

# Investigation of improvements in distribution, replication and security of Dynamic XML Documents

Prakash Gupta<sup>1</sup>

Department of Computer Science  
University of Southern California

**Abstract:** The power of distribution and replication has been harnessed since long to provide potential advantages of improved performance, reduced costs, improved reliability and shorter response times. Recently, these powers have been introduced for dynamic XML documents as new features. It is supported by Query languages like Xpath and Xquery and large network of systems provided by the Internet. The autonomy of peers to distribute data over large-scale system requires a proper framework that can guarantee that the document structure is not violated. The goal of this paper is to propose a proper unit of fragmentation for the AXML documents, so that the document remains valid while it is distributed and replicated. To address the communication cost and performance related issues, we propose an optimized query processing strategy while data is distributed and replicated over different peers using semi-join operation. A more secured and reliable security model in distributed environment has also been proposed.

**Keywords:** AXML, Distribution, Replication, Caching, Revocation, Kernel, Web Services, Query Processing, Customized Security, Distributed Databases

## 1. Introduction

XML, a self-describing semi-structured data model, wins the support of research community because of its feature of in-built structured-ness and universal format for data exchange over web. It was originally designed to meet the challenges of large-scale electronic publishing by isolating content from formatting. Separating the content of a document from how it is to be formatted simplifies development and maintenance. Different people with different expertise can work independently on the information captured in a document and on the format.

While XML provides flexibility for handling data heterogeneity, the emerging standards for web services like Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL) simplifies the interoperability problem by normalizing the way programs can be invoked over the web. The popularity of file sharing systems (such as Napster and Gnutella) has attracted research in peer-to-peer (p2p) architectures as an efficient means of sharing data. It provides a decentralized infrastructure in sync with the spirit of web and thus promising solution to the problem of independence and autonomy of sources. To provide the glue between the two paradigms of p2p approach and web services standards, [9] proposed Active XML (AXML) to form a proper ground for data integration over web. They defined AXML as “*a declarative framework that harness XML and web services for data integration that can be put in work in a peer-to-peer architecture*”.

---

<sup>1</sup> Author is a graduate student at USC (Department of Computer Science) Email: Prakash.Gupta1@gmail.com

Motivated by the spirit of including code in HTML documents (e.g. SUN JSP, PHP), [5] incorporated the idea by embedding active fragments into XML documents. The dynamic part of the AXML documents is generated through calls to web services. Some parts of the document can be pointed at completely different peer through external links. The use of service calls and external links along with properties of XML makes it useful to implement database applications over the web. AXML is very similar to XML in the syntax. The syntax information on external links and web service calls for AXML documents can be found in [5, 9]. The syntax information on Xpath and Xquery can be found in [10, 11].

The external links are denoted by the tag name <externalURL> and web service calls by <fun>. The use of external links enhances the power of distribution and replication. In the AXML documents based database system all the peers are independent for distributing and replicating the AXML documents according to their own needs. No centralized scheme is available which puts a constraint on the level of distribution of these documents. Thus a mechanism is required which along with ensuring the structure provided by the DTD (Document type definition) also provides a formal definition for the valid part of distribution of the AXML document. These are crucial consideration in designing the distributed databases as the user has only centralized view of the database or information of some distribution, limited to the peer.

A distributed database comprises of a set of databases (or fragments of databases) stored at multiple sites, which cooperate with each other to provide user a view of a single database. The existing query processing strategies for distributed databases are posed with challenges of minimizing the total volume of data transmitted. A major problem for dynamic XML documents is the absence of global schemas for AXML based database, making global query processing strategies impossible. The join operations will be performed over the database application and that will add to the increased costs of query processing. We try to optimize the storage of database fragments of AXML by extending the concept of semi-join operation used in conventional database systems and using bit array method [8] that reduces the cost of performing join operation between the documents.

Today the insecurity has stemmed into the exploitation at protocols and application level, as the scale of implementation of the applications has increased. In the scenario of dynamic XML based documents, there cannot be a centralized scheme that will govern the implementation of the protocols on all the system. There are two reasons for this; first the level of access provided to each system over the Internet will be different as security levels on some systems will be very critical while on others the level of security required may not be so critical, so each system may not allow equal level of access to the application implementing security protocol; secondly it may not be viable to centrally store the information of all the participating systems over the Internet. Thus each system is equally responsible to implement its own security policy so that the level of security of none of the systems is compromised. Therefore a security model similar to one proposed for legion system [6] is proposed which is more viable than the model proposed by author of [9].

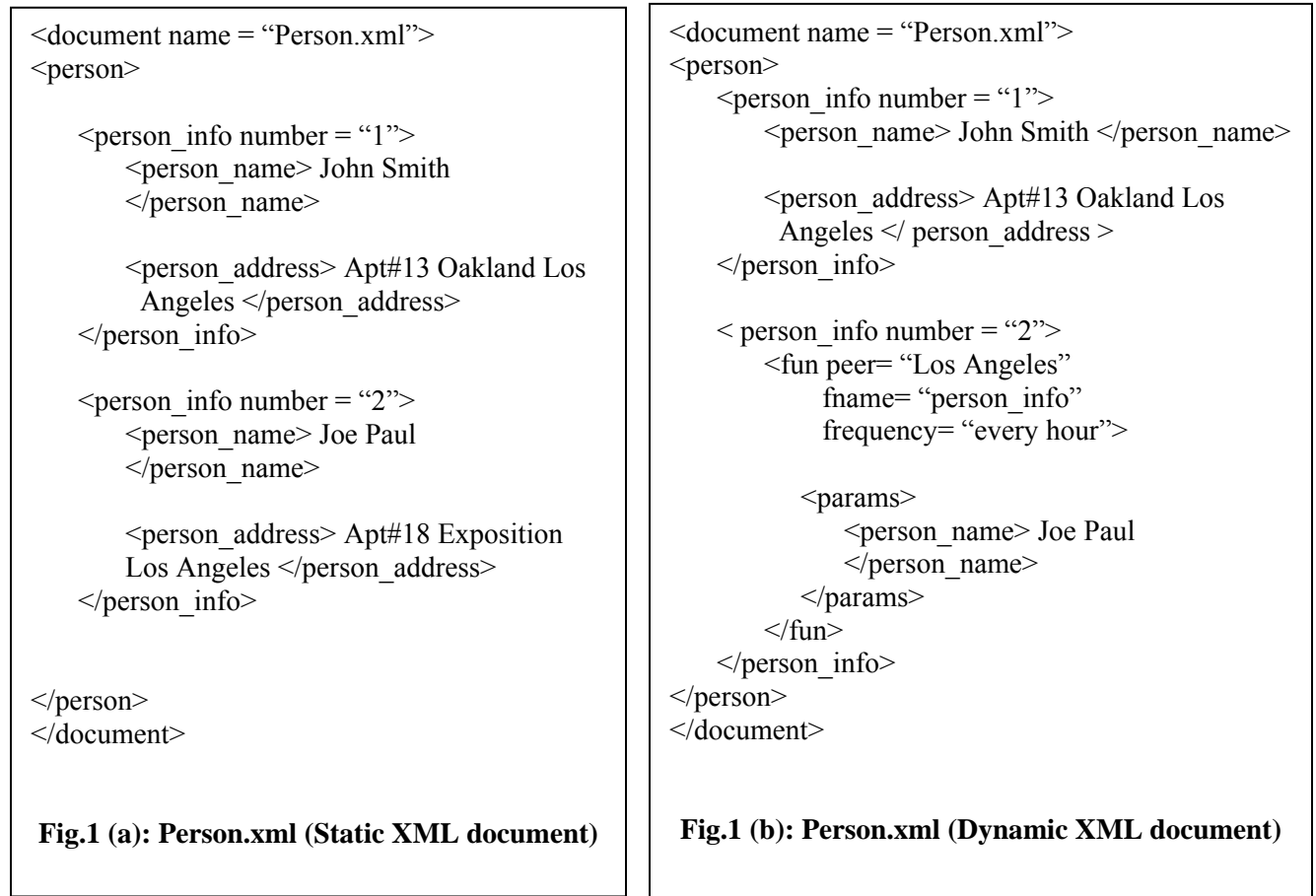
The roadmap for the rest of the paper is as follows. In section 2 we study the dynamic XML documents and propose a valid unit of fragmentation so that while the documents are distributed and replicated over different peers its structure remains valid and well-formed. Section 3 emphasizes on proposing an optimized query processing strategy between different peers over web

to address the high transmission cost incurred over web. In section 4, we study the architecture of dynamic XML documents and address the security related issues by proposing an object-oriented security model. Due to recent increase in incorporating the XML as standard format for data exchange over web, the domain has huge potential for future work stated in section 5, where we conclude.

## 2. Active XML and Fragmentation

A complete model and query processing strategy for the Active XML system has been proposed in [9]. Authors used the term *dynamic XML document* for documents where the parts of the content is materialized XML data present in the document, whereas other parts are generated by calls to programs, when the content of full document is needed.

Here we are concerned with the dynamic parts of XML documents that are provided by the calls to web services. An AXML document (static and dynamic) looks as in Fig. 1(a) and 1(b).



The document Person.xml (*Static XML document*) represents an xml document with information of each person. The tag <person\_name> gives the person name & <person\_address> gives the address of the same person. While document Person.xml (*Dynamic XML document*) represents the corresponding dynamic XML document fro the same. The document Person.xml uses the <fun> tag to call a service "person\_info" which provides information of a person from peer "Los

Angeles”. The tag `<params>` denotes the parameter sent along with the service call `<fun>`. Here the parameter is `<person_name>`. The tag `<frequency>` denotes the frequency of updating the tag information by calling the web service. Here there is a need to define what a web service is. A *web service is any service that is available over the Internet, uses a standardized XML messaging system, and is not tied to any one operating system or programming language*. The web service uses web standards like SOAP and WSDL to provide the execution of service.

In [5], a new tag `<externalURL>` is introduced in AXML to enhance the power of distribution and enable replication in AXML. Through `<externalURL>` a fragment of the document can be pointed at some different peer where that fragment exists. When a query is made on the document the user does not have the knowledge of this distribution. The document is regarded as any normal AXML document. Fig. 2 shows the use of tag `<externalURL>`.

```

<document name = "Person.xml">
<person>

  <person_info number = "1">
    <person_name> John Smith
    </person_name>

    <person_address> Apt#13 Oakland Los
      Angeles </person_address>
  </person_info>

  <person_info number = "2">
    <externalURL>
      www.LA.com/Person.xml/person_info/
    </externalURL>
  </person_info>

</person>
</document>

```

**Fig.2: Person.xml (Dynamic XML document)**

The proper functioning of both the tags discussed above totally depends upon the level of authority provided by the peer “Los Angeles” to the current peer from where these tags are invoked. A security model is provided later, which addresses this issue. As in the standard XML data model, an AXML document is viewed as a labeled tree. The tree nodes represent the AXML elements/attributes and the edges represent the component-of relationship among document elements.

The use of tag `<externalURL>` moves a part of the AXML document to some other peer or some other location on the same peer for which the reference is provided at the `<externalURL>` tag. The document achieved through the distribution by using `<externalURL>` should be a valid AXML

document. The question that arises is what is a valid AXML fragment? First we will see what fragmentation in terms of AXML is. It is the decomposition of AXML document such that the decomposable units are valid AXML documents. We now provide a formal definition for valid notation of AXML document by extending such a definition proposed for XML in [2].

**Notation for AXML documents:**

Let *doc* be an AXML document, then

1. *elements (doc)* denote the set of tag elements in *doc* where tag elements having the same tag name are identified by unique sub indices.
2. *tree (doc)* is the tree structure of the elements defined by *doc*.
3. *root (doc)* is the root element of *doc* and therefore of *tree(doc)*, too.

Let *e elements (doc)* be an element of *doc*, then

1. *doc (e)* denotes the subdocument of *doc* having *e* as its root element.

2.  $tag(e)$  is the tag name of  $e$ .
3.  $value(e, attr)$  is the value of the attribute  $attr$  of  $e$ .
4.  $axml(e)$  denotes the AXML representation or serialization of  $e$  in  $doc$  including the opening and closing tags of  $e$  and all of its contents. In particular,  $axml(root(doc))$  is the AXML representation of the entire document  $doc$ .
5.  $children(e) \subseteq elements(doc)$  denote the set of elements directly contained in  $e$  excluding  $e$  itself. Conversely  $parent(e) \in elements(doc)$  denote the element which contains  $e$ . Obviously parent is not defined for the root element.

**Definition of fragment in AXML:**

If all the above conditions hold then  $f \subseteq elements(doc)$  are a valid AXML *fragment* of  $doc$  iff the sub graph of  $tree(doc)$ , which is induced by  $f$ , is connected. This sub graph forms a tree denoted by  $tree(f)$  and having the root element  $root(f)$ .

The process of distribution and replication can be carried out with the tags `<externalURL>` and `<fun>`. In the case of replication it is the choice of the calling peer, asking for query to choose one of the sites where the fragments are replicated. This choice can be affected by number of factors. These factors are discussed in [5]. The use of external links in an AXML document requires that the fragmentation be performed such that the DTD (Document Type Definition) of the distributed document is not violated. Here we will see how the DTD for AXML documents would look like in Fig.3 for the previous examples.

```
<?xml version="1.0"?>

<!DOCTYPE Person [
  <!ELEMENT person(person_info, externalURL)>
  <!ELEMENT person_info(person_name,
                        Person_address,
                        externalURL)>
  <!ELEMENT person_name (#PCDATA,
                        externalURL)>
  <!ELEMENT person_address (#PCDATA,
                        externalURL)>
]>
```

**Fig3 (a) DTD for document with <externalURL>tag**

```
<?xml version="1.0"?>

<!DOCTYPE Person [
  <!ELEMENT person(person_info, fun)>
  <!ELEMENT person_info(person_name,
                        person_address,
                        fun)>
  <!ELEMENT person_name(#PCDATA, fun)>
  <!ELEMENT person_address(#PCDATA, fun)>
  <!ELEMENT fun(params)>
  <!ELEMENT params(person_name)>
]>
```

**Fig.3 (b) DTD for document with <fun> tag**

In Fig.3 (a) and Fig.3 (b), we provide a possible DTD structure for the documents in Fig.2 & Fig. 1(b) with inclusion of `<externalURL>` and `<fun>` tags respectively. There are more possible forms of DTD for the same documents. Whenever a new AXML document is added in the database at a peer, corresponding DTD is made as per the peer’s requirement.

The choice of including the `<fun>` and `<externalURL>` tag in the DTD depends on the site which develops the document. This choice can be driven by number of factors like query processing, security etc. When the fragment of an AXML document is created for distribution, the DTD at the original peer does not change. Now the question arises that how do we manage the new fragment created and maintain its integrity? This can be managed in two ways, either a new DTD for the

fragmented document is created in accordance to the available DTD, which will be available to the new peer or the original DTD is used, such that there are empty tags in the newly created fragment where no information is available. Use of DTD as per the originating peer's requirement ensures prevention of further replication of data by other peer if originating peer doesn't wish to do so. All the sites to which the document fragment is distributed should follow the prescribed DTD else the document will be deemed invalid. By extending the definition of a valid fragment of XML defined in [2], we have tried to address the issue for AXML documents because AXML has almost similar properties as that of XML.

### 3. An optimized Querying Strategies: Distribution and Replication

This section emphasizes on proposing an optimized query processing strategy between different peers over web using semi-join operation so that transmission costs incurred over web can be minimized. So before we propose an optimized scheme, we need to understand the storage and retrieval of XML documents in different commercially used distributed databases.

#### Storage of structured documents

There are two major approaches to the storage and retrieval of XML documents in databases. XML documents are regarded as *structured data* in one approach, and as *simple character string* in another approach.

When structured documents are regarded as *structured data*, the tree structure representing an XML document is mapped to database schemas. In this approach, database schemas are designed in accordance with the DTD of structured documents. Once such database schema is designed, XML documents stored in databases are guaranteed to conform to the DTD. When structured documents are regarded simply as *character strings*, an XML document is stored in a database attribute. Operations on tree structure are replaced by string operators, and abstract data types which have functions to execute string operators are added to databases. Under this approach, queries on structured documents are described in extended SQL.

#### 1. Using object-relational databases.

The object-relational approach proposed by [12] stores XML documents independent of DTDs or element types. For retrieving XML documents, the system rewrites declarative queries for XML into SQL queries which are executed in object-relational databases. In [12], since database schemas were independent of DTDs or element types, changes in logical structure do not influence on database schemas.

#### A Tree Structure Representing an XML Document

An XML document can be represented as a tree, and node types in the tree are of the following three kinds: Element, Attribute and Text.

- Nodes of type **Element** have an element type name as a label. Element nodes have zero or more children.
- Nodes of type **Attribute** have an attribute name and an attribute value as a label. Attribute nodes have no child node.

– Nodes of type **Text** have character data specified in the XML Recommendation as a label. Text nodes have no child node. Fig. 4(b) shows the tree structure representing the XML document in Fig. 4(a).

```

<books>
  <book style="textbook">
    <title>Designing XML applications</title>

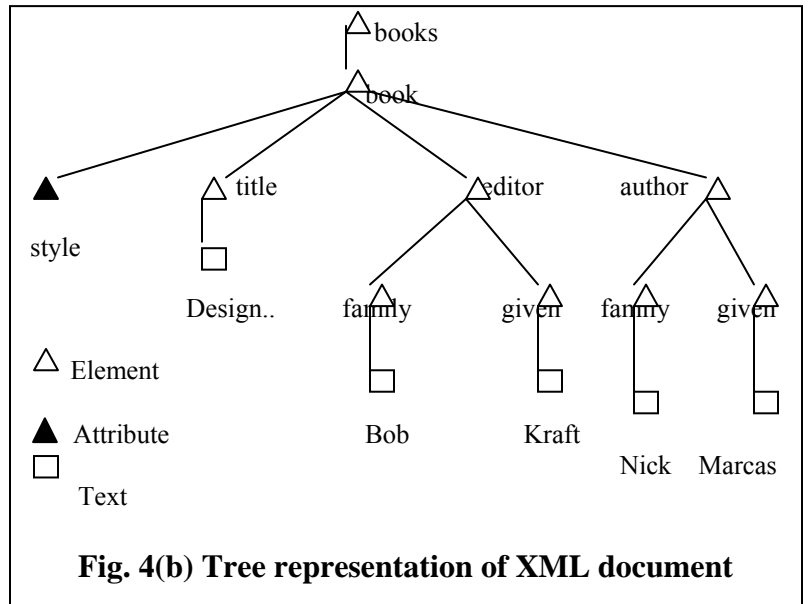
    <editor>
      <family>Bob</family>
      <given>Kraft</given>
    </editor>

    <author>
      <family>Nick</family>
      <given>Marcus</given>
    </author>

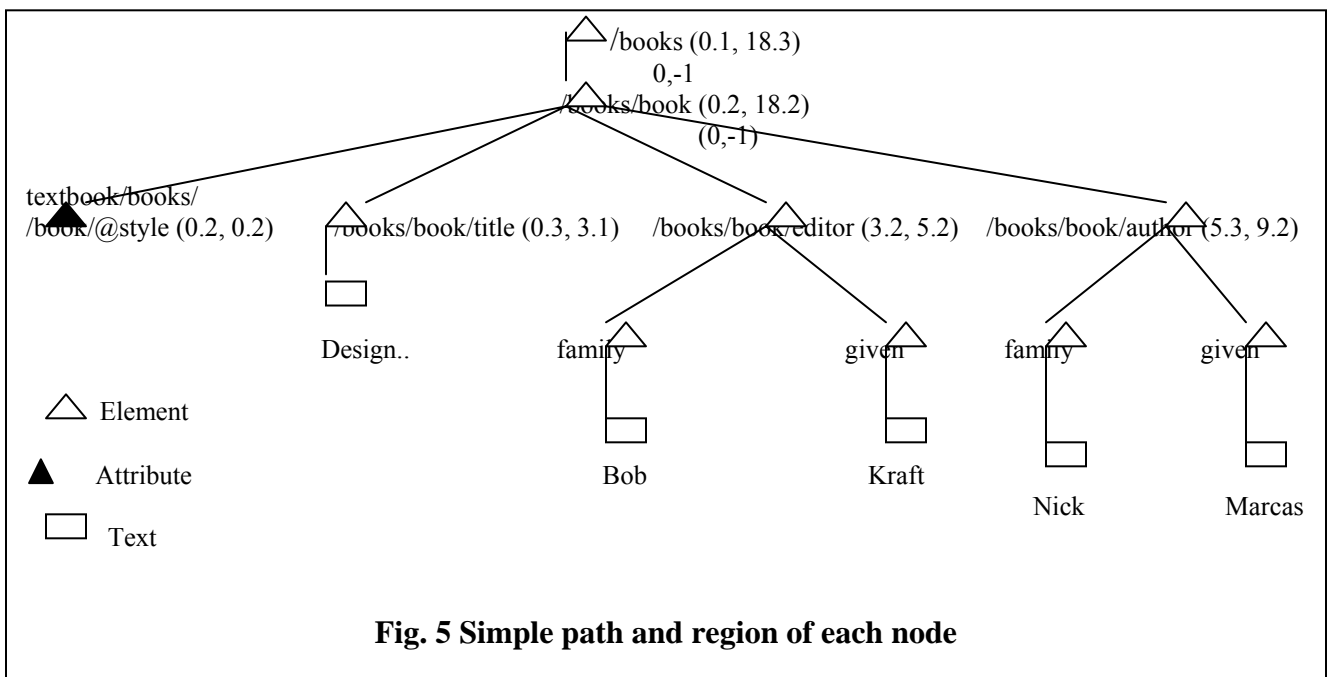
  </book>
</books>

```

**Fig. 4(a) Sample XML document**



The decomposition of tree structure into relations supported easy access and reuse by the unit of logical structure and can use index structures (e.g. *B+* trees, *R* trees, etc.) provided in database systems. It decomposed XML documents into *simple paths*, and stored them in databases. However, using only simple paths, retrieval allowing for the hierarchy or order within a document could not be handled. Therefore, [12] retained for each node, *simple path* and a *pair of positions* of the node within the document. Such pair is usually called a *region* (i.e. a pair of a start position and an end position). Because of this mechanism, functionalities of XML query language can be supported properly. Also, the inclusion relationship and the order relationship between nodes can be maintained. As an example, Fig.5 shows simple paths, regions, and occurrence order information on node type Element, which are derived from the tree structure in Fig. 4(b).



Relations for storing XML documents are four kinds: Element, Attribute, Text, and Path. The relations Element, Attribute and Text store data about each node type. The relation Path stores data about simple paths. Each relation has the following database attributes.

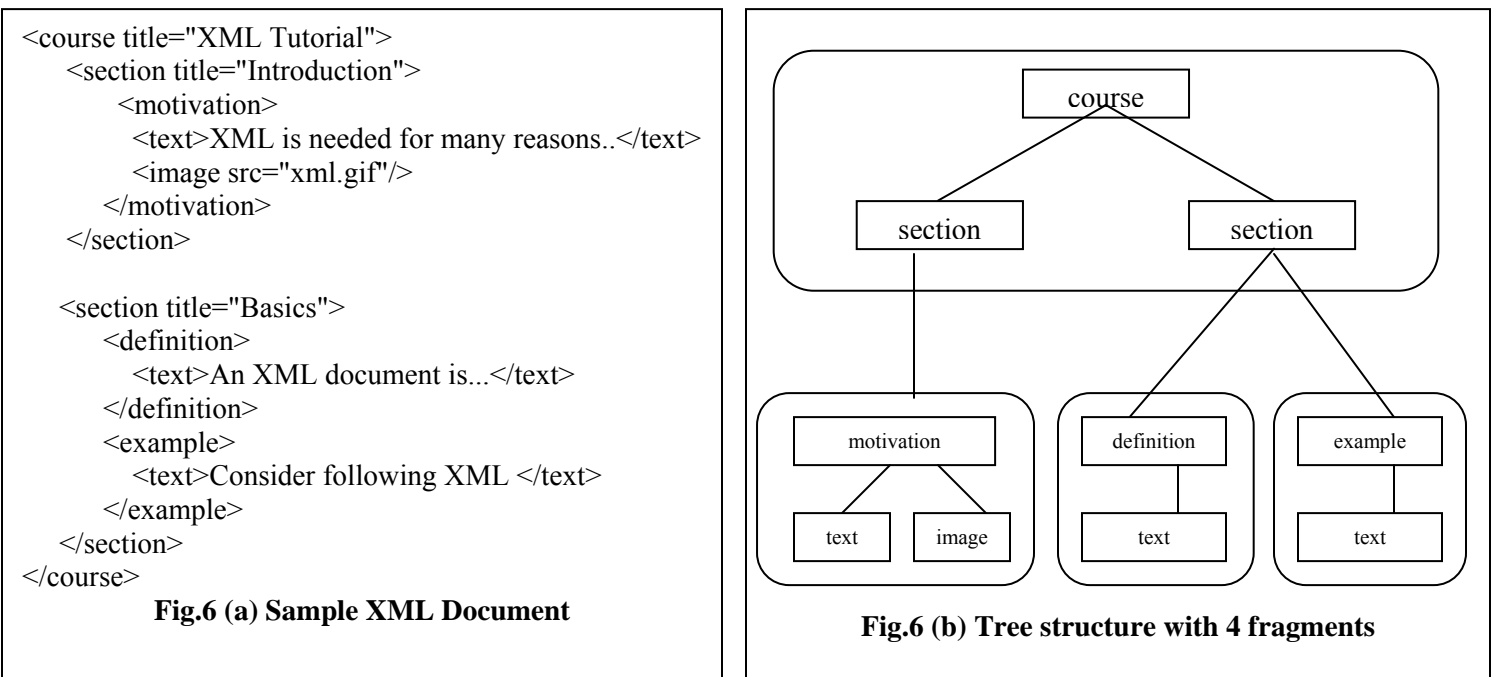
Storage of tree structures in the relational tables in Fig.5, the source XML documents can be rebuilt because of preserving document identifier and region of each node type. By dividing tree structures into nodes and storing them according to the node types, it provided following advantages.

- Database schemas to store XML documents do not depend on DTDs or element types. Any XML documents can be managed, being based on the four relational tables.
- Index structures provided in database management systems can be used.

## 2. A model-based fragmentation of XML documents

While the above mentioned strategy used object relational database systems to store XML data. We will now discuss a model-based fragmentation and storage of XML documents proposed by [2]. It adopts a flexible fragmentation of XML documents that avoids at least unnecessary joins when reconstructing frequently accessed parts of the document. The storage model is based on directed acyclic graphs that facilitates the reuse of XML subdocuments and supports different views on XML documents.

The *fragmentation strategy* that they used has already been described in the previous section [2]. We will now illustrate it better with the help of example. Fig. 6(b) shows the tree structure of XML document of Fig. 6(a) using four fragments with root elements course, motivation, definition and example identified by the unique identifiers 1 through 4.



Tree structure of an XML document Fig.6 (b) represents the fragmentation of XML document of Fig.6 (a).

### 3. A structure independent mapping approach.

[4] presented a structure independent mapping approach. With this implementation, users can store XML documents regardless of structure of documents. The application gives users two alternatives (XRel and Edge) for database schema design.

They employed the data model of XPath [11] to represent XML documents. In the XPath data model, XML documents are modeled as an ordered and directed labeled tree. There are seven types of nodes. They considered only the following four types of nodes for the sake of simplicity: *root*, *element*, *text* and *attribute*. The root node is a virtual node pointing to the root element of an XML document. Elements in an XML document are represented as an element node. Element nodes can have other elements or text as its children. Text nodes are string-valued leaf-nodes. An element node can have a set of attribute nodes. An attribute node has an attribute-name and an attribute-value.

```

<address_book>
  <card no="1">

    <name>
      <surname>Brown</surname>
      <given>Paul</given>
    </name>

    <address>
      <street>20th Floor, 300
        Lakeside</street>
      <city>OAKLAND</city>
      <state>CA</state>
    </address>

  </card>
</address_book>

```

**Fig. 7(a) Sample XML document**

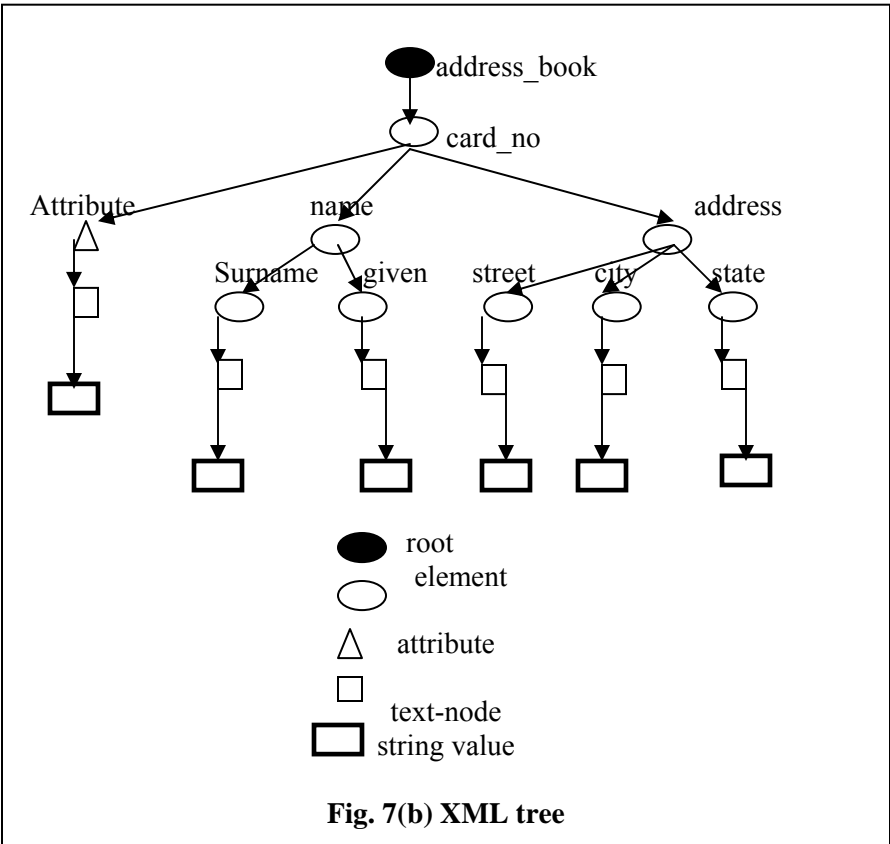


Fig. 7(b) shows an XML tree such that a left-to-right and depth-first traversal describes the order of the XML content within our document in Fig. 7(a). The Edge method proposed by [4] stores all edges of an XML tree in a single table named Edge. The table schema is as:

*Edge (Source, Ordinal, Target, Name, Flag, Value)*

An edge is specified by two node identifiers, namely Source and Target. The Name attribute stores the element or attribute name. The Ordinal attribute records the ordinal of the node among its siblings. A flag value indicates whether the target node is an inter-object reference (ref) or points to a value (val). This approach is quite simple. It stores *parent-child relationship*. A large number of joins may be required during retrieval of the original document from the records stored in the table if the tree is not shallow.

**Comparison of above mentioned approaches:**

	<b>Object-relational databases</b>	<b>A model-based fragmentation</b>	<b>Structure independent mapping</b>
<b>Query processing</b>	High performance (indexing and clustering)	Fast	Low (requires large number of joins)
<b>Granularity</b>	Fixed	Variable granularity	Fixed
<b>Flexibility</b>	Low (requires user intervention)	Highly flexible (Supports creation of multiple fragments)	Very Low (Cannot support dynamic XML documents)
<b>Complexity</b>	Complex (requires XQL intervention)	Complex (in creating algorithms to create fragments)	Low (supports child-parent relationship)
<b>Extensibility</b>	Very High (independent of DTD structure)	High (supports different views, independent of XML structure)	Very High (independent of DTD structure)
<b>Heterogeneity</b>	Can only support relational databases	Can be ported to any databases (relational, network)	Can only support relational databases

**Fig. 8 Comparison between different approaches**

In Fig. 8 we have compared different approaches from query, processing, granularity, flexibility, complexity, extensibility and heterogeneity point of view. A complete approach on various query-processing strategies for distributed database for relational database system can be found in [2, 4 and 12]. Since selecting a proper query processing strategy is always a challenge in distributed databases to minimize the transmission cost, we would now stress upon minimizing it while we distribute and replicate the AXML documents.

Now, lets us see how query processing takes place in AXML based database system and how the join operation works. A query Q is invoked at some peer P1. The processing of Q may involve some remote AXML document for the join operation with some local AXML document. The join operation can be on two complete AXML documents or on some part of it. This query processing comprises of decomposition of Q into sub queries (evaluated at the other peer’s local document)

that is distributed and processed at each peer. The results are computed at each peer and sent back to the originating node P1 using SOAP messaging. Thus convergence of query result takes place at P1. Thus in AXML based system each peer is *independent* to decide on the query processing strategy based on the way it sees the distribution and the past experiences with remote sites (trust list etc.). Thus every intermediate site has the knowledge about the query-originating site. A cost model for such a query processing strategy is discussed in [9].

For simplicity of explanation we consider simple AXML documents without dynamic parts. We also ignore the attribute values, but in actual implementation attribute values are required. Let us consider the document *Person.xml* and *Accounts.xml* in Fig. 9(a) and 9(b).

```

<person>
  <person_info>
    <person_name> John Smith
    </person_name>
    <person_address> Apt#13 Oakland Los
      Angeles </person_address>
  </person_info>
  <person_info>
    <person_name> Joe Paul
    </person_name>
    <person_address> Apt#18 Exposition Los
      Angeles </person_address>
  </person_info>
</person>

```

**Fig. 9(a) Person.xml**

```

<accounts>
  <account>
    <account_no> 0065010588 </account_no>
    <person_name> John Smith
    </person_name>
    <balance> 5000 </balance>
  </account>
  <account>
    <account_no> 0065010599 </account_no>
    <person_name> Joe Paul
    </person_name>
    <balance> 15000 </balance>
  </account>
  <account>
    <account_no> 006501111 </account_no>
    <person_name> Charles
    </person_name>
    <balance> 25000 </balance>
  </account>
  <account>
    <account_no> 006501222 </account_no>
    <person_name> Peter
    </person_name>
    <balance> 8000 </balance>
  </account>
</accounts>

```

**Fig. 9(b) Accounts.xml**

Due to large communication costs on the Internet the join operation can be very costly. We propose to use semi-join in place of simple join operation to reduce the amount of data communicated. We consider two situations where join operation is inevitable

1. User query performs join
2. Distribution of document requires the simple path expressions to be converted to join operation.

These two documents are initially located at the same site called P1. We write an Xquery that requires join operation to be performed as follows. The query as in Fig.10 is executed through simple join operation of the documents *Person.xml* and *Accounts.xml*. Suppose due to some factors (like storage, performance etc.) the document *Accounts.xml* is distributed to some other peer P2. The document *Accounts.xml* at its original site will look like document in Fig.11

```

for $x in document ("person.xml") //person_info
  for $y in document("accounts.xml")
  account[person_name=$x/ person_name]
return
<person>
  <person_name>
    {$x/person_name}
  </person_name>
  <person_address>
    {$x/person_address}
  </person_address>
  <balance>
    {$y/balance}
  </balance>
</person>

```

**Fig.10 XQuery for Join**

```

<accounts>

  <externalURL>
    http://www.P2.com/accounts
  </externalURL>

</accounts>

```

**Fig.11 Accounts.xml**

As the distribution is not visible to the user of the database, user will write the above query without any modification. Now it is the work of the Query processor to implement the query using the join operation. We will extend the use of relational algebra operations in the context of AXML documents. The main shortcoming of the join approach is that entire participating documents must be transferred between sites. The semi-join acts as a size reducer for a document. Thus the *semi-join* takes place as follows:

**Steps for semi-join operation:**

**Step 1:** Projection on document “person.xml”

```

<persons>
  for $p in document ("person.xml")//person_info
  return

  <person>
    <person_name> {$x/person_name} </person_name>
  </person>
</persons>

```

Result is sent to peer P2 say in the form of “person2.xml”

**Step 2:** Peer P2 computes  $\text{accounts}' = \text{accounts} \bowtie_{\text{person\_name}} \text{person}$  ( $\bowtie$ : semi-join operator)  
 for \$p in document (“person2.xml”) //persons

```

for $a in document ("accounts.xml") //account[person_name = $p/person]
  return

```

```

<account>
  <account_no> {$a/account_no}</account_no>
  <person_name> {$a/person_name} </person_name>
  <balance> { $a/balance } </balance>
</account>

```

**Step 3:** Result is sent to peer P1 say in the form of “accounts2.xml”

**Step 4:** Final Result calculation

```

for $x in document ("person.xml")//persons
  for $a in document ("accounts2.xml")//account[person_name = $x/account]
    return

```

```

<person>
  <person_name> {$x/person_info /person_name} </person_name>
  <person_address> {$x/person_info /person_address} </person_address>
  <balance> { $a/account/balance } </balance>
</person>

```

The overhead cost of the semi-join based approach is the cost in step 1 where we need to take a projection on attribute value and sending it to other peer so that only the tags required for the query is returned. The semi join-based approach is better if semi-join acts as a sufficient reducer that is if a few elements of ‘accounts’ participate in join. The join approach is better if almost all elements of ‘accounts’ participate in join, as there is no reduction in size of returned document. The site performing the join operation has to take the above considerations into account before making a decision on the selection between join and semi-join operation. Maintaining a history of query records can be used to perform this.

In the same way when two or more documents are distributed among various peers and the user performs a join operation then the above approaches are to be taken into consideration. In order to improve the performance of semi-join operation bit array method can be used to encode a document into binary values. Such a method can be found in [8] for relational database systems.

A bit array  $BA[1: n]$  is useful in encoding the join attribute values present in one document. Let us consider semi-join between two dynamic documents R.xml and S.xml. Then  $BA[i]$  is set to 1 if there exists a join attribute value  $A = val$  in document S.xml such that  $h(val) = i$ , where  $h$  is the hash function. Otherwise,  $BA[i]$  is set to 0. Such a bit array is much smaller than a list of join attribute values. Therefore transferring the bit array is much smaller than a list of join attribute values to the site of document R.xml saves communication time. The semi-join can be completed as follows. Each tag of document R.xml, whose join attribute value is  $val$  belongs to semi-join if  $BA[h(val)] = 1$ .

Implementing such a method is possible only within a group of sites that agree on it. If the distribution extends towards other sites, the *last site* within the group who receives the AXML document has to ensure that proper method of execution is followed and the query-originating site receives the correct form of the result. The bit array method will have to be adjusted for text data. Thus in the AXML based database system, query processor takes user query as an input, and then it scans the document involved and checks for all information required for the processing of query.

Here query decomposition takes place i.e. query is rewritten so that it contains only linear path expressions based on what-if analysis in [5], by introducing new variables. Among the resulting linear path queries there may exist dependencies and these dependencies are resolved using external links. The strategy involves running all the operators such as joins, outer-joins and projections at the peer P where query was originally asked. Here our basic approach is the same, except that author of [5] did not consider the optimization of amount of data brought in for the processing i.e. computation of joins at originator. Thus, here our semi-join approach would considerably reduce the redundant data shipping cost incurred.

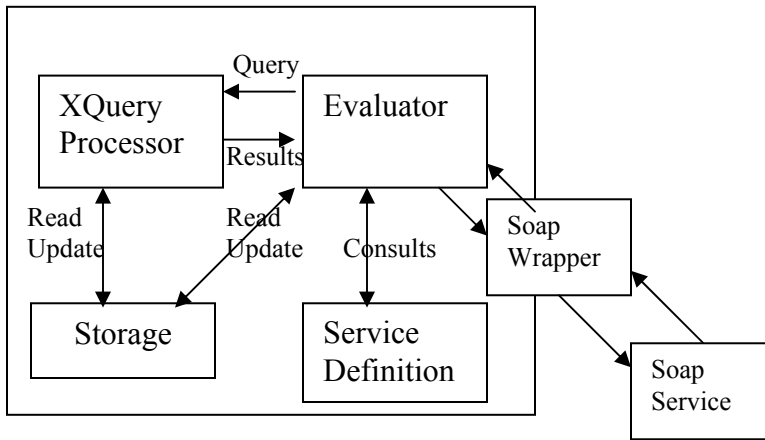
## **4. A more secured scheme for dynamic XML documents**

### **AXML data and service integration architecture**

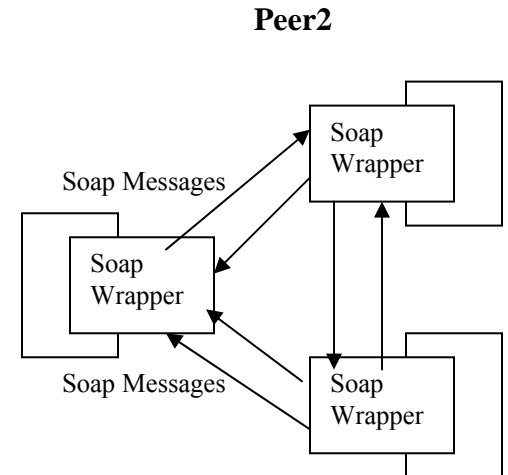
The ActiveXML is proposed in [9] as a declarative framework that harnesses web services for data integration and works in a peer-to-peer architecture. The system revolves around dynamic AXML documents that are similar to XML documents. This is well supported by the increasingly popular web services technology and standards like SOAP and WSDL which simplify the interoperability problem by normalizing the way programs can be invoked over the web. The proposed AXML language format includes linguistic features to control the timing of service call activations, the lifespan of data, and the exchange of extensional and intentional data. The System ensures that the structure of the AXML documents, proposed through a DTD is maintained. An ActiveXML site looks as shown in Fig.12 (a).

An ActiveXML system has an evaluator, which does the evaluation of the user query and performs query decomposition. The query is decomposed into Qlocal and Qnext. The evaluator sends the Qlocal to Xquery processor on the same site and based on the what-if analysis in [5] distributes the Qnext to other related peers. The AXML peer also has storage of locally available AXML documents on which execution of Qlocal is performed. As ActiveXML system communicates through Internet, the level of insecurity is high. Therefore secure SOAP messaging is used. For this sake each peer has a SOAP wrapper that is used for communicating between various peers.

The communication between all the ActiveXML sites takes place as shown in Fig. 12(b). The above architecture works very well on the Internet but suffer drawbacks. Few drawbacks are discussed and solved in [9], but can be further improved. The effect of the existence of external links is not considered in [9], we need to consider this and propose a security model, which can work well on such a wide scale of distribution. We propose an *object oriented security model* that improves the security considerations in [9] and makes each site responsible for its own security. Such a model exists for legion system in [6].



**Fig. 12 (a): Active AXML architecture**



**Fig. 12 (b): Communication**

The conventional security approach of “the system” being the mediator of all interactions between users and resources to enforce a single system-wide policy is not applicable in this scenario. There is no single protected kernel. There are several autonomous peers each having their own administrative domains, which communicate to solve problems of each other and form a *federated system*. Here we involve several machines that are not equally trusted and even some of them might be malicious. Thus there are some sets of design principles that we have to follow in order to enforce a practical security model, on such a large scale.

Some of the problems arising due to dynamic nature of AXML documents have been identified in [9], they are

1. *A malicious client may make the following call to qod.com*  
`<fun> qod.com/QuoteOfDay (<fun>buy.com/BuyCar ("BMW Z3")</fun>)</fun>`  
 Although the client does not buy the BMW car but he makes qod.com buy it.
2. *A malicious site qod.com may send an AXML document as result to our query in the form of intentional call to buy.com*  
`<quote> Love means never having to say you're sorry.  
 <fun>buy.com/BuyCar("BMW Z3")</fun>  
 </quote>`
3. *A site may send its data to possibly malicious site i.am.bad. This data may be private parts of a document*  
`<fun>i.am.bad/SneakAbout ([../../*]) </fun>`

Due to inclusion of external links new security concerns arise which are not addressed in [9]. These problems can be solved through the security model proposed below and it also improves the security measures suggested in [9]. They are as follows:

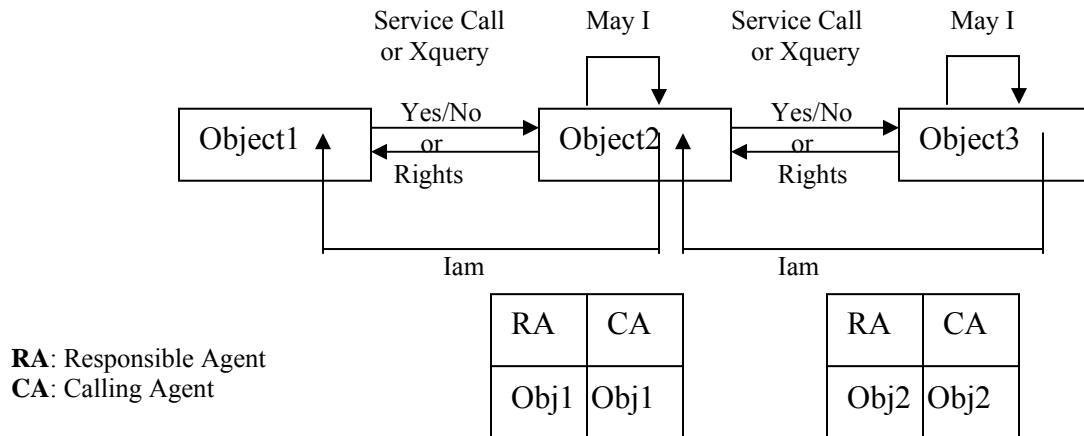
1. We do not have control over the sites which will be involved in the processing of Xquery as the series of external links can make the call move to the sites deemed unreliable by the

originating site. One possible solution to this is to determine the *secure execution path* during the *what-if analysis*, but this will increase the communication cost tremendously.

2. The security requirements of all the AXML documents at the same peer may not be the same. *Cancel* (*sv, S*) in [9] does not address the issue of *customized security* which may be likely for performance consideration as well as for proper security of sensitive information.
3. It may not be feasible to store the list of all the trusted sites with the involvement of large number of sites returning the results to Xquery, rather some punishment mechanism is required so that the intermediate sites can punish the site it trusts which serves unreliable data to the Query originating site.
4. It has not considered the *secure soap messaging*, which might not be possible to implement over all participating sites on the Internet but can be implemented among the sites of the organization describing their own set of secure policies. This method wraps the data provided by various peers without those peers knowing how the wrapping will be performed. This gives added security advantage.
5. The removal of *expired links* has not been addressed in the [9]. These expired links can be a major drawback in query execution if not handled regularly. The expired links will give wrong information on the rights it has on the target site that may result in choosing execution path that no longer exists.
6. The protocol of data transfer on query execution provides knowledge of query originating site. Here the performance benefits are large, still *DOI* (Denial of Information) attacks cannot be ignored. Information may be misused and malicious information may be sent to that site.

## Design Principles of Security Model

The main responsibility of the system is minimizing any possibility of intrusion or mischief on any remote system through proper authentication. Every system is responsible for ensuring its own security as it's up to the user to decide level of security and policies as per its own requirement and discretion. Another responsibility of the system is to ensure safe and secure communication protocol, so that smooth interaction can take place as per the policies and agreement between the sites.



**Fig. 13: Function Calls**

The set of AXML files are the *user-defined objects* with some set of member functions or services. Here the user sends a query, which requires use of information stored in AXML documents. But to access this data, specific rights are required on the object, and this is absolutely for the peer to decide based on extent it trusts the user (user can be local user as well as some other peer through external link). Here the external links in the AXML document basically acts like a member functions that makes SOAP request to corresponding peer.

Every document or group of documents provides certain known services that can be used for authorization, before accessing the documents, such as *MayI()* and *Iam()*. There is a responsible agent (RA) who can be held liable for any operation, which is nothing but generally the peer at which query request has originated. There are some sets of rules defined for the actions at the invocation of each function, shown in Fig.13.

Every time an object calls a service on another object on some other peer, *MayI* is called. *MayI* returns access rights to the particular peer based on the policy enforced by the site in the form of set of bits. These rights can be of particular service or a particular object, or class or a generalized right for the user for subsequent requests. The author of [9] has proposed to send binary value instead of set of bits containing rights. *MayI()* is a relatively costly operation that requires consultation of external databases and extra exchange of authentication messages. Thus it would be quite inefficient to make this call before every function. So instead of returning a simple Boolean the access rights are sent as bits for all the objects.

The bits include the access rights over other services the CA has, that are provided by the called object. Sending rights has more advantages than sending Yes/No binary value as more information is available over the access rights available and improper calls can be avoided. This will reduce the failing and execution time of the Query in case necessary rights are not available. Hence, the service provider provides a license to user to access its resources, which do not require subsequent authentication for that class of objects. The license also consists of information regarding any expired external link of the user using which user can make corresponding changes in the AXML database (Updation).

The function *MayI* and *Iam* have the following form.

**MayI** (Xquery X, S) – Arguments involve an Xquery and sub query S generated by query decomposition.

**Iam** () is another default function, which is called each time a *MayI* () is called. Here the service provider calls back the CA for the authentication purpose. Authentication can be based on public key cryptography.

As it is already stated, it is the peer's responsibility to decide on the level of security. A peer may choose to default *MayI* to null, thus keeping minimum level of security or may choose a higher level by following proper authentication.

## Caching the licenses

The two functions *MayI* and *Iam* ensure the authenticity and trustworthiness for both the interacting peers – the *service provider* (by authenticating request) and *calling peer* (whether the

service provider is really willing to provide the service or not). For this purpose we need to cache the licenses. Licenses may be stored at caller's address space, but there is a risk of caller manipulating with the license by adding rights, etc. although these can be encrypted but the better approach would be storing license at called *object's address space* so that total control is with the service provider and caller can lookup whenever he feels like with no write permission on it. This also opens possibility of easy revocation of licenses, as it is present in local cache.

Also it leads to flexibility in terms of protection level the Service Provider is interested in i.e. if security is not the issue, no caching or authentication is needed. Every peer called maintains its cache with appropriate information about RA, CA and rights and policies available to the caller. There is an *expiry time* for these licenses. After expiry the *MayI & Iam* functions are again called for authentication.

## 5. Conclusion

We have addressed new challenges posed by the introduction of dynamicity in *distribution* and *replication* in XML based documents over web. The entire work is divided into two parts. Firstly, a proper framework for fragmentation was found and an optimized query processing was proposed in a distributed environment. Since in AXML a part of document needs to be distributed, we need to build a mechanism by which validity of document can be checked at different peers. So we proposed the definition of a valid fragment and use of DTD at each peer.

The earlier approaches of *what-if analysis* [5] did not consider the optimization of amount of data brought in for the processing i.e. computation of joins at originator. Thus, here our *semi join* approach would considerably reduce the redundant *data shipping cost* incurred.

Secondly, we introduced a more reliable security model, which effectively addresses the specific security issues arising due to distribution and replication of AXML documents over Internet. We proposed an *object oriented security model* similar to the model proposed in [6] that gives every user, flexibility to decide level of security and policies as per his/her own requirement and discretion. It ensures proper agreement between peers by using the two-way authentication mechanism. It also supports the feature for *caching* and *expiry time* of licenses at service provider's end and thus opens possibility of easy *revocation* of licenses.

Future work can focus upon the implementation of the proposals made and observe its performance in different architectures such as p2p and client-server model. Some other query processing strategies can be developed which may provide improved results in terms of data communication, global processing and storage costs. Effective mechanisms to address DoS and DoI attacks needs to be developed to support the dynamic features of XML more effectively.

## References

- [1] Jayavel Shanmugasundaram et al, "*Relational Databases for Querying XML Documents: Limitations and Opportunities*", Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999

- [2] Christian Süß, “*An approach to the model-based fragmentation and relational storage of XML-documents*”, Universität Passau, D-94030 Passau, Germany, 2000
- [3] Syam Menon, “*Allocating Fragments in Distributed Databases*”, IEEE Transactions on Parallel and Distributed Systems, 2005, Vol.16, No.7
- [4] Zülal Sevkli, Mine Mercan et al “*A Middleware Approach to Storing and Querying XML Documents in Relational Databases*”, ADVIS 2004, LNCS 3261, pp. 223–233, 2004.Springer-Verlag
- [5] S. Abiteboul, T. Milo, A. Bonifati, G. Cobena and I. Manolescu, “*Dynamic XML Documents with Distribution and Replication*”, ACM 2003, pg 1-23
- [6] W.A. Wulf, C. Wang and D. Kienzle, “*A New Model of Security for Distributed Systems*”, ACM 1996, pg 34-39
- [7] Mark Graves, “*Designing XML Database*”, First Edition, USA, 2002, pg .2-23
- [8] M.T. Ozsu and P. Valduriez, “*Principles of Distributed Database Systems*”, Second Edition, 1999, pg. 212-254
- [9] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo and R.Weber, Active XML: “*A Data-Centric Perspective on Web Services*”, ACM 2002, pg 527-538
- [10] Xquery 1.0: An XML Query Language. W3C Working Draft February 2005, <http://www.w3.org/TR/Xquery/>[5]
- [11] A. Moller and Michael I. Schwartzbach, “*The XML Revolution: Technologies for the futureWeb*”, <http://www.brics.dk/~amoeller/XML/index.html>
- [12] Takeyuki Shimura, Masatoshi Yoshikawa and Shunsuke Uemura, “*Storage and Retrieval of XML documents using Object-Relational Databases*”, Proceedings DEXA '99 Florence Italy, 1999