Inf Syst Front (2013) 15:203–222
DOI 10.1007/s10796-012-9363-z

On the improvement of active XML (AXML) representation and query evaluation

Binh Viet Phan · Eric Pardede · Wenny Rahayu

Published online: 15 July 2012

Abstract Active XML (AXML) as intensional data aims to exploit potential computing powers of XML, Web services and P2P architecture. It is considered a powerful extension of XML to deal with dynamic XML data from autonomous and heterogeneous data sources on a very large scale via Web services. However, AXML is still at an immature stage and various issues need to be investigated before it can be accepted widely. This paper will focus on two issues facing the current AXML system, namely the representation and the query process. We propose superior representation and improved query evaluation for AXML. For justification purposes, we compare our proposed algorithms with the existing algorithms.

Keywords Active XML · AXML representation · AXML query evaluation algorithm

1 Introduction

Recently, the widespread use of XML, the development of Web services and the powerful computing ability of Peer-To-Peer (P2P) architectures have changed and affected distributed data management as well as Web technologies.

B. V. Phan (✉) · E. Pardede · W. Rahayu
La Trobe University, Kingsbury Drive, Bundoora, VIC 3083, Australia
e-mail: vbphan@students.latrobe.edu.au

E. Pardede
e-mail: E.Pardede@latrobe.edu.au

W. Rahayu
e-mail: W.Rahayu@latrobe.edu.au

XML is a flexible and scalable text-based language which can capture semi-structured data (Bray et al. 2008; Bradley 1998; Hoque 2000; Harold and Means 2002). It has been approved and accepted as a standard format for exchanging and publishing data over the web. Web services are proposed as a new paradigm to integrate Web-based applications, and are effective tools in removing the antagonism between diversified systems and platforms. Web services offer infrastructures for distributed computing and have become one of the most important information providers over the internet (W3C 2004; Alonso et al. 2004). In addition, P2P architectures, considered the third stage of the development of the internet (Pras et al. 2007), are claimed to have potential computing power and are able to support various properties needed in current Web applications such as exchangeability, heterogeneity, scalability of data and autonomous systems. To exploit the capability of the three technologies, (Milo et al. 2003; Abiteboul et al. 2008; GEMO 2007) has proposed a new XML extension, called Active XML (AXML).

The main goal of this XML extension is data and services integration. This means integrating static XML data with its dynamic components via Web services from autonomous and heterogeneous sources on a large scale. The advantages and remarkable capabilities of AXML are proved through prototypes and case studies in Abiteboul et al. (2004a), Abiteboul et al. (2003), Vidal et al. (2008), Abiteboul et al. (2009).

However, AXML is still in a development stage so it has several shortcomings. This paper will focus on improving the current AXML representation and algorithms to query AXML data. These two issues are of vital importance to ensure AXML systems perform effectively.

The rest of paper is structured as follows. Section 2 will introduce AXML systems as well as its current problems in terms of representation and query evaluation. Our solution for AXML representation and query evaluation will be proposed in Sections 3 and 4, respectively. We will evaluate our proposals in Section 5. Finally, Section 6 will conclude the paper and summarize potential future work.

2 Related work

AXML was developed to manage regular XML data as well as dynamic parts of data, namely intensional data (Milo et al. 2003; GEMO 2007; Abiteboul et al. 2004d). Intensional data can be seen as *metadata* that provides information used to retrieve explicit data from Web services. Intensional XML data includes information about Web services, their parameters and their control information for service invocation and result storage. Intensional XML data facilitates, instructs and provides the means for retrieving explicit XML data from multiple sources via Web services. This section will introduce some background to AXML and discuss problems with its current representation and query evaluation.

2.1 Active XML: An overview

The main purpose of AXML is to integrate XML data and intensional data via Web services (Milo et al. 2003; Abiteboul et al. 2009, 2004b). AXML offers potential advantages in exploiting the computing power of decentralized network architectures and Service-Oriented architectures. Moreover, AXML can be applied to save internet bandwidth, retrieve on-demand data, and achieve dynamic as well as up-to-date XML data (Milo et al. 2003; Abiteboul et al. 2004d, c).

AXML representation AXML documents are valid XML documents and comply with XML W3C standards (Bray et al. 2008; Milo et al. 2003). AXML documents include regular (static) and intensional (dynamic) XML data. Intensional data is placed as nodes in XML trees and is materialized into normal XML data when requested. Since intensional XML data is not real XML data, it needs a particular definition to distinguish it from normal XML data.

Milo et al. (2003) and Abiteboul et al. (2004d) proposed special elements called sc to represent intensional XML data. These sc elements contain information for invoking Web services to retrieve their matching XML data. AXML documents are modeled as ordered trees with two kinds of nodes: (i) XML data nodes and (ii) intensional (function) nodes sc (Milo et al. 2003; Abiteboul et al. 2008; The Active XML Team 2005). When an intensional node is being requested, its XML data received from the invocation will replace or append as siblings of the intensional node (Milo et al. 2003; Abiteboul et al. 2004d). Other attributes of intentional nodes such as *timeStamp* and *id* can be added into returned results for tracking and statistical purposes. In its current state, elements sc consist of four groups: (i) Web service information for activation; (ii) information to specify how to activate the Web services; (iii) result handling to indicate how results of invocation will be held; and (iv) parameters for Web service calls (Milo et al. 2003; The Active XML Team 2005; Ruberg and Mattoso 2008).

The current representation can manage both static and dynamic XML data. However, the spread of intensional nodes in AXML documents is concerning because the high cost of searches and updates is unavoidable. In addition, redundancy and repetition of data can create data management problems. Issues relating to the current representation will be analyzed and discussed in Section 2.2.

AXML architecture The current AXML systems are comprised of AXML Peers. AXML peers are built from four different types of software: the Tomcat web server, eXist native XML database, Axis2 Web service engine, and the AXML engine (Milo et al. 2003; Abiteboul et al. 2008). Each AXML peer performs as a server; a client and an engine (see Fig. 1). Each AXML peer can provide a collection of Web services that can be invoked by other peers. It also stores AXML documents in its own AXML repository. The AXML engine in each peer can provide access and evaluate XML queries against its AXML repository.

AXML query evaluation One key task of the AXML system is to efficiently evaluate XML queries against AXML data. The goals of AXML research, including representation, repository platforms and rewriting techniques are to make data query evaluation more efficient.

Since AXML documents contain both normal and intensional XML data (Milo et al. 2003; Abiteboul et al. 2004d, b; Ferraz et al. 2007), there are differences between querying XML and AXML documents. For XML documents, all data in the nodes are explicit so XML data that satisfies conditions (predicates) of XML queries are selected and extracted as the final results of the queries. However, in AXML documents, XML nodes and intensional node can contribute to the final results to answer XML queries. It is noted that,

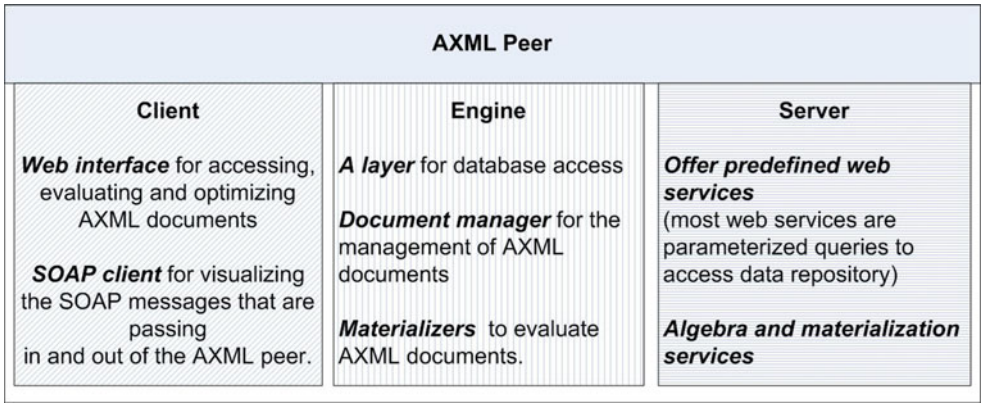

Fig. 1 AXML peer architecture

XML data corresponding to those intensional nodes are stored at other peers so 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 (Milo et al. 2003; Abiteboul et al. 2004b). Such questions include: do any intensional nodes 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 XML data in AXML documents before querying so that XML queries are evaluated normally; (ii) materializing only intensional XML data related to queries before evaluating the XML query as normal; and (iii) materializing intensional data encountered during the evaluation of XML queries (Abiteboul et al. 2004b).

The first solution is the most simple when querying intensional XML data, which is the materialization of all intensional XML data. However, it seems to be the naivest solution because this solution can invoke many Web services, which are not used and unneeded for query evaluations. This algorithm will cause time waste, high computing resource usage, high bandwidth (Abiteboul et al. 2004b), as well as outdated results. However, it can be useful in some cases, for example, if users want to work off-line or for resource limited mobile devices.

Current AXML systems (Milo et al. 2003) apply *the second solution* (Abiteboul et al. 2004b). A superset of intensional nodes related to the query is specified, materialized and updated into the AXML document. Finally, the input XML query is evaluated against the evolved AXML documents (Abiteboul et al. 2004b). This implementation can take full advantage of existing query optimization algorithms in XML database systems. It also does not intervene with XML query engines and it is possible to apply parallel invocations and computing in query evaluation processes. Based on explanations in (Abiteboul et al. 2004b), we can summarize the current implementation to evaluate queries against AXML data as follows, where $\mathcal{Q}$ is a query being evaluated against an AXML document $\mathcal{D}$.

Algorithm 1 Current algorithm for query evaluations

Input: AXML document $\mathcal{D}$, XML query $\mathcal{Q}$.
Output: AXML result$\mathcal{R}$
1 BEGIN
`/* Pre-Query-Evaluation: */`
2 Traverse whole AXML document $\mathcal{D}$ to create "Fast data access" structure *F-Guides* [3] using *dataGuides* [15,12] to facilitate access to intensional nodes whenever it is needed in further steps.
3 Create Node-Focused queries (*NFQ*) for query $\mathcal{Q}$.
4 REPEAT
5 Run an algorithm called *NFQA* on *F-Guides* to search relevant intensional nodes.
6 Classify those intensional nodes into layers to determine a sequence of Web services being invoked.
7 Materialize intensional nodes in each layer based on their relationships.
8 Update original AXML documents by data corresponding to intensional nodes.
9 This new evolved AXML document is named $\mathcal{D}_i$.
10 Update *F-Guides* if any new intensional nodes are received.
11 UNTIL there is no intensional node in *NFQ* trees.
`/* Query-Evaluation */`
12 Evaluate the XML queries on the last evolved XML document $\mathcal{D}_n$ to extract needed AXML data result $\mathcal{R}$ [3].
13 END.

It is noted that when the pre-query evaluation step is finished, it is guaranteed that all data in the AXML documents needed for that query are 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 implementation can take full advantage of query optimizations, it does not intervene with the XML query engine and it is possible to apply parallel invocations and computing in query evaluation processes. However, there are concerns about this algorithm such as the cost of creating and updating *F-Guides* for each concrete query, the cost of traversals of the AXML documents and re-evaluation of *NQF* algorithms as well as rewriting documents. Furthermore, mixing invocations of intensional nodes in predicates and context nodes in queries can create new intensional nodes which will affect the whole query evaluation processes. All of these issues will be discussed later in our proposal.

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

Current practices of AXML data management (Milo et al. 2003; Abiteboul et al. 2004c) satisfy the requirements of dealing with static and dynamic data. However, it also has some performance issues. In the next sections, we will analyze and explain some of these problems.

2.2 Issues with current AXML representation

Milo et al. (2003) and GEMO (2007) proposed a representation for intensional XML data in AXML documents that centered around sc elements. These elements consist of information that indicates which Web services will be invoked, where they are, how to activate them, what their parameters are, and how to store the results. However, the current representation suffers from shortcomings such as wasted storage space, data repetition, it is time-consuming in searching and managing Web service calls, it is an inefficient mechanism to determine relationships between intensional nodes, as well as other issues when dealing with P2P architectures. Each of the shortcomings will be illustrated by using an AXML document example for tourism information (see Fig. 2).

Wasted storage space As shown in Fig. 2, every hotel element can contain intensional nodes such as *getNearbyMuseum*, *getNearbyRestaurant* and *getRatingHotel*. Therefore, similar information of these Web services is repeated in each hotel element. It is easy to see that when the number of hotel elements is large, the amount of repeated information will also be considerably large. Hence, repetitions of data regarding Web services will results in wasted storage.

Repetition of service call and data nodes Intensional nodes with the same information and even the same data nodes can be repeatedly stored under a parent node after some activation of a Web service call under the parent node. For example (see Fig. 3), the first materialization of *intensional Node A* results with *Node 1*, *Node 2*, ... ,*Node n*, and an *intensional Node B*. Then, the second materialization of A will return *Node m*, ... , *Node p*, and an *intensional node B*, and so on. Therefore, if A is materialized many times, returned results can contains many *intensional nodes B* (with the same parameters) and even many repeated data nodes. These repeated intensional nodes can return exactly the same results when they are materialized in the future. This will deteriorate performance and it is difficult to manage.

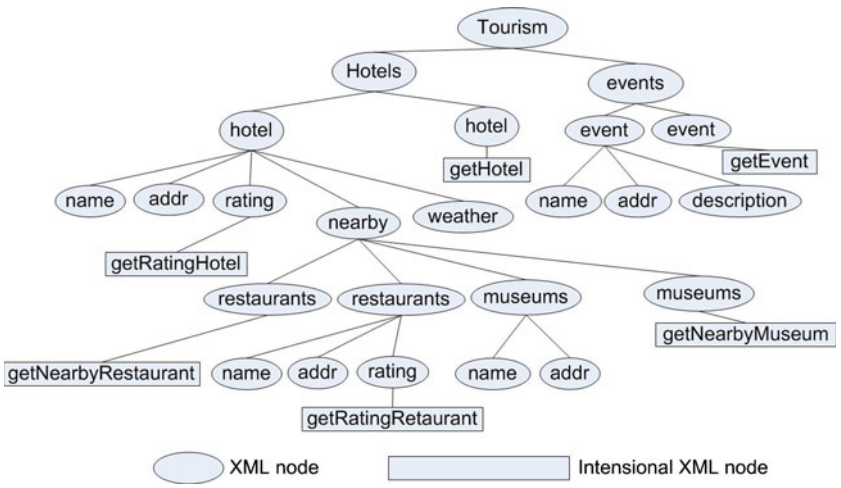

Fig. 2 AXML document example

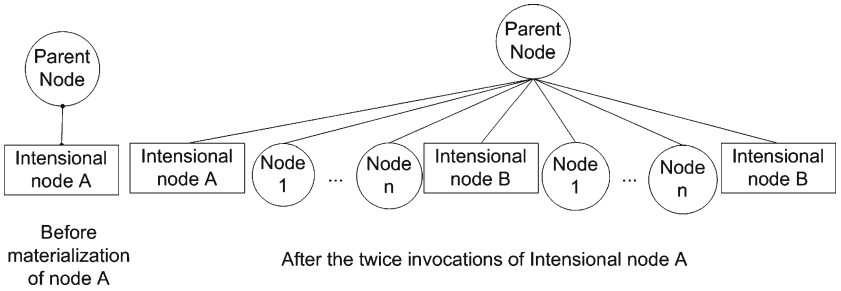

Fig. 3 Repetition of data nodes

Service call management issues It is hard to disable one or more Web service calls in AXML documents for several reasons including security. Currently, an attribute callable with a "false" value is used to counteract Web service calls in sc node and it affects only the Web service in that particular sc node. However, the intensional nodes can appear anywhere in the AXML documents (Milo et al. 2003; Abiteboul et al. 2008, 2004b), so it wastes time and computing resources to search then to deactivate all relevant nodes.

Inefficiency to materialize all intensional nodes in AXML documents or fragments To work off-line or to work with devices of limited abilities, AXML documents can be transformed into pure XML data. Moreover, AXML results from query evaluation against AXML documents can be requested in XML data. In these cases, all intensional nodes in AXML documents or AXML fragments need to be materialized. With the current representation, AXML systems must always traverse and search all intensional nodes in whole documents, determine relationships between them, then invoke their Web services. These processes are time consuming, particularly in large AXML documents. Moreover, conversion issues between XML and AXML documents have not been investigated yet (Milo et al. 2003). These issues are important to deploy AXML systems in the real world because it allows suitable data exchanges between AXML and XML documents, as well as applying a suitable strategy to process them.

Inefficient mechanism to determine dependencies among existing intensional nodes in each AXML document To indicate dependencies among services, additional algorithms such as *F-Guides* are applied but these algorithms can negatively impact the performance of AXML systems. It is even more costly when new intensional nodes appear (see Goldman and Widom 1997).

Lack of support to deal with typical P2P networks characteristics such as unavailability of slave peers If invoking a Web service related to a sc element is the only option, this is problematic in P2P architectures. If we only store one URL and method name, etc. that refers exactly to a Web service as the current presentation, many queries related to that Web service will not be evaluated when the peers providing the Web service are not available. In P2P networks, every peer can join or leave the network at any time so dealing with the unavailability of intensional nodes is an essential issue for AXML systems.

In addition, the representation proposed by Abiteboul et al. (2004d, 2008), GEMO (2007), Milo et al. (2003) creates problems in finding alternative Web services or choosing the most suitable one to invoke. Normally, in P2P networks and distributed database systems, data and Web services should be repeated at some peers (Kemme et al. 2010; Brinkmann and Effert 2008), who assists in ensuring service providers and data are always available. Therefore, AXML representation needs to be improved to facilitate multiple choices in invoking a Web service and dealing with unavailability of peers.

This section has listed AXML representation problems which can affect the future performance of AXML systems. These issues are strongly related to the issues in query evaluation against AXML data. The next section will discuss the shortcomings in query evaluation performance for current AXML systems.

2.3 Issues with current AXML query evaluation

Based on the current implementation of AXML systems, there are several shortcomings in the evaluation process of XML queries against AXML. These shortcomings will be specified and discussed in this section.

Inflexible workload shares between master and slave peers Master peers in current AXML systems still take most responsibilities and computations in arrangements and management of Web service invocations (Abiteboul et al. 2004b). Master peers are responsible for creating and re-evaluating *F-Guides* structures and Node-focused queries *NFQ*, to search whole documents, to arrange and organize temporal results returned from invocations, etc. These workloads require

resources and time and should be shared with other peers with strong computation capabilities. Hence, it is hard to apply AXML systems on devices with limited resources.

Overhead of extra data structures It is expensive to build additional data structures *F-Guides* (Abiteboul et al 2004b; Goldman and Widom 1997) for AXML documents based on *Data-Guides* (Goldman and Widom 1997) when they are not reusable. In addition, *F-Guides* is created and frequently updated at the time of querying so it increases the query response time. Furthermore, it is expensive to update *F-Guides* structures (see Goldman and Widom 1997) whenever new intensional nodes are received. It is worse that frequent updates for *F-Guides* during query evaluation processes are unavoidable.

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

NFQA (Abiteboul et al. 2004b) cost during query evaluation In the pre-query evaluation steps, *NFQs* will be created and executed against AXML documents to find the intensional nodes related to the query and divide them into layers (see Abiteboul et al. 2004b) for further materialization. Each *NFQ* execution needs a whole document search so there are many documents searches in these steps. 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.

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 in "Lazy Mode" (Abiteboul et al. 2004b), the results returned from the materialization will be updated into AXML documents. This can be expensive, particularly when there are a large amount of invoked Web services. It is because the system also needs to frequently manage, update and delete the temporary results (Abiteboul et al. 2004b).

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.

Failed query for unfinished Web service invocations In current query evaluation, if there is any unfinished Web service invocation, the query will fail. However, in many cases, results with some missing parts (partial results) are accepted and useful.

Cost of traversing and searching whole documents Traversals of whole documents are expensive. However, some must still be applied in current AXML systems such as running *NFQA*, re-evaluating *NFQs*, creating and updating *F-Guides*, determining relationships of intensional nodes and AXML document updates (Abiteboul et al. 2004b). These traversals are applied many times during query evaluation.

The same algorithm to process intensional nodes in context nodes and predicates of XML query There are two types of intensional nodes when evaluate 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 materialize intensional 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 materialization of those nodes can be large and complex.

In addition, invocation results can contribute to the final results to answer queries so we need to send sub-queries extracted from input queries and desired data structures to slave peers to filter those invocation results. However, Abiteboul et al. (2004b) has not differentiated these nodes. All intensional nodes are processed the same. Hence, the number of intensional nodes, relationships between intensional nodes during materialization processes can increase. It means that we need to materialize more and more intensional nodes, and determine many relationships between those intensional nodes. This will negatively affect the parallel computing of AXML systems.

3 Proposed AXML representation

In this section, we propose a new AXML representation to solve the current issues discussed above. The

aims of AXML representation are: (1) the efficient management of intensional nodes; and (2) the efficient performance of query evaluation by preparing computations before query execution time. As mentioned, current intensional nodes are represented by sc nodes to store information to obtain XML data by means of Web services. Information stored inside sc nodes can be classified into three groups: (1) information about Web services such as URLs, method names, etc.; (2) their parameters; and (3) control information which is used to indicate how these services are invoked, how to store results, frequency of invocations, etc (Milo et al. 2003; Abiteboul et al. 2004d). However, the current representation can result in inefficient performance during query evaluation as stated in Section 2.2. Therefore, it is necessary to re-organize and improve current AXML representation so that the representation not only serves materialization but also facilitates query evaluation and intensional data management. We propose adjustments to the current AXML representation and the construct structures of AXML documents to overcome the issues discussed. We firstly focus on the heart of AXML documents.

Definition 1 An intensional XML node is a node named sc to store essential data regarding Web services, parameters and control data to invoke the Web service and process invocation results.

Intensional XML nodes (sc nodes) are used as representatives of their corresponding XML data. When intensional nodes are passively or automatically activated (Abiteboul et al. 2008), Web services belonging to those nodes will be materialized to retrieve XML data. The process of obtaining XML data from Web service invocations is called materialization. Sets of intensional nodes are called intensional data. Figure 4 is an example of a general sc node.

These sc nodes store information including the ID of a Web service, parameters for the Web service and other control information to process invocations and invocation results. The first attribute of a sc node is *wsRef*, which holds the ID of a Web service. Element Parameters have child elements named para to store

```
− <element1 ID="nodeID1">
  − <sc wsRef="WSID1 ....">
    − <parameters>
        <para> parameters for Web service call</para>
        <para> ..... </para>
      </parameters>
      <subQuery> sub-query for this node </subQuery>
    </sc>
  </element1>
```

Fig. 4 Example of a general sc node

parameters for Web services. Elements *subQuery* store sub-queries applied on invocation results of this node. In our proposal, intensional nodes only contain a reference to Web services. Information on Web services will be gathered in other places so that we can easily manage all Web services and avoid the repetition of Web services in intensional nodes.

The element *subQuery* is a new element in sc. This *subQuery* element is applied to allow intensional data exchange between AXML peers to be possible and more efficient. For example, in Fig. 2, we want to update a document with data on five-star ('*****') hotels in New York, from an AXML peer P2. A Web service P2, named *getHotels* with parameters location, can provide data on hotels in specified locations. If the AXML representation proposed in Milo et al. (2003), Abiteboul et al. (2008, 2002), The Active XML Team (2005) is applied to exchange data, all exchanged data on hotels must be extensional. This does not satisfy the requirements as stated for AXML systems: data exchange should be intensional so that users have the right to materialize intensional data when needed; and data are up-to-date (Milo et al. 2003; Abiteboul et al. 2004b, 2002). If we employ our representation, the exchanged data can be intensional. These intensional data will include the Web service *getHotels* with the parameter New York and sub query *hotels/hotel[rating = '*****']*.

Definition 2 AXML documents are XML documents containing XML nodes and intensional XML nodes.

AXML documents can become XML documents when all intensional nodes are replaced by their corresponding XML data by invoking Web services. XML documents can become AXML documents when some normal XML fragments are represented by their representative Web service calls.

To assist with efficient intensional node management and the performance of query evaluation, we divide the AXML documents into two sections. The first, called *webServiceInfor*, is a catalogue of Web services and the *ID*s of intensional nodes; the second, called AXML data, contains XML data as well as intensional XML data. Structures of *webServiceInfor* fragments are likely schemas of the AXML data section of the documents. This means that representatives of similar elements' names will be hierarchically stored in *webServiceInfor*. Each element will store the *ID*s of its intensional nodes and Web services belonging to this element's name. These Web services are stored in *ws* nodes. The *ID*s of intensional instances in the data section are stored in *node* nodes.

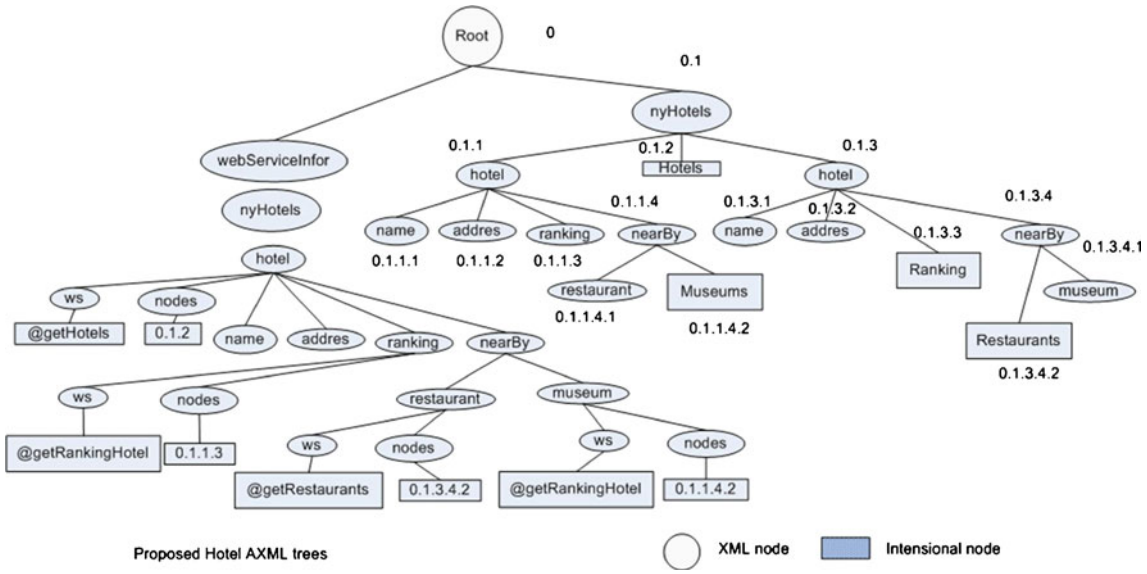

Fig. 5 An example of an AXML tree

Table 1 Elements and attributes in proposed AXML representation

Name	Description	Attributes and child elements
webServiceInfor	It is used to manage information about Web services ws and intensional nodes sc	*elementName*
elementName	It is used 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	*name, ws, node*
name	This attribute specifies the name of elements which will be provided as XML data when their relevant Web services are invoked	
node	Contains the ID of sc node and mode to materialize the node	*scID, frequency*
scID	Contains the ID of sc node to connect to its corresponding sc node	
frequency	Indicates the frequency of materialization of this node	
ws	This element contains information about a Web service call	*wsID, namespace, serviceURL, methodName, signature, useWSDLDefinition* (Milo et al. 2003; Abiteboul et al. 2008; The Active XML Team 2005)
nameSpace	Used in SOAP message to invoke the Web service (Abiteboul et al. 2008; The Active XML Team 2005)	
serviceURL	Used in SOAP message to invoke the Web service (Abiteboul et al. 2008; The Active XML Team 2005)	
methodName	To invoke the Web service (Abiteboul et al. 2008; The Active XML Team 2005)	
signature	To validate the Web service (Abiteboul et al. 2008; The Active XML Team 2005)	
useWSDLDefinition	To validate the Web service (Abiteboul et al. 2008; The Active XML Team 2005)	
sc	Intensional nodes	*Parameters, subQuery*
parameters	It is used to organize parameters for service invocations (Milo et al. 2003; Abiteboul et al. 2008; The Active XML Team 2005)	para
para	Contains concrete parameter for Web services (Abiteboul et al. 2008; The Active XML Team 2005)	
subQuery	Sub-query applied on invocation results	

For example, Fig. 5 is a tree of the *nyHotel.xml* document, where the sub-tree rooted at *Root/nyHotel* is the AXML data fragment and the sub-tree rooted at *Root/webSericeInfor* stores the structure of the AXML data in the document, collections of Web services and the IDs of intensional nodes in the document. In the data fragment, there are many *hotel* nodes under *Root/nyHotels*. The representative of these hotel nodes is named *hotel* node and is stored under *Root/ webServiceInfor/nyHotels*. All *ID*s and Web services of intensional instances of *hotel* nodes in the data section are collected and stored as child nodes of the hotel node in *webServiceInfor*. It is similar for the rest of the nodes in the data fragment.

As *webServiceInfor* fragments store the *ID*s of all intensional nodes of AXML documents, it is fast and simple to locate the positions of intensional nodes in AXML documents. This reduces the time to search intensional nodes for materialization. In addition, Web services for the same elements' names are also gathered, thereby reducing repetitions of Web services in intensional nodes. To facilitate query evaluation, the Dewey decimal system will be applied to label the *ID*s of nodes in the data section. These *ID*s and *webServiceInfor* will be combined to decide whether or not an intensional node is materialized while evaluating XML queries. Nodes *ws* contain attributes including *wsID*, *namespace*, *serviceURL*, *methodName*, signature, *useWSDLDefinition* (see Table 1). Nodes node contain attributes *scID* and frequency, where *scID* is the ID of its intensional node in the data section and frequency indicates mode and frequency of materialization for this node. These nodes' frequency are stored in the reference nodes of intensional nodes in *webServiceInfor* but not in intensional node sc. In the current AXML representation, it is necessary to traverse whole AXML documents to locate intensional nodes which are automatically activated, whereas those intensional nodes can be found by searching in a small fragment of *webServiceInfor* in our proposal. Figure 6 is an example of a general AXML document.

The attribute name in *elementName* is used to indicate the elements that will receive normal XML data by invoking the corresponding child Web service *ws*. These elements are organized hierarchically, based on their positions in AXML documents. All equivalent Web services will be placed under an *elementName*, in which the attribute name is equal to the element name owning these Web services. Therefore, it can be said that Web service *ws* nodes are also stored hierarchically. Positions of intensional nodes can be used to quickly detect relationships between them.

```
<root>
  <!--Web service Information part-->
  <webServiceInfor>
    <elementName name="element1">
      <ws wsID="WSID1"/>
      <ws wsID="WSID2"/>
      <node scID="nodeID1" frequency="..."/>
    </elementName>
  </webServiceInfor>
  <!--AXML Data-->
  <AXMLdata>
    <XMLelement>XML elements</XMLelement>
    <element1 ID="nodeID1">
      <!--This is an intensional elements-->
      <sc wsRef="WSID1....">
        <parameters>
          <para> parameters for Web service call </para>
          <para> ... </para>
        </parameters>
        <subQuery> sub-query for this node </subQuery>
      </sc>
    </element1>
  </AXMLdata>
</root>
```

Fig. 6 Example of a general AXML document

In practice, there are replications of exactly the same Web services in different peers; and there are many equivalent Web services (at a peer or different peers) with the same functions and parameters which are able to provide the same XML data. Therefore, these Web services should be stored and grouped together (under elementName) to assist choosing the most suitable one

```
<root>
  <webServiceInfor>
    <elementName name="newspaper"/>
    <elementName name="title"/>
    <elementName name="date"/>
    <elementName name="weather">
      <ws wsID="WSID1" endpointUrl="http://forecast.com/soap"
      methodName="Get_Temp" namespaceURI="urn:xmethods-
      weather"/>
      <node scID="nodeID1" frequency="lazy"/>
    </elementName>
    <elementName name="events">
      <ws wsID="WSID2" endpointUrl="http://timeout.com/paris"
      methodName="TimeOut" namespaceURI="urn:timeout-
      program"/>
      <node scID="nodeID2" frequency="lazy"/>
    </elementName>
  </webServiceInfor>
  <newspaper>
    <title>The Sun</title>
    <date>04/10/2002</date>
    <weather ID="nodeID1 ">
      <sc wsRef="WSID1">
        <parameters>
          <para>Paris</para>
        </parameters>
        <subQuery/>
      </sc>
    </weather>
    <events ID="nodeID2">
      <sc wsRef="WSID2">
        <parameters>
          <para>exhibits</para>
        </parameters>
      </sc>
    </events>
  </newspaper>
</root>
```

Fig. 7 Example of local newspaper

for invocation. Figure 7 is an example of an AXML document *localNewspaper.xml*.

The Table 1 is a summary of the elements and attributes of the proposed AXML representation.

It is noted that there are some attribute and elements for sc in Milo et al. (2003), Abiteboul et al. (2008), The Active XML Team (2005) which have been removed in our representation. For example, attributes for Web service invocation result handling (*storedMode*) and managing call results (*valid*) Abiteboul et al. (2008); The Active XML Team (2005) are eliminated because the results of Web service invocations are not allowed to be automatically updated into AXML documents during query evaluation. Therefore, AXML documents are not changed after being queried.

This proposed AXML representation and reorganized structures of AXML documents will be exploited during the query evaluation process. The advantages and efficiency of the new AXML representation will be explained in the next sections.

4 Proposed AXML query evaluation

This section proposes and explains the algorithms for XML query evaluation against AXML data. We start with the preliminaries, followed by the query evaluation algorithms.

4.1 Preliminaries

Assume that we will evaluate the following XPath query $\mathcal{Q}$ against an AXML document *doc.xml*. *doc('doc.xml')*/N_1 [n_{11} = 'val_{11}', n_{12} = 'val_{12}'... n_{12} = 'val_{12}']/N_2[n_{21} = 'val_{21}', n_{22} = 'val_{22}'...] /.../N_k[n_{k1} = 'val_{k1}', n_{k2} = 'val_{k2}'...]/.../N_m[n_{m1} = 'val_{m1}', n_{m2} = 'val_{m2}'...].

In AXML documents, nodes can be extensional or intensional. Hence, both context nodes N_i (i = 1,...,m) and predicates n_{ij} (i = 1,..., m and j = 1,..., r) of query $\mathcal{Q}$ can contain intensional instances. For each XML query $\mathcal{Q}$, intensional nodes can be divided into two groups: (i) intensional candidate nodes (*ICa* nodes); and (ii) intensional conditional nodes (*ICo* nodes) based on the positions of these intensional instances.

Definition 3 Intensional candidate nodes (ICa nodes) of an XML query are intensional instances of context nodes in the query path.

ICa nodes can contribute to the results of the query after materialization. For example, for the XPath query *doc('doc.xml')*/N_1/N_2/... /N_k[n_{k1} = 'val_{k1}', n_{k2} = 'val_{k2}' ...]/.../N_m, ICa *nodes are intensional instances of nodes* N_i *(i = 1,2,...,m)*.

Definition 4 Intensional condition nodes (ICo nodes) of an XML query are intensional instances of nodes in the predicates of the query.

Values of *ICo* nodes in predicates are used to determine whether or not their descendant nodes (both extensional and intensional nodes) satisfy the XML queries. The relationships among them are called dependant relations. For example, for the XPath query *doc('doc.xml')*/ N_1/N_2/.../N_k[n_{k1} = 'val_{k1}', n_{k2} = 'val_{k2}'...]/.../N_m, *ICo* nodes are all intensional instances of n_{ij} (i = 1,...m and j = 1...r).

Intensional candidate nodes can appear in the final results of the query, if the results from invocations after filtering are not empty. It is noted that *ICa* node materialization can generate additional *ICo* nodes but this does not happen in reverse.

To reduce the amount of data exchanged between AXML peers, to share workloads and facilitate intensional data exchanges during materialization, master peers will send queries to slave peers to filter invocation results and share computation tasks. These queries are derived from the original query being evaluated by master peers, and stored in elements *subQuery* in intensional nodes sc. Queries sent to slave peers are called sub-queries.

Definition 5 An XPath query SQ is known as a subquery of an XPath query Q if the root node of SQ is a context node of Q, and all predicates of nodes in SQ are exactly the same as those that appear in Q.

For example, assume $\mathcal{Q}$, $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are the XPath queries listed below.

$\mathcal{Q}$ *Root*/N_1*[predicate1]*/N_2*[predicate2]*/.../N_i*[predicate i]*
$\mathcal{Q}_1$ N_1*/[predicate1]*/N_2*[predicate2]*/.../N_i*[predicate i]*
$\mathcal{Q}_2$ N_j*[predicate j]*/.../N_i*[predicate i]* (2 > j > i)

$\mathcal{Q}_1$ and $\mathcal{Q}_2$ are sub-queries of the query $\mathcal{Q}$. Moreover, $\mathcal{Q}_2$ is also a sub-query of $\mathcal{Q}_1$.

Definition 6 For a query $\mathcal{Q}$, AXML document $\mathcal{D}$ with two nodes n_a and n_b of $\mathcal{D}$, the node n_a is called a precondition node of the node n_b (in the context of query $\mathcal{Q}$), if (1) n_a and n_b belong to a path which is an instance of path in query $\mathcal{Q}$ and (2) n_a is the instance node of predicate in query $\mathcal{Q}$.

This means that the value of n_a is one of the conditions in query $\mathcal{Q}$ to determine whether or not n_b is

examined and satisfies Q. If n_a is a precondition node of n_b, n_b is also called a dependent node of n*a*.

For example, in the AXML document *tourism.xml* (see Fig. 2) and a XPath query *doc(tourism.xml) /tourism/hotels/hotel[name = 'Best Western', rating = '*****']/nearby/restaurant[rating = '*****']*, node *getNearbyRestaurants* and node *getNearbyMuseums* have the same precondition node *getRatingHotel*. Node *name* (= 'Best Western') is also a precondition node of node *getRatingHotel*.

AMXL documents contain both intensional and extensional XML data so some queries need to access and materialize intensional data. However, other queries do not relate to intensional XML data. Queries are called related to intensional data if nodes (including predicates and context nodes) which have intensional instances exist in the query path. Otherwise, the query is unrelated to intensional data.

We will propose algorithms that focus on a flexible sharing of workload between master and slave peers, reducing additional algorithms during querying AXML data and reducing update operations as much as possible. Furthermore, our algorithms will apply a divide and conquer strategy, parallel computing and are based on assumptions that peers providing intensional data should take the responsibility to fully materialize their intensional XML data and filter results whenever requested.

In this paper, we apply the proposed AXML representation above to facilitate the proposed algorithm. The *webServiceInfor* is applied to indicate relationships among existing intensional nodes in AXML documents. Moreover, *webServiceInfor* is also useful to analyze and to decompose original queries into sub-queries for intensional nodes related to the queries being evaluated. Furthermore, *webServiceInfor* is also employed to decide the desired format of results such as which nodes can be intensional, etc.

By applying the proposed AXML representation, when an intensional node is materialized, AXML systems are able to choose the available and most suitable peers as well as the Web services to invoke by using information regarding Web services in *WebServiceInfor* fragments. The flexibility of choosing the best Web services for invocations will assist AXML query evaluation to be more efficient.

In addition, master peers not only request slave peers to materialize intensional data but also to share computing tasks and filter results from invocation by the enclosed sub-queries. This means that slave peers will be responsible for the materialization of intensional nodes being sent from master peers, and new intensional nodes generated in the materialization and subquery evaluation process.

However, if a slave peer is busy and cannot fully materialize the intensional node, the slave peer can report that to master peer, and send back its results (that are not fully materialized). The master peer will be able to take responsibility for processing the rest of the materialization.

The proposed algorithm will also employ attributes *ID* of intensional nodes to determine the relationship between two arbitrary intensional nodes. It is noted that these attributes ID are numbered using the Dewey encoding system.

All temporary results from Web service invocations and materialization will not be updated into the original documents. These AXML documents are not changed after query evaluation, unlike the existing system.

In the current AXML systems in Milo et al. (2003), intensional nodes related to a query are not distinguished. All intensional nodes are considered and processed in similar ways. During the materialization process, the algorithm creates sub-queries both for predicate and context nodes. It also creates the schema for data exchanges and finds the relationship between the (predicate and context) nodes and other related intensional nodes.

These processes can be inflexible and inefficient to indicate a superset (see Abiteboul et al. 2004b) of intensional nodes for invocations. Therefore, classifications of different types of intensional nodes will be carefully considered in our proposal to reduce additional computations and costs, to find relationships between intensional nodes, and to detect and control data exchanges between master and slave peers.

The main ideas of our proposed algorithm are:

1. Apply the new AXML representation to 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. Other queries will be processed by different algorithms to make performance more effective.
2. Exploit the new AXML representation to manage intensional nodes and to detect relationships among intensional nodes. In addition, the new representation also allows the system to invoke the available and most effective alternate Web services.
3. Distinguish intensional nodes into two groups, namely *ICo* and *ICa* nodes. This helps to efficiently process intensional nodes and parallel computing.
4. 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 subqueries appear in materialization processes will be materialized by those slave peers.

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

The proposed algorithms include an algorithm each for master peers and slave peers. 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 service 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 proposed algorithm, the intensional nodes' positions in the queries are also considered to improve the performance of the query evaluation. Based on the nodes' positions, we have divided the query into two types as follows:

Query type 1: XML Path queries unrelated to intensional XML data, where every node in the path of the query does not have intensional instances.

Query type 2: XML Path queries, which are related to intensional data. This means that 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 satisfaction of *ICo* and *ICa* nodes are dependent on their precondition nodes. Therefore, each *ICo* and *ICa* node needs to satisfy its precondition nodes before materialization. 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.

Algorithms to indicate types of XPath queries and to evaluate those two query types will be introduced in the next sections.

4.2 Algorithm for classifying queries

The main module in the master peer will be used to classify input queries into three groups and prepare sets of *ICa* and *ICo* nodes as well as sub-queries for those intensional nodes. In this module, *ICa* and *ICo* nodes of input queries will be examined and compared with data in *webServiceInfor* for classification.

After completing this process, the input query is ready to apply different algorithms for query evaluation. Type 1 queries will be evaluated by normal XML engines. To evaluate type 2 queries, additional computations are needed so these queries will be processed by AXML engines.

4.3 Query evaluation for type 1 queries

As mentioned earlier, all type 1 queries will be evaluated as normal XML queries to avoid additional, expensive and unnecessary computations before employing normal XML query engines. Evaluation processes for this query type only happen at the peer containing the AXML document.

4.4 Query evaluation for type 2 queries

Input query Q belonging to query type two will be in the form of $doc(\text{'doc.xml'})/N_1[n_{11} = \text{'}val_{11}\text{'}, n_{12} = \text{'}val_{12}\text{'}...]/N_2[n_{21} = \text{'}val_{21}\text{'}, n_{22} = \text{'}val_{22}\text{'}...]/.../N_k[n_{k1} = \text{'}val_{k1}\text{'}, n_{k2} = \text{'}val_{k2}\text{'}...]/.../N_m[n_{m1} = \text{'}val_{m1}\text{'}, n_{m2} = \text{'}val_{m2}\text{'}...]$, where intensional instances can appear in both context and predicate nodes. This means that N_i and n_{ij} ($i = 1,..., m$ and $j = 1,.., p$) can contain intensional instances.

For this query type, we firstly look for all *ICa* and *ICo* nodes related to query Q by traversing *webServiceInfor* fragments to find all instances of *ICaSet* and *ICoSet*, which were found in the query classifications above.

Nodes in *ICoSet* will be processed before nodes in *ICaSet*. This step is called *ICo* node elimination. Based on the structure of data in *webServiceInfor* and the ID of nodes in the data section of the AXML document, we will locate all preconditions related to each *ICo*

Algorithm 2 Query classification algorithm

```
   Input: AXML document 𝒟, XPath query 𝒬.
   Output: Type of Query, sub-queries for intensional
           nodes, set of ICa nodes (ICaSet), set of ICo
           nodes (ICoSet)
1  BEGIN
   /* Collect all context nodes of 𝒬                    */
2  contextNodeSet := [set of contexts nodes of the query];
   /* Collect all predicate nodes of 𝒬                  */
3  predicateNodeSet := [set of nodes in predicates of the
   query];
   /* collect all instances of ICa nodes               */
4  ICaSet := [];
   /* collect all instances of ICO nodes               */
5  ICoSet :=[];
   /* To classify query into three groups              */
6  queryType := 0;
7  foreach node i in webServiceInfor do
8      if i is intensional node then
9          if i in contextNodeSet then
10             ICaSet:= ICaSet+[intensional child nodes
               of i];
11         if i in predicateNodeSet then
12             ICoSet:= ICoSet+[ intensional child nodes
               of i];
13 if (ICaSet = []) and (ICoSet = []) then
14     queryType :=1
15 else
16     queryType := 2;
   /* Extract Sub-Queries for ICo nodes                */
17 if queryType = 2 then
18     for i in contextNodeSet do
19         sub-query SQ for node i := sub-string of 𝒬
           start from i;
   /* Query Evaluation                                 */
20 if queryType = 1 then
21     evaluate Q using XML engine
22 else
23     Evaluate Q by Algorithm 3;
24 END.
```

node. If there is an extensional node in the precondition of an *ICo* node which does not satisfy the conditions in query Q, we will eliminate that *ICo* node as well as all dependent nodes *ICa* of that *ICo* node from the list of intensional nodes for materialization. When this elimination step is finished, the rest of the *ICo* nodes will be materialized to serve query evaluation.

Next, we will remove the *ICa* nodes (in the *ICaSet*) which are not related to or do not satisfy the conditions in query Q. For each *ICa* node, we also find its precondition nodes based on the structure of *webServiceInfor* and the ID of nodes in AXML documents. If there is a precondition node of an *ICa* node which does not satisfy the conditions indicated in query Q, we remove the *ICa* node from *ICaSet*. Each *ICa* node will be attached to a corresponding sub-query extracted from Q for materialization if those nodes are requested to materialize. Results of materialization will be connected or assembled based on the ID of intensional nodes.

Algorithms for query evaluation are implemented in master and slave peers as listed in Algorithms 3 and 4.

Algorithm 3 Type 2 queries evaluation in master peer

```
   Input: ICaSet, ICoSet, Sub-Queries for context nodes
   Output: Query Evaluation result ℛ
1  BEGIN
2  foreach node i in ICoSet do
       /* Pruning based on extensional
          preconditions of ICo nodes.                   */
3      if there exists a extensional precondition node of
       i, which is not satisfied Q then
4          Remove node i and all dependent nodes ICo
           and ICa of node i from ICoSet and ICaSet
       /* Pruning based on values of node i ICo
          nodes.                                        */
5      else
6          Materialize node i;
7          if value of node i is not satisfied query Q
           then
8              Remove node i and all dependent ICo and
               ICa nodes of node i from ICoSet and
               ICaSet;
   /* Pruning based on predicate of ICaSet. ICa
      nodes unrelated to ICo nodes are needed to
      be pruned.                                        */
9  foreach node i in ICaSet do
10     if (i does not have any intensional ICo node) and
       (any precondition node of node i does not satisfy
       Q) then
11         remove node i from ICaSet;
   /* Materialization process.                         */
12 foreach i in ICaSet do
13     Material intensional nodes i; /* Send information
          W including intensional data of node i,
          along with sub-query SQ for node i to
          corresponding slave peer                      */
14 Result of node i := Result from materialization of
   node i ;
   /* Query Evaluation                                 */
15 Result := Evaluate query 𝒬 against AXML document
   𝒟;
   /* The process will use ICoSet to evaluate
      query 𝒬. The ICaSet nodes will be kept as
      candidate results.                                */
   /* Assembling results from slave peers to
      results in master peers to create the final
      result                                            */
16 Assemble Result with Result of node i;
17 Return Result;
18 END.
```

For example, assume that node H (with label 1.1.2.4), node G (with label 1.1.2.2), D (with label 1.2.1) and G (with label 1.2.2.1) are intensional nodes. The query $\mathcal{Q}$: *A/B[D = '1']/E[H = '3']/G* will be evaluated against the

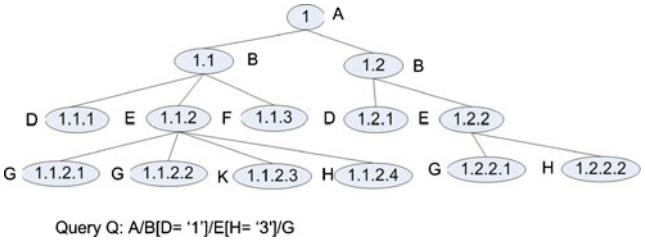

Fig. 8 Example for query evaluation

Algorithm 4 Type 2 queries evaluation in slave peer

Input: Web service information $\mathcal{W}$; Sub-Queries $\mathcal{SQ}$ for corresponding intensional nodes
Output: AXML data
1 BEGIN
2 $\mathcal{RI}$:= Invocation result from invoking the Web service based in W;
3 Implement Algorithm 3 with $\mathcal{RI}$ and $\mathcal{SQ}$ as document $\mathcal{D}$ and query $\mathcal{Q}$, respectively.
4 Return Result to master peer;
5 END.

tree in Fig. 8. *ICoSet* includes D(1.2.1) and H(1.1.2.4). *ICaSet* includes G(1.1.2.2) and G(1.2.2.1).

In the *ICo* node elimination step, node D (1.2.1) will be firstly examined. This node does not have any precondition node so node D (1.2.1) will be kept in *ICoSet*. Next, H (1.1.2.4) will be examined. There is only one precondition node D (1.1.1) for node H (1.1.2.4). If node D (1.1.1) is not satisfied D = '1', *ICo* node H (1.1.2.4) will be removed from *ICoSet*. In addition, node G (1.1.2.2), which is a dependent node of node H (1.1.2.4), will also be removed. After finishing this step, *ICo* nodes will be materialized.

When examining node G (1.1.2.2) in *ICa* node elimination step, the precondition nodes of node G including node D (1.1.1) and node H (1.1.2.4) will be checked. If one of them does not satisfy the conditions in query Q, node G will be eliminated, and so on.

Subsequently, sub-queries for nodes in *ICaSet* will be attached to those nodes for materialization.

When users request XML data from these *ICa* nodes, information on these intensional nodes and sub-queries will be sent to slave peers to materialize. Algorithms in slave peers are similar to those in master peers.

5 Evaluation

5.1 Proposed AXML representation evaluation

In this section, we will evaluate the advantages of our proposed AXML representation in comparison to the current AXML representation. We use the same AXML document described in Fig. 2.

The proposed representation avoids employing AXML engines when processing pure XML documents. In the real world, AXML peers do not work with AXML documents only, but also with pure XML documents. Therefore, when processing a query against XML documents, we can use normal XML engines with additional expenditure from algorithms attached inside AXML. In our proposal, we only need to examine the existence of *webServiceInfor* elements to indicate whether a document is an AXML or XML document. In the current representation, we need to traverse through the whole document to examine the existence of intensional nodes.

The proposed representation saves time in cases invoking all Web services. Converting AXML to XML documents is necessary in cases where XML data is exchanged between AXML and non-AXML systems, requests are received from limited resource systems, as well as for working off-line, etc. It is easy to see that these requests for full materialization are essential and even frequent in the real world, particularly requests to fully materialize results of concrete queries. In these cases, all Web service calls in AXML documents need to be invoked. An advantage of the new proposal is that is saves time when searching to invoke all sc elements in whole AXML documents because we only need to look for sc under *webServiceInfor* instead of traversing whole documents.

Using the document described in Fig. 2, we consider the XML query "List all hotels in Sydney with their ratings". By using the proposed AXML representation, in SydneyTourism.xml, we only need to traverse *webServiceInfo* to find and materialize all nodes *getRatingHotel* under rating whereas the current AXML system has to traverse whole AXML documents, calculate *NQFs* and use *F-Guides* (Abiteboul et al. 2004b) to materialize rating.

The proposed representation can save time and resources to invoke particular Web services, which are activated at a specific time or at constant time intervals because the scope of searching is limited under *webServiceInfor*. Using the *document* in Fig. 2, the getEvents intensional node will automatically retrieve data regarding events, such as daily exhibition schedules, every day at 9:00 am. In the current representation, the *getEvents* node must be located by examining many elements because *getEvents* can appear anywhere in the document. However, we only need to find and specify these intensional nodes in a small section of *webServiceInfor*.

With the proposed representation, it is easy to determine the dependency among Web services without

using any additional algorithms such as *DataGuides* (Goldman and Widom 1997; Eda et al. 2005). This is because in the proposed representation, *webServiceInfor* is organized as a hierarchical tree so dependencies among Web services will be indicated by their position in *webServiceInfor*.

Using the document in Fig. 2, rating is a child node of the hotel node. *getRatingHotel* and *getHotels* intensional nodes provide data for rating and hotel, respectively. Therefore, *getRatingHotel* must be child of hotel 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 label algorithms, the dependencies of Web services can be identified more easily, based on their concrete labels.

The proposed representation offers the flexibility to choose more suitable Web services to invoke. All Web service calls are organized into groups and stored under the same element names. This assists in reducing the effect of peer unavailability when invoking Web services. In addition, it is possible to invoke the required Web services at peers which are not busy, have the nearest distance, have the lowest bandwidth cost, etc.

Using the document in Fig. 2, assume that a tourism office branch in Victoria stores data and services in different AXML peers such as AXMLPeer1, AXMLPeer2 and AXMLPeer3. Data and services are offered by these peers.

We assume AXMLPeer1 is disconnected, AXMLPeer2 is too busy, and AXMLPeer3 is free. If there is a request for information on hotels in Victoria, the proposed representation enables us to choose a Web service in AXMLPeer3 to invoke, since equivalent Web services from these peers are stored under the same element.

In the current representation, there is only one URL endpoint, for example to AXMLPeer1, so every request to this peer will not be able to proceed.

The proposed representation avoids the replication of Web services in each AXML document. In contrast, with the current AXML representation, the same Web services will appear at many sc nodes (Milo et al. 2003; Abiteboul et al. 2008; The Active XML Team 2005). This repetition causes storage waste and management problems in some processes such as searching, materializing, or disabling a Web service.

Using the document in Fig. 2, assume we notice that *getRatingHotel* is suspected of dangerous behavior, so this Web service must be disabled. With the proposed representation, we will find *getRatingHotel* in *webServiceInfor* then update this *getRatingHotel* once only. Using the current AXML representation, we must search for this Web service in a whole document and update it many times when the Web service is found.

The proposed representation is useful in preventing unnecessary updates. In a previous proposal (Milo et al. 2003), it is difficult to control updates on AXML documents because sc elements can appear at any position in the documents. Each sc has its own *storeMode* attributes which will be applied immediately when Web services in that sc are executed. However, in the new proposal, we will remove the *storeMode* attribute and do not allow unexpected update into AXML documents, which are queried. Therefore, it is easy to control updates on AXML documents during materialization processes.

The proposed representation can detect the desired format for results fairly quickly. Whenever a Web service is invoked, the structure of its child elements in *webServiceInfor* requires a desired results format. This expected structure also assists in stipulating which elements should be in intensional data as well as what Web service should be accepted.

Using the document in Fig. 2, when *getNearbyRestaurant* is invoked, it is easy to specify the desired data including restaurants with their child elements such as *restaurantName*, restaurant-Address, and *restaurantRating*.

From this evaluation, we demonstrate that the proposed AXML representation can overcome the shortcomings in the current AXML representation. The proposed representation assists in saving time and computing resources, avoids replications of Web service calls and enables the application of additional algorithms. It also assists in managing Web service calls effectively and enhances the performance of AXML systems.

5.2 Evaluation of proposed AXML query processing algorithms

The proposed algorithms can be applied to exploit default optimizations for querying XML documents because the AXML documents are not changed in the query processing period. In addition, the proposed algorithms do not intervene or force XML query engines to wait for Web service invocations. The materialization of intensional nodes is transferred to other slave peers to invoke and query. We list the evaluation of our proposed query process algorithms as follows.

The proposed algorithms do not need to use expensive *F-Guides* and *NFQA* algorithms because they require whole document traversals and updates of AXML documents, the re-evaluation of *NQFA* (Abiteboul et al. 2004b) as well as *F-Guides* structures (Abiteboul et al. 2004b) whenever any new Web ser-

vice 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 affected.

The proposed algorithms require fewer intensional nodes to be materialized in comparison to the existing algorithms. *ICo* 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.

The current algorithms are not able to eliminate intensional nodes based on satisfying the criteria of related extensional nodes. Hence, there are many intensional nodes, which are not needed for query evaluation, to be invoked. These excessive invocations can result in ineffective performance such as wasted time, resources and computations. Furthermore, with support from *WebServiceInfor*, the proposed algorithms are able to choose which Web services to invoke, that is, those which are the best in terms of availability, distance, etc. by using statistical information and alternative Web service invocations.

For example, we will evaluate a query $\mathcal{Q}$ *Root/A[*a_1 *<= '50']/B[*b_1 *<= '50']/C[*c_1 *<= '50']/D[*d_1 *<= '50']/E* against a tree in Fig. 9A. a_1, b_1, c_1 and E are intensional XML nodes; we assume that their extensional values of those nodes will be smaller than 50. d_1 is an XML node and its value is 60. In current AXML systems, to evaluate $\mathcal{Q}$, all nodes a_1, b_1, c_1 and E are materialized. *NFQA* algorithm cannot eliminate intensional nodes a_1, b_1, c_1 and E from a list of intensional nodes, which will be materialized. In our algorithm, a_1 is an intensional node so related condition nodes of a_1 in predicates of $\mathcal{Q}$ will be examined, particularly extensional nodes. d_1 is extensional node related to a_1, and d_1 is not satisfied condition in query $\mathcal{Q}$. Therefore, a_1 will be removed from the list of intensional nodes being materialized. b_1, c_1 and E are dependent nodes of a_1 so these intensional nodes also removed from the list. Hence, numbers of nodes, which need to be materialized in our proposal, are smaller than that in the current XML systems.

Materialization of unneeded intensional nodes such as A, B, C are also happened, when we evaluate query $\mathcal{Q}_2$ *Root/N[A <= '50', B <= '50', C <= '50', D = '50']/E* against Fig. 9B, where A, B, C are intensional; D is extensional XML node; and value of D is greater than 50.

Workloads for query evaluation should be shared as much as possible among peers. Other peers should be involved in query evaluation processes but 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 proposed algorithms will eliminate redundant data before sending them back to requester-master peers. The advantages of our proposal are that it controls the content of data exchanges and reduces the amount of data exchanges between peers. Moreover, the proposal also assists in reducing the costs of management for temporary results such as updates, deletions, filter and searches for the required results.

Fault tolerances to cope with P2P network characteristics, such as dealing with unavailability of peers, is another aspect that needs to be compared between the proposal and the existing algorithms. The measurement unit for fault tolerance is the probability of completing a query.

If i is ith Web service invocation for intensional node ith; n is the number of Web services being invoked to serve query $\mathcal{Q}$; P_i ($P_i < 1$) is the probability of the unavailability of Web service provider for inten-

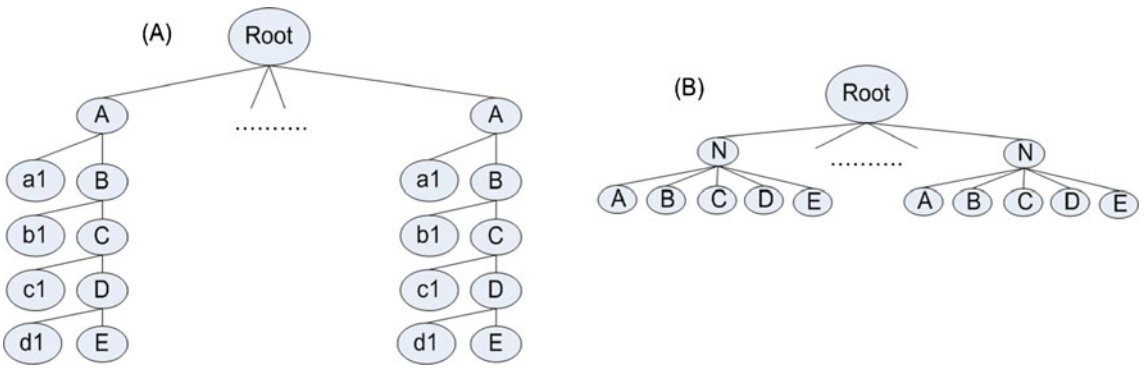

Fig. 9 Eliminations of intensional nodes in query evaluation

sional node ith, the formula to calculate the probability of completeness for evaluating query $\mathcal{Q}$ ($P_{Existing(\mathcal{Q})}$) in the existing AXML systems will be $P_{Existing(\mathcal{Q})} = \prod_{i=1}^{n}(1 - P_i)$.

In our algorithm, there can be k different equivalent Web services for each intensional node i, the probability of those Web services being unavailable is $P_{i1},...,P_{ik}$ ($P_t < 1$, $t = i_1...i_k$) so the probability of the completeness of the materialization of intensional node i is $(1 - \prod_{j=1}^{k} P_{ij})$. Therefore, the formula to calculate the completeness probability of our proposal will be $P_{Our(\mathcal{Q})} = \prod_{i=1}^{m}(1 - \prod_{j=1}^{k} P_{ij})$.

Fault tolerance probability for existing algorithms and our proposed algorithm are $P_{Existing(\mathcal{Q})} = \prod_{i=1}^{n}(1 - P_i)$; and $P_{Our(\mathcal{Q})} = \prod_{i=1}^{m}(1 - \prod_{j=1}^{k} P_{ij})$, respectively. It is noted that $m < n$ because the numbers of intensional nodes being materialized in our algorithms are fewer than those materialized using the existing algorithms. Moreover, $(1 - \prod_{j=1}^{k} P_{ij})$ is never higher than $(1 - P_i)$ because $P_i... P_{ij}... P_k < 1$. Therefore, $P_{Our(\mathcal{Q})}$ is never higher than $P_{Existing(\mathcal{Q})}$. In other words, our algorithm is better than the existing algorithms in dealing with the unavailability of Web service providers.

In current AXML systems, original documents being queried are updated frequently after each single Web service invocation. Therefore, the contents 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.

The new proposal is superior to current mechanisms because it saves time in materializing specific Web services. Using the document in Fig. 2, to materialize all *getHotels* nodes in the existing algorithm, it is necessary to search all *getHotels* nodes in the whole AXML document whereas in our proposal, it only necessary to search in the *webServiceInfor* fragment to invoke the Web services and materialize the indicated nodes.

The ability to perform concurrent computing is an important factor in P2P architectures. The exploitation of P2P computing power in concurrent processing, management and implementation of parallel computing in AXML systems should be investigated because it assists in the reduction of query response time. In the existing implementation, we need additional computations such as *F-Guides*, traversals in documents, determinations of relationships (Abiteboul et al. 2004b) 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 our proposal, *ICa* nodes can be concurrently processed. New intensional nodes (including *ICo* 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 proposed algorithms work more directly and faster than the existing algorithms.

The proposed algorithm takes full advantages 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, the workload in finding, organizing and managing Web service invocations is 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 our 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.

With the proposed algorithm, the original AXML documents are not changed after the query. In current AXML systems, AXML documents evolve after each single query evaluation, having a negative effect on the performance of the system as well as increasing the cost of data management and updates.

Updates are very expensive operators so query evaluation should avoid using updates except for requests for updates from users. With the current algorithm, each single Web service invocation can result in many updates. Moreover, after query evaluation, the current algorithms will require deletion operations for tempo-

rary results. In contrast, with the proposed algorithms, the temporary results are not updated into original documents.

5.3 Quantitative evaluation of proposed AXML query processing

In this section, we use simple queries to compare the query evaluation using existing algorithms and the proposed algorithms.

Cost of query evaluation In this case study, we will use the tree in Fig. 9A, where Root have 100 A nodes. We assume that all nodes A, B, C, D and E can have intensional instances. Fifty percent of each A, B, C, D, and E nodes are intensional instances. Values of these nodes will be equally distributed from 1 to 100. All nodes in result of query evaluation can be intensional. We consider to evaluate query Q: *Root/A[*a_1 *<= '5']/B[*b_1 *<= '5']/C[*c_1*<= '5']/D[*d_1*<= '5']/E* against the tree. We assume that cost to access a node in the tree is C_{access}, and the cost is equal for accessing an arbitrary node. Cost to materialize an arbitrary intensional node is $C_{materialization}$. To evaluate Q, the current AXML systems need to access 193 nodes and must materialize 87 intensional nodes. Therefore, cost of current AXML systems for Q is $C_{Current} = 193 * C_{access} + 87 * C_{materialization}$. Our algorithm access only 94 nodes in *webServiceInfor* section; and materialize 45 intensional nodes so cost to evaluate Q in our algorithm $C_{Proposal} = 94 * C_{access} + 45 * C_{materialization}$.

Fault tolerances For demonstration, we use the AXML document "*tourism.xml*" shown in Fig. 2, describing information on hotels and events. This AXML document contains information including names, addresses and ratings of hotels. In addition, the nearby node consists of other information in relation to hotels such as nearby museums and restaurants. Names and addresses of hotels are extensional XML data, but rating, museum and restaurant can be intensional XML data. Web services *getRatingHotel*, *getRatingRestaurant*, *getNearbyMuseums* and *getNearbyRestaurants* provide data for rating a hotel, rating a restaurant, museums and restaurants elements, respectively. We use three queries below:

Q_1: *doc('tourism.xml')/tourism/hotels/hotel*
Q_2: *doc('tourism.xml')/tourism/hotels//**
Q_3: *doc(tourism/.xml)/tourism/hotels/hotel[name = 'Best Western', rating = '*****']/nearby/restaurant [name = 'KFC' rating = '*****']*

For the three queries, the existing algorithm needs to traverse a significantly larger number of nodes in comparison to the proposed algorithm, due to the fact that the existing algorithm needs to use the *NFQ* algorithm to find relevant intensional nodes. The comparison is shown in Fig. 10.

Using the same AXML document, we also measure the probability of completing a query using our proposed method and the existing method. For this evaluation, we vary the number of intensional nodes from 1 to 15 and the number of available Web service providers for each node between 2 and 3. Assume that the probabilities of incomplete invocations for these nodes are the same and equal to 5 % ($P_i = 0.05$). The comparison (Fig. 11) shows that the probability of completing the query using the proposed method is better than the existing method. It is even better when

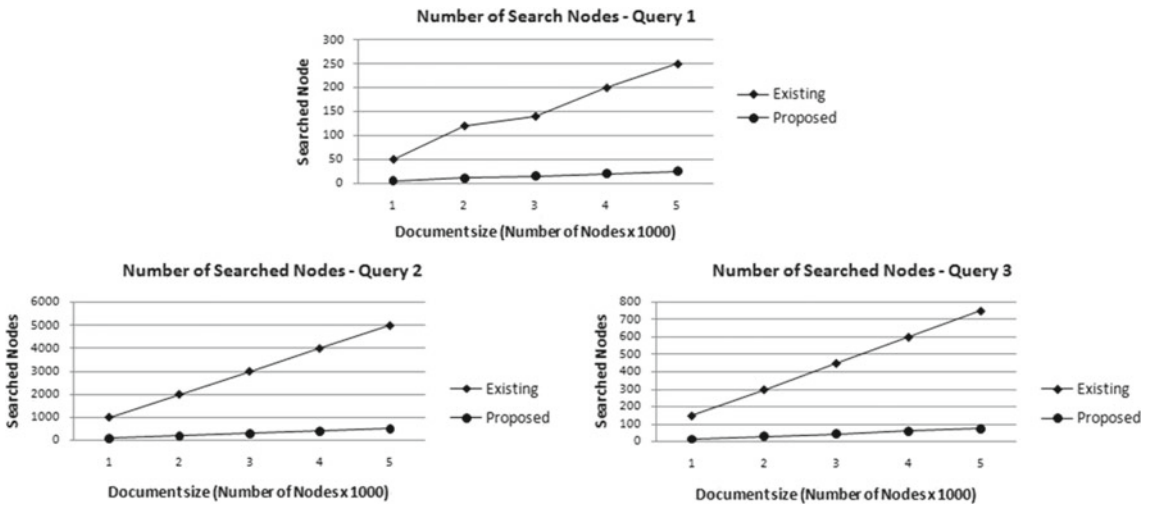

Fig. 10 Number of searched nodes for different queries

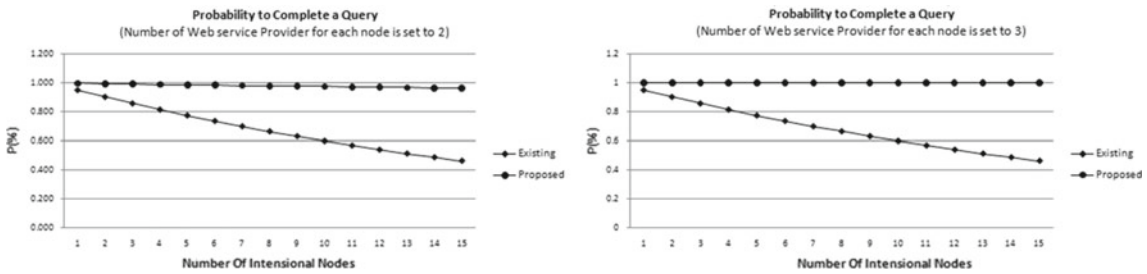

Fig. 11 Probability to complete a query

the number of intensional nodes, as well as the number of Web service providers for each node is increased.

Based on the evaluation, it can be seen that our proposal in relation to AXML representation and AXML query evaluation algorithms can contribute to improving the performance of current AXML systems. However, there is one particular limitation to this work, namely the use of the Dewey coding system into the AXML document, as it can create overheads for AXML systems when AXML documents are frequently updated.

6 Conclusion and future work

Data and service integration is imperative in the rapid development of information technology today. AXML, which is able to combine and exploit the potential power of XML, Web service and P2P architectures, has become one of candidates in integrating and managing XML data and intensional XML data. This paper introduces the essential foundations of AXML, which is the formal representation and algorithms for querying AXML data. It specifies the shortcomings in AXML representation and querying the AXML data of current AXML systems. Then, it proposes improvements for AXML representation as well as evaluating XML queries against AXML data. The proposed representation and algorithms will be applied in the management of intensional data, AXML query evaluation, and intensional data materialization to improve the performance of AXML systems.

There are many issues that need to be studied in relation to current AXML systems. These include the ability to transform arbitrary XML data into AXML data, effective recursive invocations, and processing strategies for Web services both in synchronous and asynchronous communication. In addition, it is essential to propose a benchmark criterion that can be used to assess and compare AXML systems as well as AXML systems with other XML systems.

Although AXML systems are still in their infancy, with remarkable abilities to integrate data and services, AXML can be considered a powerful XML extension to satisfy the need for Web technologies and distributed data management (Abiteboul et al. 2003, 2009) in the future.

References

Abiteboul, S., Alexe, B., Benjelloun, O., Cautis, B., Fundulaki, I., Milo, T., et al. (2004a). An electronic patient record "on steroids": Distributed, peer-to-peer, secure and privacy-conscious. In *Proceedings of the thirtieth international conference on very large data bases, VLDB '04* (Vol. 30, pp. 1273–1276). VLDB Endowment.

Abiteboul, S., Baumgarten, J., Bonifati, A., Cobéna, G., Cremarenco, C., Dragan, F., et al. (2003). Managing distributed workspaces with active xml. In *VLDB '2003: Proceedings Of The 29th international conference on very large databases* (pp. 1061–1064). VLDB Endowment.

Abiteboul, S., Benjelloun, O., Cautis, B., Manolescu, I., Milo, T., & Preda, N. (2004b). Lazy query evaluation for active xml. In *SIGMOD '04: Proceedings Of The 2004 ACM SIGMOD international conference on management of data* (pp. 227–238). New York, NY, USA: ACM.

Abiteboul, S., Benjelloun, O., Manolescu, I., Milo, T., & Weber, R. (2002). Active xml: Peer-to-peer data and web services integration. In *Proceedings Of VLDB* (pp. 1087–1090). Morgan Kaufmann.

Abiteboul, S., Benjelloun, O., Manolescu, I., Milo, T., & Weber, R. (2004c). Active xml: A data-centric perspective on web services. In *Web dynamics* (pp. 275–300).

Abiteboul, S., Benjelloun, O., & Milo, T. (2004d). Positive active xml. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems* (pp. 35–45). New York, NY, USA: ACM.

Abiteboul, S., Benjelloun, O., & Milo, T. (2008). The active xml project: An overview. *The VLDB Journal, 17*(5), 1019–1040.

Abiteboul, S., Gottlob, G., & Manna, M. (2009). Distributed xml design. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, PODS '09* (pp. 247–258). New York, NY, USA: ACM.

Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004). *Web service concepts, architectures and applications.* Springer.

Bradley, N. (1998). *The XML companion* (1st ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2008). *Extensible markup language (xml) 1.0*

(5th ed.). http://www.w3.org/TR/2008/REC-xml-20081126/. Accessed 11 July 2012.
Brinkmann, A., & Effert, S. (2008). Data replication in p2p environments. In *Proceedings of the twentieth annual symposium on parallelism in algorithms and architectures, SPAA '08* (pp. 191–193). New York, NY, USA: ACM.
Eda, T., Onizuka, M., & Yamamuro, M. (2005). Processing xpath queries with xml summaries. In *Proceedings of the 14th ACM international conference on information and knowledge management, CIKM '05* (pp. 223–224). New York, NY, USA: ACM.
Ferraz, C. A., Braganholo, V. P., & Mattoso, M. (2007). Storing axml documents with araxa. In *SBBD* (pp. 255–269).
GEMO (2007). Active xml. http://webdam.inria.fr/axml/index.axml.html. Accessed 11 July 2012.
Goldman, R., & Widom, J. (1997). Dataguides: Enabling query formulation and optimization in semistructured databases. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, & M. A. Jeusfeld (Eds.), *VLDB'97, proceedings of 23rd international conference on very large databases, August 25–29, 1997, Athens, Greece* (pp. 436–445). Morgan Kaufmann.
Harold, E. R., & Means, W. S. (2002). *XML in a nutshell*. Sebastopol, CA, USA: O'Reilly & Associates, Inc.
Hoque, R. (2000). *XML for real programmers* (1st ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
Kemme, B., Peris, R. J., & Patio-Martnez, M. (2010). *Database replication* (1st ed.). Morgan and Claypool Publishers.
Milo, T., Abiteboul, S., Amann, B., Benjelloun, O., & Ngoc, F. D. (2003). Exchanging intensional xml data. In *SIGMOD '03: Proceedings Of The 2003 ACM SIGMOD international conference on management of data* (pp. 289–300). New York, NY, USA: ACM.
Pras, A., Schönwälder, J., & Stiller, B. (2007). Peer-to-peer technologies in network and service management. *Journal of Network and Systems Management, 15*, 285–288.
Ruberg, G., & Mattoso, M. (2008). Xcraft: Boosting the performance of active xml materialization. In *EDBT '08: proceedings of the 11th international conference on extending database technology* (pp. 299–310). New York, NY, USA: ACM.
The Active XML Team (2005). Active xml user's guide. Technical report, AXML Group. http://www.activexml.net/reports/docs/AXML%20Guide.pdf. Accessed 11 July 2012.
Vidal, V., Lemos, F., & Porto, F. (2008). Towards automatic generation of axml web services for dynamic data integration. In *DataX '08: Proceedings of the 2008 EDBT workshop on database technologies for handling XML information on the web* (pp. 43–50). New York, NY, USA: ACM.
W3C (2004). Web services architecture. http://www.w3.org/TR/ws-arch/. Accessed 11 July 2012.

Binh Viet Phan is a PhD Candidate in Computer Science in the Department of Computer Science and Computer Engineering at La Trobe University, Australia. From the same institution, he received his Master of Education in 2008. His research interest is XML Databases and Web Services.

Eric Pardede is a lecturer in the Department of Computer Science and Computer Engineering at La Trobe University in Melbourne, Australia. His teaching and research fields are Information Systems, Databases and Software Engineering. He has published more than 50 research papers in international journals, proceedings and books.

Wenny Rahayu is currently an Associate Professor and the Head of Data Engineering and Knowledge Management research at the Department of Computer Science and Computer Engineering La Trobe University. The main focus of her research is the integration and consolidation of heterogeneous data and systems to support a collaborative environment within a highly data-rich environment. She has been leading a number of large projects in the above area. In the last 10 years, she has published two authored books, three edited books and more than 100 research papers in international journals and proceedings in the area of XML-based data integration, database design and processing, as well as semantic web and ontology.