occur unexpectedly when evaluating queries. In practice, updates should be only implemented at the users' request. It is inefficient if there are many updates for every query because update operations are expensive and the temporary results may not be used in the future.

h. **Failed query for unfinished Web service invocations.** In the current technique, if there is unfinished Web service invocation, the query will fail even though in many cases, partial results are still accepted and useful.

i. **Cost of traversing and searching whole documents.** Traversals of whole documents are expensive. However, it is still an inevitable process in the current AXML systems, such as running NFQA, re-evaluating NFQs, creating and updating F-Guides, determining relationships of intensional nodes and AXML document updates. These traversals are applied many times during query evaluation.

j. **The same algorithm to process intensional nodes in context nodes and predicates of XML query.** There are two types of intensional nodes when evaluating XML queries against AXML data. They are intensional nodes belonging to predicates of queries and intensional nodes belonging to context nodes of queries. Intensional nodes in predicates need to be invoked immediately to prune and process their related intensional nodes and other XML nodes. When materializing intensional nodes in predicates, we may not need to send sub-queries and desired data structures to slave peers to filter invocation results because data types of those nodes are simple. Intensional nodes belonging to context nodes can be processed later. XML data from the materialization of those nodes can be large and complex.

### 6.2. ARAXA's AXML Systems query evaluation

Based on execution modes, there are two classifications of intensional nodes. The first one is the lazy mode, where intensional nodes are materialized to serve query evaluations. The second one is continuous mode, where intensional nodes contain continuous Web service calls.

The former needs to be observed during the query evaluation process because the obtained results will be used to answer given queries. The second one must be monitored at all times to activate them regularly. The ARAXA system uses two relations to store the Web service calls of intensional data namely Service call (id, path id, dewey, doc id, serviceURL, method Name, serviceNameSpace, useWSDLDefinition, signature, callable, frequency, last-called, followed, mode, doNesting) and Parameter (id, service id, path id, type, name) (Ferraz et al., 2010; Phan et al., in press).

According to (Abiteboul et al., 2004a, 2004b, 2004c, 2004d), XML query evaluation against AXML data in ARAXA follows Gemo's algorithm presented in (Abiteboul et al., 2004a, 2004b, 2004c, 2004d). Materialization of intensional data to evaluate XML queries incorporates seven steps (Ferraz et al., 2010; Phan et al., in press): (i) looking for appropriate services to answer a given query; (ii) translating the given query into SQL; (iii) finding dependency of the service calls; (iv) arranging these service calls; (v) storing obtained results in corresponding relational tables by mapping; (vi) evaluating the given query, and (vii) reconstructing results to XML and sending to users.

It is noticed that there are different ways to invoke service calls during the materialization process. Therefore, optimization strategies are essential in order to make the performance effective. ARAXA applies techniques proposed by (Team, 2005; Tatarinov et al., 2002) to optimize the materialization process and to avoid invoking unnecessary service calls. For example, in a relational table book (author, price, ISBN), price is dynamic data provided by a service call with the parameter ISBN. When evaluating a query related to price information, dynamic data price must be materialized by invoking the price service. Figure 12 demonstrates the translation process in ARAXA.

ARAXA query evaluations have both advantages and disadvantages which also exist in the Gem’s AXML system because it adopts algorithms from (Abiteboul et al., 2004a, 2004b, 2004c, 2004d; Milo et al., 2003). In addition, there are other advantages and disadvantages because of the particular features of OR DBMSs.

XML query evaluations and intensional data materialization are deployed over OR DBMSs so this can take advantage of matured technologies and stable techniques of database systems such as data management, searching, updates and query optimizations and so forth. The greatest disadvantage of this implementation is the cost of mapping AXML data to OR DBMSs, decomposing XML query into sub-queries, converting XML queries to SQL queries and reconstructing AXML data from OR DBMSs.

Processes to find intensional nodes related to queries being evaluated can be nested and can occur at any time during query evaluation processes. Therefore, the seven steps of materialization in ARAXA can encounter problems or is not optimized yet because the latter six steps must wait until the first step is finished.

Based on studies of the previous query evaluations in AXML systems, (Phan et al., in press) proposes other algorithms to evaluate an XML query against AXML data.
6.3. LTU’s AXML system query evaluation

A new initiative for AXML Query Evaluation has been proposed in (Phan et al., in press) with the following main ideas.

a. Exploit the new AXML representation (Phan and Pardede, 2011) to manage intensional nodes and to detect relationships among intensional nodes. The new representation will also allow the system to invoke the available and most effective alternate Web services.

b. Classify input queries so that queries which are not related to intensional data can be evaluated as normal XML queries without employing the AXML engine.

c. For each query, divide intensional nodes into two groups, namely ICo (intensional nodes that belong to predicates of the query) and ICa nodes (intensional nodes that do not belong to predicates of the query). This helps to efficiently process intensional nodes and enable parallel processing.

d. Request slave peers to be more involved in query evaluation processes. Slave peers will invoke Web services and evaluate sub-queries corresponding to ICa nodes. New intensional nodes related to sub-queries appearing in materialization processes will be materialized by those slave peers.

e. Store invocation results into memory or temporary documents instead of updating them in the original AXML documents.

This proposed technique includes an algorithm for both master peers and slave peers (Phan et al., in press). The master peer will find and determine ICa nodes for a query and indicate sub-queries for those nodes. Then, those nodes and their sub-queries will be sent and processed at the corresponding slave peers. Master peers evaluate the query against the AXML document while slave peers process the nodes. During query evaluation, master peers will collect the positions of the ICa nodes encountered for assembling the final results. After the query evaluation in the master peers, the results will contain all extensional nodes as well as the positions of the required ICa nodes. These nodes’ positions will be replaced by the invocation results from the slave peers. In cases where some intensional nodes cannot be materialized, these nodes can be kept in the final results with a report on failed materialization. These partial results can also be useful in some cases.

Slave peers, which are Web service providers, apply the same algorithms in master peers after invoking Web services in intensional nodes. The results of Web service invocations and the sub-query will act as AXML documents and a query in master peers, respectively. New intensional nodes can appear at the slave peers, after Web service invocations. If these new nodes are related to sub-query evaluation, they will be materialized by the slave peers. In this circumstance, the slave peers will become master peers. In the LTU’s technique, the intensional nodes’ positions in the queries are also considered to improve the performance of the query evaluation. Based on the nodes’ positions, two types of queries can be classified.

- **Query type 1**: XML queries unrelated to intensional XML data, where every node in the path of the query does not have intensional instances.
- **Query type 2**: XML queries related to intensional data, where all nodes, including context nodes and nodes in the paths of those queries, can have intensional instances.

Instead of using the AXML engine, we can evaluate type 1 queries with a normal XML engine, which will be more efficient since the latter requires less computation. For type 2 queries, the materialization of ICo and ICa nodes are dependent on their precondition nodes. To examine the preconditions of arbitrary intensional nodes, backward and forward traversals of these nodes are required. To facilitate multi-direction traversals, Dewey encoding will be applied for labeling nodes in AXML documents. Full algorithms for this new initiative can be found in (Phan et al., in press).

The new algorithms to evaluate AXML queries offer several advantages as follows.

- The LTU’s algorithm can exploit default optimizations for querying XML documents because the AXML documents are not changed in the query processing period. In addition, the algorithms do not intervene or force XML query engines to...
The LTU’s algorithms take full advantage of the P2P network.

The algorithms do not need to use expensive F-Guides and NFQA algorithms. These two algorithms require whole document traversals and updates of AXML documents whenever any new Web service is received during the query evaluation process. It is also noted that F-Guides are created and updated during the query so the cost of query evaluation is significantly reduced.

These algorithms require fewer intensional nodes to be materialized in comparison to the existing algorithms. IC0 and ICa nodes are only materialized when all of its precondition nodes (including extensional and intensional nodes) satisfy the query. These intensional nodes are elaborately filtered, based on extensional and intensional predicates.

Workloads for query evaluation are shared as much as possible among peers. Other peers should be involved in query evaluation processes and not only for Web service invocations. Filtering data before sending it to master peers can reduce the amount of data exchanges between peers, save bandwidth and reduce the cost of data management and processing at master peers. By sending sub-queries to the slave peers of the Web service providers, the LTU’s algorithms will eliminate redundant data before sending them back to requester-master peers. The advantages of the new algorithms are that they control the amount of data exchanges and reduce the amount of data exchanges between peers. Moreover, they assist in reducing the costs of management for temporary results such as updates, deletions, filters and searches for the required results.

Unlike the previous system, the new algorithms do not require original documents being queried to be updated frequently after each single Web service invocation. In the previous system, the contents of original documents are usually changed by query evaluation. Temporary results from previous query evaluation can be affected by the results of other queries in the future. In addition, temporary results which have been updated in AXML documents need to be removed or this will result in redundancy as well as additional cost for data management.

LTU’s algorithms can save time in materializing specific Web services, in comparison to the previous systems. This is due to the utilization of new AXML representations, where the Web services are located under one fragment of the AXML document only, which is the webServiceInfor fragment.

These algorithms exploit P2P computing power in terms of concurrent processing. In the existing implementation, we need additional computations such as F-Guides, traversals in documents and determinations of relationships (Abiteboul et al., 2004a, 2004b, 2004c, 2004d) to divide intensional nodes into layers so that intensional nodes in each layer can be concurrently invoked in the hierarchy. After performing invocations on each layer, the rest of the layers need to be calculated and updated because of the new intensional nodes received. Hence, it can be said that parallel invocations in each layer must wait for all invocations in previous layers as well as computations to re-classify the rest of the intensional nodes and the newly arrived ones. In the new algorithms, ICa nodes can be concurrently processed. New intensional nodes (including IC0 and ICa) derived from the materialization of existing nodes are divided, managed and processed in slave peers so it helps to reduce computations, classifications of intensional nodes, and the determination of relationships of new intensional nodes with existing ones. Parallel materialization in the LTU’s algorithms work more directly and faster than the existing algorithms.

The LTU’s algorithms take full advantage of the P2P network by exploiting power from other peers, not only by invoking Web services but also by filtering results by enclosed sub-queries. It overcomes shortcomings such as being disconnected by other peers, by having the ability to choose alternative Web services to invoke. In addition, workloads to find, organize and manage Web service invocations are shared with other slave peers so the answer retrieval time is faster. Therefore, this can help to reduce the time needed for computing requests. However, abuses, overuse and the exploitation of the computing power of other peers are issues which need to be investigated because of their adverse effects.

To query AXML documents using this proposal, master peers will receive and manage the least amount of AXML data from invocations because the results are filtered by enclosed sub-queries. These queries also function as a controller for the format of data exchanges.

The new algorithms also have a disadvantage, particularly due to the use of Dewey code to label nodes and facilitate query evaluation. Updating Dewey codes for nodes whenever AXML documents are changed can be expensive.

7. Future research issues

The previous sections have presented the current state of research and development of AXML systems. Even though the benefits of using intensional data through AXML for various applications are apparent, there are many aspects that require further investigation. The issues include distributed query optimization, cost for distributed update propagation, concurrency control, distributed catalog management and strategies to exchange AXML data and securities.

- Query optimization. Similar to a distributed database, query optimization for an AXML system has to take into account the additional communication costs of shipping data or shipping queries from peers to peers. The mechanism to find the available and closest peers to execute a query is necessary. Additionally, query evaluation in AXML must also decide which part of AXML data is being materialized and checked to serve the query.

Although at some level, AXML works like distributed databases, there is no global schema for AXML documents because of anonymous and dynamic peers in P2P architectures. Therefore, AXML cannot efficiently apply query optimization algorithms that have been developed for distributed DBMSs. This issue can be solved either by having new algorithms for non-existing global schemas or by creating a global schema at runtime.

Nested Web service invocation is another issue that needs further investigation. This type of invocation can occur during the materialization of intensional XML data because an invocation of a Web service can trigger invocations of other Web services. These potentially infinite invocations can make AXML system suffer from high query processing costs. The Gemo’s solution is proposed by limiting the level of nested Web services being invoked (Abiteboul et al., 2004a, 2004b, 2004c, 2004d; Milo et al., 2003), by reducing and preventing nested Web services invocation; or by postponing to the invocation of Web services in intensional XML data (Yu et al., 2008).

- Catalog management. There are four common approaches to manage catalogs in distributed databases: (i) centralized, which stores one master copy of the catalog; (ii) fully replicated, which stores one copy of the catalog at each peer; (iii) partitioned, which partitions and replicates the catalog as
usage patterns demand; and (iv) centralized/partitioned, which is the combination of the former three.

Catalog management has not been studied in AXML systems because peers participating in a query evaluation are autonomous. In practice, AXML can facilitate the catalog management feature to enhance system performance to manage replicated data.

- **Update propagation.** AXML needs to replicate copies of data (and even Web services) in some peers. These copies of data must be synchronized whenever one of them is updated. This is problematic because there is no peer which is responsible for synchronization. In addition, there are peers that require updates but may not be available.

- **Concurrence control.** Concurrency control is indispensable for both centralized and distributed databases. Concurrency control in AXML has not been studied yet. Therefore, AXML systems must employ techniques which are widely applied in both centralized and distributed databases such as Time-Stamping and Looking. The issue of concurrency control in AXML is more complex than in distributed databases for several reasons: (i) unidentified location of replicated data or services; (ii) arbitrarily partitioned data; and (iii) the inexistence of global schemas.

- **Strategy to exchange AXML.** Data exchange in AXML systems occurs during query evaluation processes or upon users' demands. As original ideas of AXML (Milo et al., 2003), all intensional XML data which are not immediately needed for query evaluation will be exchanged as intensional data. There are two main issues in intensional data exchange. The first issue is the determination of those parts of intensional data that must be materialized immediately and those parts that should be kept as intensional data. The second issue is the validation of the result to satisfy requirements of requester peers.

Currently, the decisions to materialize intensional data rely on the capability of devices, the cost of communication, the structures of data exchanged and so forth (Milo et al., 2003); or are based on the relationship between intensional data and the queries being evaluated (Phan et al., 2011, in press).

The lack of a global schema in distributed environments of AXML increases the difficulty of intensional data exchange, making the cost of matching data structures between peers expensive because of chain materializations. Hence, a study is needed to investigate how to create temporary global schema and how to check the compliance of structures of intensional data exchanged.

- **Securities.** AXML systems are constituted of Web services, XML technologies and use P2P architectures. Therefore, security in AXML systems is more complex than in each of these components. AXML documents are able to receive numerous Web services (intensional data) during materialization processes but these Web services can return contents that contain unwanted and harmful data. There is only a minor study (Milo et al., 2003) regarding security in current AXML prototypes. Therefore, security is an aspect that AXML systems require before the systems can be considered stable and reliable.

8. **Conclusion**

Active XML (AXML) is a new proposal to extend the power and capability of XML with Web service technologies and P2P architectures. An AXML system can be considered an extension to distributed XML databases, since it has the capability to manage and integrate data from various sources including Web services.

This paper has surveyed several issues in AXML systems including AXML representations, AXML system architectures, AXML data exchange strategies as well as query processing algorithms in AXML systems. In the discussion, the paper describes three existing AXML systems. The paper also presents future issues for investigation in AXML research.

**References**


