Achieving Adaptivity for OLAP-XML Federations

Dennis Pedersen
TARGIT
dp@targit.com

Torben Bach Pedersen
Aalborg University
tbp@cs.auc.dk

ABSTRACT

Motivated by the need for more flexible OLAP systems, this paper presents work on logical integration of external data in OLAP databases, carried out in cooperation between the Danish OLAP client vendor TARGIT and Aalborg University. Flexibility is ensured by supporting XML as the external data format, since almost all data sources can be efficiently wrapped in XML. Earlier work has resulted in an extension of the TARGIT system, allowing external XML data to be used as dimensions and measures in OLAP databases. This work has led to a number of new ideas for improving the system's ability to adapt to changes in its surroundings.

This paper describes the potential problems that may interrupt the operation of the integration system, in particular those caused by the often autonomous and unreliable nature of external XML data sources, and methods for handling these problems. Specifically, we describe techniques for handling *changes* in external XML data sources. We also describe techniques for improving the reliability of external XML sources, e.g., when these are found on the Internet, by dynamically trying to locate alternative sources during the evaluation of a query. Finally, we discuss solutions to a number of other possible problems, and show how the techniques can be integrated in the TARGIT architecture. Experiments performed with a prototype implementation of central functionality shows the viability of the proposed solutions.

Categories and Subject Descriptors

H.2.4 [**Database Management Systems**]: Distributed Databases

General Terms

Reliability, Performance

Keywords

OLAP, XML, federated databases, adaptivity

1. INTRODUCTION

This paper presents the results of work on logical integration of XML data in OLAP databases. The work has been carried out in cooperation between TARGIT, a Danish OLAP client vendor [11], and Aalborg University, and is based on previous results [7, 9]. TARGIT's practical experience with the OLAP industry and our experience with a prototype extension to TARGIT's client tool [7], has raised a number of new research issues on the adaptivity of OLAP-XML integration systems, which are also interesting in a broader context. These issues are the subject of this paper.

The TARGIT Analysis Suite [11] is an OLAP client system, which is known for its user-friendliness and short roll-out time. This system, as most other OLAP systems, enables the analysis of large amounts of data using a multidimensional approach, where the data is considered as a (multidimensional) *cube* consisting of numerical *measure* values characterized by a set of hierarchical *dimensions*. The multidimensional approach is often easier to comprehend from a user perspective than traditional types of DBMSs [12] and it offers good query performance even for complex queries, mostly due to the use of pre-aggregation [12]. However, these advantages come at a price. In practice, one of the most important tradeoffs is difficulties in handling *changes*, both in the data and in the structure of the database. For example, adding a new dimension is often a very time and resource consuming process.

We have previously presented a *federated database system* that allows data outside the OLAP database to be used as *logical* dimensions or measures in a cube [7, 9]. Since almost any kind of external data can be efficiently transformed to XML [14], including data from relational, object-oriented, and multidimensional DBMSs, XML was chosen as the external data format, thereby ensuring flexibility. In this way, external XML data located, e.g., on the Web can be used to group or select data in the cube, just as additional external information can be added to the cube. This is done without actually storing the external data in the OLAP database. This *logical integration* approach has many advantages over traditional *physical integration*, including the ability to incorporate fresh versions of frequently changing data, such as stock quotes or price lists, as well as being able to quickly change the structure of the cube, e.g., when new kinds of data are suddenly needed or for prototyping purposes. Although good performance results have been shown, performance will often be reduced compared to physical integration. Thus, logical integration is only intended as a *supplement* to physical integration for situations where the additional flexibility is desirable. Recently, we have adapted the mainly theoretical framework of [9] to the TARGIT system and implemented a running prototype of the federated system [7]. This work has pointed out a number of research issues concerning the federation's ability to adapt to changes in its environment. Most notably, a number of vital problems related to the often autonomous and unstable nature of external XML data sources have not been considered. These problems and their solutions are the focus of this paper.

The ability to define dimensions and measures based on data outside the OLAP database provides a great deal of flexibility, but it also introduces a number of new problems. First, since XML data retrieved from external sources is typically controlled by other organizations, these sources are likely to *change* at some time without any way to prevent it. In general, three types of changes may occur: data changes, schema changes, and changes in the accessibility and performance of the source. For *data changes*, the main problem is how to discover that a data update has occurred. Generally, all the external data may have to be fetched each time it is used, but other strategies are also possible, such as letting the data source inform about changes to its data. Handling *schema changes*, e.g., updating view definitions over external XML to reflect the new structure, is a considerable challenge in itself and is treated in another paper [8]. Finally, XML data will often be retrieved from *unreliable* data sources such as on the Internet, which will frequently cause interruptions in the use of external data. For example, if the city information used in an external dimension is found on the Web, this data may sometimes be unavailable, e.g., because a server is temporarily or permanently down, or because the data has moved or changed its name. Even if the data is available, the connection over which the data is retrieved may sometimes be too slow to be useful. The techniques presented for handling these types of problems explore ways to locate alternative sources of the external data, e.g., in a public Web cache such as Google's [4]. Which source to use is determined dynamically during the evaluation of a query. Prototype implementations have been made of the fundamental algorithms, and a set of experiments have been performed on these prototypes demonstrating the power and practical usefulness of the presented techniques.

There has been much previous work on *data changes* in XML sources. In particular, the Xyleme project [16] has spawned a number of papers on this subject. For example, [5] describes an algorithm for determining the differences, a so-called *delta*, between different versions of XML documents, while [6] describes an architecture for subscribing to changes in XML documents. Zhuge and Garcia-Molina [17] have considered the more general problem of maintaining materialized views over graph-structured data when the sources change. However, common to this work is that it does not consider the special issues of logical integration in OLAP databases such as dimensions with hierarchies and correct aggregation of data. Adam et al. [1] present techniques to *detect* schema changes in semi-structured documents. However, they only consider special scientific documents, and thus not XML documents and OLAP integration issues. Our work on improving reliability is somewhat inspired by the Telegraph project [2], which suggests that *dynamic query optimization*, by reordering query operators during query execution, is often profitable in federated database environments. However, we use this technique at a higher level, considering each data source only once, rather than for each tuple. Of course, [2] does not consider the special requirements for logical integration of XML data in OLAP databases. As mentioned, our own previous work [9] describes the theoretical foundations for federations of XML and OLAP data sources. This mainly theoretical framework is extended and adapted to the TARGIT system in [7], which also describes a full prototype implementation of the extension to this system. However, none of these papers deal with the many problems that may interrupt the operation of the system.

We believe this paper to be the first to describe techniques for improving adaptivity when logically integrating XML data in OLAP databases. More specifically, the contributions of this paper are 1) an overview of the *problems* that may occur during the operation of a federation of XML and OLAP data sources, 2) a description of different techniques to handle *data changes* in the federation. 3) a generally applicable technique for improving the *reliability* of XML sources, 4) a description of how to *apply and integrate* these general techniques in the context of our federation, and 5) results from *experimentation* with the techniques.

The remainder of this paper is organized as follows. Section 2 gives an overview of the federated system. Section 3 provides an overview of the potential problems in the federation. Section 4 deals with data change problems, and Section 5 with performance and accessibility problems. The remaining problems are discussed in Section 6. Finally, Section 7 gives an integrated overview of the presented techniques and Section 8 concludes and gives pointers to future work.

2. THE EXTENDED TARGIT SYSTEM

This section gives an overall description of our extensions to the TARGIT system, which were presented in [7]. We begin by briefly introducing the TARGIT system.

The main part of the TARGIT system is an OLAP client, TARGIT ANALYSIS [11], which is known primarily for its ease of use. The OLAP client allows the browsing of multidimensional data using a wide variety of charts, maps, and tables. These objects are interconnected such that an action in one object is reflected in the other objects. For example, by clicking a country on a map, the other objects are automatically drilled down such that they display values for that country. Although the client is aimed at non-technical users, it also covers more advanced functionality such as data mining, user defined measures, and report building. In addition to the client tool, the TARGIT system includes a server that ensures uniform access to different kinds of data sources, and an administrator tool for configuring the server.

The general architecture of the TARGIT system before extension is shown to the left in Figure 1. Whenever the user performs an operation in the client, such as drilling down in a dimension, the client issues a generic OLAP query to the server, which translates this query to a data source dependent query and passes it on to the data source. Two different types of sources can be accessed through the server: Relational tables and Microsoft Analysis Services cubes. When using relational tables, a star or snowflake schema is built from a set of tables in the administrator tool; otherwise the cube is typically constructed in Microsoft's Cube Editor [12].

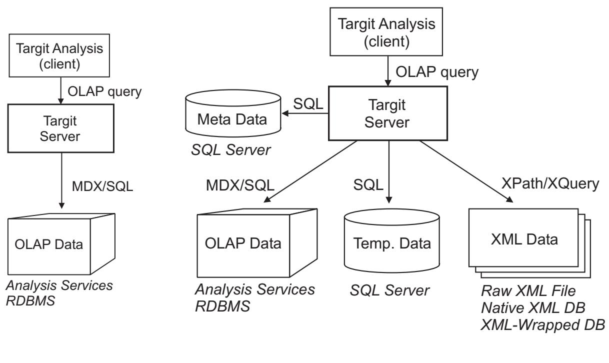

Figure 1: The original (left) and extended (right) TARGIT system.

The TARGIT system has been extended with federation facilities by adding the ability to handle external dimensions and measures without having to integrate the data physically in the OLAP database. To users, the external dimensions and measures are indistinguishable from ordinary dimensions and measures. Thus, the goal of the federation is to offer users complete *transparency* with respect to the *location* of data, the external data's *schema*, and the methods that are used to *access* the external data. Note that data in this setting is only stored in either the OLAP or the external database, i.e., this functionality is different from the normal OLAP ability to "drill through" the cube down to the underlying relational data. External dimensions are added by using an existing level in the cube to identify a hierarchy of elements in the XML document, while a combination of levels is used to identify external measures. We will briefly illustrate how this is done with an example:

Example 2.1 Assume a Book Sales cube already exists with the dimensions Store, Product, and Time, and the measures Price and Quantity. Assume the availability of an XML document containing information about authors and books with the nesting of elements Authors(Publisher(Author(Book(Title)))), where all authors and books (Authors element) are grouped first by publisher, then by author, and finally by the individual books with their associated titles. A new logical Author dimension can be constructed in the cube containing the Author and Publisher information from the document as shown in Figure 2. The correct author for each fact in the cube can here be identified automatically by matching the Title values in the Product dimension and the Title elements in the XML document. In a similar way an external measure Delivery Expenses can be added to the cube if an XML document was available containing for example delivery expenses by city and year. These new measure values would of course only be visible in the cube for the level combination (City, All Products, Year, All Authors) and higher combinations. This behavior is similar to the semantics of a regular join between two cubes [12]. The external dimension Author and the external measure Delivery Expenses can then be used in exactly the same way ordinary dimensions and measures are used. □

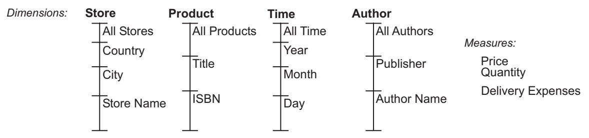

Figure 2: Example Cube with external dimension and measure

For external dimensions, each dimension value is used to identify the external XML values that correspond to this dimension value, and similarly with combinations of dimension values for measures. One of the main challenges is how to handle the fact that there need not be a one-to-one correspondence between the values in the cube and the XML values. A number of solutions are possible here, and different solutions may be best in different situations [9]. Missing XML values can for example be handled by adding a special N/A-value indicating that the value is not available, while multiple XML values can be handled by simply picking one of them. Which behavior is wanted, can be specified when the external dimension or measure is defined. External dimensions and measures are defined by giving an XPath expression, or a graphical equivalent to this, that identifies the values of interest by referring to one or more levels in the cube. For example, if `$T` is bound to each of the values in the Title level, `/Authors/Publisher/Author[Book/Title=$T]` may be used to identify the values needed to construct the new Author dimension in Figure 2. More general ways of defining external data also exist [7, 9].

The general structure of the extended TARGIT system is shown to the right in Figure 1. (The extended TARGIT system will also be referred to as the *federation* in the following.) Instead of having only an OLAP data source as in the original system, three different data sources, also known as *components* in federation terminology, participate in the federation: an OLAP component, an XML component, and a relational component for performing temporary calculations during the evaluation of a query. Because of the processing overhead entailed by changing the structure of a cube in most OLAP systems, including Microsoft Analysis Services, external dimensions and measures cannot generally be implemented simply by adding them physically to the cube at query time. Consequently, a different approach is taken here. The basic idea is to evaluate a multidimensional query that refers to external dimensions and measures by constructing and evaluating an OLAP query and a set of XML queries separately, and then combining the results of these queries using a relational database. A relational DBMS is used for the temporary component because the final result can easily be computed by joining the fact table resulting from the OLAP component query, and tables containing the external data. In order to achieve acceptable performance with this approach, it is assumed that the temporary tables can be stored on the same machine as the OLAP database. These tables will mostly fit in RAM, since they typically contain only data that is to be displayed to a user, and thus, amount to at most a few hundred rows. In most cases, all the necessary aggregations on the cube can be performed in the OLAP system, thereby exploiting its pre-aggregations. In addition to these three components, an auxiliary component stores metadata used in the query evaluation such as specifications of the external dimensions and measures. Both the temporary and the metadata component use the MS SQL Server DBMS.

With this overall architecture, the evaluation of a multidimensional query referring to external data is performed in these main steps:

1. Split the multidimensional query into a pure OLAP query and a set of XML queries.
2. Evaluate these queries in parallel.
3. Join the resulting OLAP fact table and the tables containing external data thereby producing a new fact table.
4. Perform any additional grouping and/or selection on the combined result.

The XML data can be fetched from a number of different sources, ranging from raw XML files to full-blown XML DBMSs. In fact, most other data sources can be wrapped in XML format and can therefore be used. The source may or may not support queries directly, and consequently, it may sometimes be necessary to fetch the entire document and query it locally.

Obviously, the wide variety of possible sources makes optimization extremely difficult, but a number of effective techniques have been presented [7, 9]. One of the most important is *caching* of both raw XML data and of query results containing both OLAP and XML data. This can speed up the federation significantly, in particular if external components are slow. Another important technique is *inlining* of selection predicates that refers to external dimensions. Here, the selection predicate is translated into an equivalent predicate that only refers to the *cube* values that were used to identify the external data. In this way all selections can be evaluated in the OLAP system. As a proof-of-concept and to evaluate the effectiveness of these techniques, a prototype has been implemented. The experiments with this prototype have shown that logical integration of external dimensions and measures is indeed a viable alternative to physical integration, in some cases even outperforming physical integration [7, 9]

Next, we consider the many events that may occur, which can interrupt the federation's operation. These events, the problems they cause, and techniques to handle these problems are the subject of the remainder of this paper.

3. POTENTIAL FEDERATION PROBLEMS

This section discusses which problems may occur during the operation of the federated system, causing a complete or partial interruption of its services. In sections 4 to 6 a number of techniques will be presented that handle or reduce these problems in an attempt to make the federated system more robust.

The problems considered in this paper fall in two main groups, *changes in data and metadata* and *problems with the accessibility of the components*.

The first group of problems may occur when a schema changes or when a data source changes in terms of the data it stores. Schema changes can be top-down, i.e., by modifications to the *federated* schema (also known as the *integration* schema [10]), or bottom-up, i.e., by modifications to the *component* schemas. Top-down schema changes are discussed in Section 6, while bottom-up schema changes are treated in another paper [8].

The second group of problems includes sudden reductions in performance and complete interruptions in the access to components, both of which are frequent when dealing with Web data. Although the solutions to these problems are related to the general performance optimization problems discussed in [7], these more sudden problems require explicit actions, such as trying to locate a new data source or informing the user.

Component changes and accessibility problems may occur for both XML and OLAP components. However, we will focus on XML components since these are more difficult to handle than OLAP components, which will mostly be maintained by the same organization that operates the federation. Also, most of the problems with OLAP and relational components are already handled in the existing TARGIT system.

In addition, a number of other and more general problems may occur, e.g., related to the stability and performance of the HW and OS or to radical user actions such as terminating (a part of) the system. However, since these problems are no different from the problems that can be experienced in other systems, including TARGIT's existing system, they are outside the scope of this paper.

In the remainder of this paper we will discuss the problems with changes and accessibility further, with the emphasis on XML components, and present techniques that will eliminate or reduce these problems. Sections 4 and 5 discuss problems with *data changes* and *accessibility* of XML components, respectively. Section 6 discusses the problems with OLAP components as well as top-down schema changes. Section 7 describes the integration of the presented solutions.

4. XML COMPONENT DATA CHANGES

We now discuss the problems concerning *data changes* in XML components and how changes can be handled in the federation.

In its basic form the federation handles changes in XML data automatically because XML data is only integrated *logically*, i.e., it is not stored in the cube but is fetched every time it is needed. However, for performance reasons this model is not always followed and instead the XML data may be cached locally, often providing large performance gains. To ensure that a recent version of an XML document is used, a *cache timeout* value is specified for each document, ensuring that cached XML data is expunged when its age exceeds this value. Thus, if a document changes very often, caching can be entirely disabled for a document by choosing a cache timeout of zero. This timeout is either chosen by the DBA (Database Administrator) based on knowledge about the component, or it is selected automatically by recording the document's change frequency.

Currently, only few of the most widely used XML databases offer to inform its users when data changes occur. However, systems such as the XML Web-warehouse Xyleme [5] allow users of XML data to subscribe to changes in an XML document of interest. When changes are made to the XML document subscribers automatically receive information that makes it possible to construct the new document from the old one. Transferring only the changes, also known as a *delta* [3], is, of course, usually much faster than transferring the entire document each time it is changed.

Several protocols have been proposed for delivering XML data (see, e.g., [15] for a comparison), some of which includes the ability to deliver *changes* to XML data. As an example, W3C has proposed the Information and Content Exchange (ICE) protocol for automatically informing about and requesting changes to XML documents [13] (ICE is supported by major companies such as Oracle, Sun, and Adobe). Thus, ICE defines both *push* and *pull* methods for keeping a copy of an XML document up-to-date. For components supporting the ICE protocol it is therefore possible to avoid the full retrieval of XML documents using two general strategies: Requesting the changes each time a query is issued or keeping the local copy updated by subscribing to changes. These two strategies are illustrated in Figure 3.

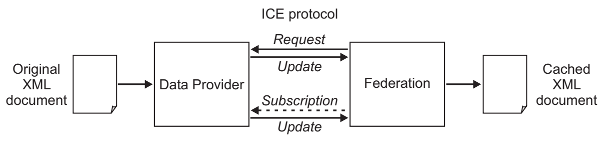

Figure 3: Two methods for keeping a cached copy of an XML document updated using the ICE protocol. The dashed line refers to the fact that subscription only occurs once, whereas requests and updates occur when data has been changed.

By using the subscription strategy the cache is updated whenever a change occurs, and thus, it never expires and the external data is always fully available when needed. However, the overhead of keeping a document updated may be significant, in particular if changes are common. Thus, which strategy to choose depends on a number of technical aspects such as the change frequency, the typical size of the changes, and the general system load, as well as things that are more difficult to determine automatically, such as what other uses the system has and how responsive the system should be. For this reason the choice between these two strategies is made by the DBA when specifying the external data.

5. PERFORMANCE AND ACCESSIBILITY PROBLEMS FOR XML COMPONENTS

XML data is often retrieved from components that are outside the control of the federation, e.g., on the Web. For this reason, little can typically be done to ensure that these components are working and always perform sufficiently well to be useful. What we *can* do is to have a set of alternative strategies to follow when the performance of an XML component is inadequate. In this section we discuss three such strategies for handling poorly performing XML components: allowing several alternative components to be specified, using locally cached data, and trying to locate the data elsewhere.

An XML component may suddenly become slow for many reasons, e.g., because the component machine is overloaded or the network is congested. If the problem is not the time it takes to get a *reply*, but rather the time it takes to return *all* the data, an incremental approach may be useful. Here, only a part of the result is presented to the user, and, as the data arrives, the remaining data is processed and presented. This can be accomplished with a straightforward extension to the integration strategy presented in Section 2: Rather than waiting for all the XML data, the XML and OLAP data should be combined when a certain amount of XML data has been received and a certain time has passed. When more XML data arrives, this is combined with the OLAP data and appended to the partial result constructed earlier.

Often this incremental approach is insufficient. The problem may be that it takes a long time to even receive a reply or the XML component may simply return data too slowly to be useful. As a special case, the component may be completely inaccessible, either because the XML document has been removed or because the retrieval of the document is somehow obstructed, e.g., the network connection may be lost. There is no essential difference in the way these problems can be handled. Whether indicated by a timeout or an explicit error, such as the 404 error that is common on the Web, the current XML component is in effect useless, leaving only two options: Finding an alternative data source or informing the user about the problem. A special case is when the XML document has been replaced by another document, because then an answer *may* be returned depending on the particular query. Furthermore, if the same information is available in the new document, only with a different schema, it may be possible to automatically identify the original information by inferring how the new schema relates to the old one. The issues related to replaced documents and schema changes are discussed further in another paper [8]. In this section, we concentrate on the situation where an XML component exhibits poor performance or, as a special case, becomes completely inaccessible.

Our general strategy for alleviating both of these problems is to allow several data sources to be specified for an XML document. This often increases performance since the fastest source can be used and also provides much better tolerance against inaccessible components. This may, of course, not always be sufficient, e.g., if the XML document is removed permanently and no alternative exists. However, for many situations, these strategies will ensure that an alternative data source can be used, thereby hiding the problem from the user.

5.1 Trading Consistency For Speed and Fault Tolerance

Using more than one data source in order to improve performance or tolerance against inaccessible data may cause problems with the *consistency* of data. By allowing different specifications of external data, the resulting dimensions or measures may also differ, depending on the specific document used to answer the query. When used within a short time-span this may confuse a user, in particular, if he is not aware that the dimension or measure is based on external data. A user's confusion over data from different sources is not fundamentally different from the confusion that may arise when a *single* data source changes. However, the changes that occur in a single document will typically be more controlled and occur less frequently than if, e.g., five different documents are used interchangeably during a user session.

The question is what degree of consistency is required. Even warehouse data changes sometimes and it should almost always be acceptable to update the external data at the same time. However, one of the main arguments for integrating external XML data *logically* is to always have updated information. Consequently, it will typically be acceptable for the data to change more often, even within user sessions. However this depends on a number of things, including the semantical differences between the different sources, the user's understanding of and expectations for the data, and the requirements for good and stable performance. Consequently, these choices must be made by someone with a thorough knowledge of the data and system requirements. For this reason, the federation allows a DBA to specify the acceptable frequency with which a shift can be made from one document to another.

A special case of this is when only a single data source can be trusted. In that case, the problem should not be hidden from the user, but instead he should be informed about it when it occurs, either explicitly or by simply not showing the data. In other situations, the goal is to provide as high a degree of location transparency as possible, and thus, as much as possible must be done to find an alternative document. Often the goal will be somewhere in-between, but because of these subjective questions about the utilization of the data, the choice of strategy for finding alternative data sources is also made by the DBA when specifying external data.

5.2 Using Alternative Data Sources

The federation is made more tolerant of inaccessible XML data by extending the specification of external data to allow *multiple* XML documents for each external dimension or measure. Thus, if one document is inaccessible, another document can be used instead. Also, a different XPath query can be used to identify the external data in each document, and consequently, the documents are not required to be identical or even have the same schema.

Besides improving fault tolerance, specifying multiple XML documents will typically lead to significantly better *performance*. This is achieved by, in principle, always *starting* queries on *all* or a subset of the specified documents, *waiting* for one of them to return the answer and then *canceling* all the other queries, thereby always using the fastest data source. A primary decision is whether to cancel the queries when the *entire* result has been returned or when the *first part* of the result has been returned. The first option has the advantage that it always uses the fastest data source, but if there are many alternative documents and the amount of data retrieved from each of them is large, the overhead may be substantial, not only for the components but also for the federated system. This overhead will be much smaller for the second option, at least for the federated system, because all the other queries are cancelled as soon as any data is received. On the other hand, the query that returns *something* first need not be the same that also returns the *full result* first. The actual strategy used in the federation is a combination of the two: If many alternative components are specified, a query is issued for a subset of them (default is 8) and the three first queries (default) to return something are allowed to continue until the first of them has completed. In this way, the immediate response time is taken as an indication that the component is fast, without betting everything on a single component.

The burden on the components is not as large in practice as it may seem. The number of queries will often be limited since a query is only issued when it is not available in the cache, and with typical cache timeout times ranging from 10 minutes to several hours, this is relatively rare. Also the defaults may be reduced to further limit the load on components. If it is important to keep the load on components at a minimum, an adaptive variant of this strategy can be used, along the lines of [2]. Here, only a single query is issued, but if a reply is not received within a short time (default 2 sec.) another query is issued, etc. until some query returns the result or a timeout occurs. The component with the shortest reply time is then issued first the next time.

If the cache contains a valid copy of the desired document, this is, of course, used, rather than accessing any data sources and in that case it is not discovered that the original document is inaccessible. Only if the cached copy has expired are the original data sources accessed. However, if no alternative data sources are specified or if they are all inaccessible, the cache may contain useful information even though it has expired. Thus when a cached copy expires, it is not expunged before the space it occupies is needed again. This means that the expired copy can still be used if the original data becomes inaccessible, following the idea that it is better to have old data than no data. However, as discussed above, this is not always true, and hence, the use of expired data in the cache can be explicitly disabled. Alternatively, the user can be notified whenever an old version of the data is used, if this is requested when specifying the data.

If the requested data is not in the cache, the data may still be found elsewhere. For example, the Xyleme Web-warehouse [16] stores a large number of XML documents and could be searched for a copy of the document. Where to search depends on the type of document: If the original document is found on the Web, places like Xyleme [16], and Google's Web-cache [4][1], whereas if the document is located on a restricted intranet, a local proxy server or some backup device are more likely alternatives. Since these sources may vary, it is possible to specify the alternative search strategies on a per document basis. When specifying these strategies, it is also possible to specify more loose criteria, such as "Any document with the same file name" or "Any document that conforms to the same DTD," which can be used to search for data.

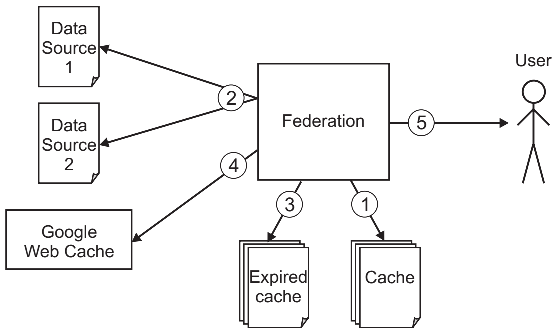

Figure 4: Example of how XML data is retrieved in the federation.

Figure 4 shows an example setting with the different approaches for locating external XML data. The numbers 1–4 refer to the order in which actions, such as trying to retrieve data from a source, occur. How aggressive this search should be, is determined when specifying the data. If neither of these approaches result in a useful copy of the data, such as when an alternative does not exist, the only option is to inform the user about the problem (5). The typical speed and quality, in terms of freshness, of the different types of sources are summarized in Table 1.

Data Source	Speed	Quality
Local cache	Fastest	Fresh
Original data source	Fast	Freshest
Expired cache	Fastest	Old
Backup source	Fast - very slow	Very old

Table 1: Typical features of different XML data sources.

The combined strategy for locating XML data is outlined in Algorithm 1. For simplicity, the algorithm assumes that as much as possible should be done to locate the data. The input to the algorithm is a set of specifications of external data to be used in constructing a dimension or a measure. Each specification consists (essentially) of a URL that points to an XML document and an XPath query that extracts the dimension or measure data from the document. The output of the algorithm is the result of evaluating one of the queries on its corresponding document. The steps of the algorithm follow the discussion above.

Algorithm 1 FetchXML($Specifications = \{(d,q) | (d \in Documents \wedge q \in Queries)\}$): Result

```
if a result of some $q_i \in Queries$ is in the local cache
      and it is not expired then
    return the cached result
for all $(d_i, q_i) \in Specifications$ do
    Start $q_i$ on $d_i$
Wait for the $n$ first queries $Q_n = \{q_{j_1}, \ldots q_{j_n}\} \subseteq Queries$
  to reply or for Timeout
Kill off the remaining queries $Q = Queries \setminus Q_n$
Wait for some $q_j \in Q_n$ to complete or for Timeout
Kill off the remaining queries $Q = Q_n \setminus \{q_j\}$
if some $q_j$ completed successfully then
    return the result of $q_j$
else {Timeout or all $q_k \in Q_n$ returned an error}
    if result is in local cache and it is expired then
        return the cached result
    else if some copy $d'_i$ of $d_i$ is found in backup locations
    then
        return the result of query $q_i$ on $d'_i$
    else {We give up}
        Inform user of missing data or proceed without data
```

In general, trying to locate data from several different sources will take longer than simply retrieving the data from a single source. However, as discussed above, the use of alternative specifications *reduces* the average retrieval time significantly, and the time it takes to search the local cache is negligible. Thus, it is only the last step, i.e. searching backup locations, that may delay the retrieval significantly. How much, depends, of course, on the speed and type of the data sources that are searched. Consequently, this option will typically only be used when these sources are relatively fast or when it is unacceptable to give up and inform the user about the problem. Fortunately, the most widely used archives of Web data, e.g., Google [4], are as fast or faster than most websites. Thus, for many common cases where Web data is used, the full strategy for finding XML data can provide a much higher tolerance against unreliable components, at speeds that are comparable to and often better than using a single data source.

Figure 5 shows an experiment where Algorithm 1 is applied to a set of copies of the same XML document located at different sites. This corresponds to the situation where an external dimension is specified using the XML document, and used for selec-

[1] Although it is possible to *search* explicitly for XML documents on Google, it currently does not *cache* XML documents as it does for HTML documents. However, since the number of XML documents on the Web is increasing rapidly, it will probably not be long before XML documents are included in the Google cache.

tion/grouping in an OLAP query, meaning that the XML document should be retrieved from the fastest source. The size of the document was 137 KB, which we consider a realistic size for an external dimension.

The top three sites were the first three to reply, and at the time the last of these replies were received, the remaining five queries would have been cancelled in the algorithm (indicated by the zigzag lines). However, in this case they were allowed to continue to measure their evaluation times. Similarly, when the first of the three remaining queries completed, the two other queries would have been cancelled by the algorithm.

Rather than using a single site, which, in this experiment, on average would have an evaluation time of 8.4 sec. and could be as high as 19.8 sec., this time is reduced to 2.7 sec., which was also the optimal time in this case. Also, it can be seen from the experiment that the reply time is a fairly good indication of the total evaluation time. Consequently, the assumption that the fastest query will be among the first three to reply is supported by this experiment.

The bottom measurement shows the time it took to retrieve the document (actually an HTML document of the same size) from the Google cache, after, unsuccessfully, searching the local cache for expired results. The local cache was maintained in a SQL Server database. In this experiment, the Google time was actually relatively close to the optimal time, but this need, of course, not be true for other kinds of backup sites. Also, since the resulting copy of the document may be outdated, it is only used if all the other sources return an error or time out.

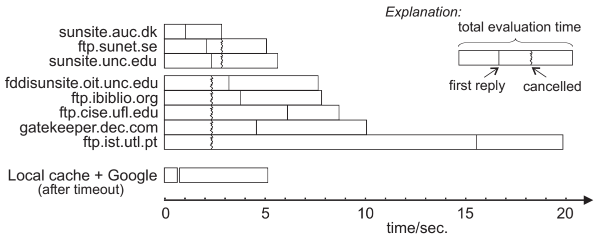

Figure 5: Reply and total retrieval times for different sources.

In summary, the experiment has shown that the proposed technique for finding alternate data sources is practically viable.

6. ADDITIONAL CHANGE PROBLEMS

This section discusses the problems that are related to OLAP components as well as to changes in the federated schema. We only discuss these problems briefly since most of them have already been dealt with in the TARGIT system. Also, they are not as difficult to handle as the problems with XML components, mainly because OLAP components are typically controlled by the same organization that operates the federation.

The only type of problem that is not found in the existing TARGIT system is changes to the *federated schema*, i.e. top-down schema changes that redefines the relationships between OLAP data and external data found by evaluating queries over XML documents. Such changes are typically made by the DBA when user requirements change, when XML documents are moved or changed, or when OLAP schemas are changed. The problem with federated schema changes is that a part of the cache may be invalidated by the change, and thus, the cache should automatically be restored to a consistent state after the change. It is theoretically possible to update some types of cached results to reflect the changes. However, since these schema changes will be relatively rare, the solution adopted here is simply to discard all cached results that are affected by the schema change. Also, lo sing a part of the cache contents will not stop the federation from working, but only make query processing slower. Thus, a cached result is discarded if it contains data from an XML documents whose identifying query is changed, or if the relationship between OLAP data and XML data has changed. Notice that changes in the relationships only affect cached results where OLAP and XML data has been combined. In addition to these, the results of XML queries may also be cached, and these are only affected by changes in the identifying XPath queries.

Changes to the *OLAP data* can also invalidate a part of the federation's cache. Like changes in the federated schema, OLAP data changes occur much less frequently than XML data changes and thus the same solution is used: discarding the affected part of the cache. However, changes to OLAP data typically require substantial processing, which means that the cache contents will often have expired anyway. The OLAP component's *schema* may also change. Such a change, e.g., renaming a level in the cube, may invalidate the part of the federated schema that refers to the changed levels. Insertion and renaming of dimension values *may* also invalidate the federated schema, but only if these values have explicitly been used to identify external data. Often the level name will be used instead. However, detecting which changes affect the federated schema is easy, and the affected parts of the schema can then be presented to the DBA. Renaming of levels can even be handled automatically.

A relational component is used by the federation for storing meta data and cached results. However, only the federated system should be able to access these tables, thereby eliminating the need for actions that concern changes to the relational component.

7. AN INTEGRATED APPROACH

This section shows how techniques described in this paper are integrated into the component architecture of the federated system. Only the parts of the system that are affected by the techniques are discussed here. The remaining components are described in [7].

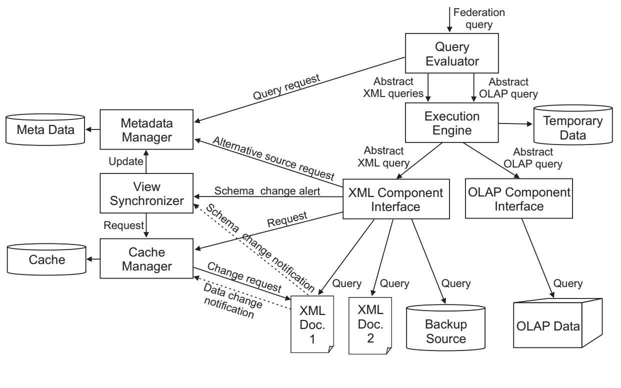

Figure 6: Overview of the federation with the presented techniques.

The basic component architecture of the resulting federated system is shown in Figure 6. When a federation query, referring to both ordinary and external dimensions and measures, is sent from the client to the server, it is first split into its source-independent XML and OLAP parts. This is performed by the *Query Evaluator*, which also determines the exact query evaluation plan based on a cost analysis. The actual coordination and parallel execution of the queries is handled by the *Execution Engine*. During this evaluation a temporary database can be used to store intermediate results. To ensure independence from the specific sources, the translation of abstract queries, i.e. queries that do not refer to a specific source, into source-specific queries is handled by the *XML/OLAP Component Interfaces*. They are also responsible for checking the cache contents before issuing any queries, which is done through the *Cache Manager*. (Although not shown in the figure, the Query Evaluator may also request information about the cache contents, for use during query planning.) The *Metadata Manager* maintains information defining external dimensions and measures as well as their relationships to the OLAP database. This information includes the views used to extract dimensional or measure data from external XML sources.

The *Cache Manager* maintains the cache contents, and as such it is also the component that is responsible for keeping the cache updated when sources change. Thus, the Cache Manager handles incoming change notifications, e.g., through the ICE protocol as described in Section 4. If automatic notification is not supported by a source, the Cache Manager may explicitly request changes from the source at a specified interval.

The choice between different XML data sources, as described in Section 5, is made by the XML Component Interface. When an abstract XML query is to be evaluated, the XML Component Interface first requests the identifying queries and a list of alternative data sources from the Metadata Manager. This information is then used to retrieve the data from these sources as described by Algorithm 1. This may include accesses to the cache, for both fresh and expired data, to several alternative data sources, and to backup sources, which may contain outdated data. If all these attempts fail, an error is reported back to the Execution Engine.

Schema changes are handled by the *View Synchronizer* [8]. When schema changes are explicitly notified by the XML component, this information is handled directly by the View Synchronizer, which then updates the metadata. When changes are not explicitly notified, the detection of schema changes is performed by the XML Component Interface by assessing the validity of the returned result [8]. The View Synchronizer then tries to determine the new view query based on earlier versions of data found in the cache. The resulting query is returned back to the XML Component Interface along with an indication of the quality of the matching between the old and the new schema. If the match is satisfactory, the XML Component Interface uses the new query, and if not, an error is reported back to the Execution Engine.

Apart from the definitions of external dimensions and measures, the Metadata Manager also maintains all information about the external XML components used in the presented techniques. This information includes for example what notification facilities and protocols are supported by the source, the frequency of change requests, alternative queries and sources, synchronization rules, and weights used in schema element comparisons.

In addition to the previously implemented prototype system [7], which covers most of the federated system described in Section 2, implementations have been made of the fundamental algorithms for view synchronization and location of alternative sources. Referring to Figure 6, this includes the essential parts of the View Synchronizer and the extensions to the XML Component Interface.

8. CONCLUSION AND FUTURE WORK

In our federated system as well as in many other systems, XML data sources will often be controlled by other organizations, which makes it exceedingly difficult to prevent these sources from *changing*. The changes may affect the data and its structure or it may affect the source's ability to respond to data requests. In either case, the changes are likely to interrupt the federated system's operation.

This paper has presented a set of techniques for improving the robustness of our federated system, in particular with respect to external XML components. We discussed the problem of keeping locally cached XML data updated in case of changes to the data source. The solutions discussed for this problem included periodically *requesting* the changes as well as *subscribing* to changes. Techniques were also presented for improving the *reliability* of external XML components. The main idea was to allow several equally valid data sources to be specified, including possibly outdated backup sources that may be used if all sources are inaccessible. By always trying more than one source and then choosing the fastest, performance can be significantly improved, in addition to the increased reliability. This has been supported by experiments on real-world data sources. Hence, this paper's contribution has been two-fold: it provided an overview and integrated solution to the problems that may occur in OLAP-XML federations, including data changes in OLAP sources, and it presented a technique for improving the reliability of external XML sources. We believe this paper to be the first to provide such techniques for the special setting of OLAP-XML federations.

The most immediate continuation of this work is the refinement of the prototype into an industrial-strength implementation inside the TARGIT system and extensive experimentation with this new system. This is expected to happen over the next few releases. As mentioned above, the techniques can also be applied to a wide range of other applications, e.g., Web Service applications. It would also be interesting to extend the system to handle changes in dimensions over time.

9. REFERENCES

[1] N. R. Adam et al. Detecting Data and Schema Changes in Scientific Documents. In *Proc. of ADL*, pp. 160–172, 2000.

[2] R. Avnur and J. M. Hellerstein. Eddies: Continuously Adaptive Query Processing. In *Proc. of SIGMOD*, pp. 261–272, 2000.

[3] G. Cobéna, S. Abiteboul, and A. Marian. Detecting Changes in XML Documents. In *Proc. of* ICDE, 2002.

[4] Google Inc. `www.google.com`. Current as of June 30th, 2003.

[5] A. Marian et al. Change-Centric Management of Versions in an XML Warehouse. In *Proc. of VLDB*, pp. 581–590, 2001.

[6] B. Nguyen et al. Monitoring XML Data on the Web. In *Proc. of SIGMOD*, 2001.

[7] D. Pedersen, J. Pedersen, and T. B. Pedersen. Integration of XML Data in the Targit OLAP System. Submitted for publication, 2003.

[8] D. Pedersen and T. B. Pedersen. Synchronizing XPath Views. Submitted for publication, 2003.

[9] D. Pedersen, K. Riis, and T. B. Pedersen. XML-Extended OLAP Querying. In *Proc. of SSDBM*, pp. 195–206, 2002.

[10] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[11] Targit A/S. `www.targit.com`. Current as of June 30th, 2003.

[12] E. Thomsen, G. Spofford, and D. Chase. *Microsoft OLAP Solutions*. Wiley, 1999.

[13] W3C. The Information and Content Exchange (ICE) Protocol. `www.w3.org/TR/NOTE-ice.html`. Current as of June 30th, 2003.

[14] W3C. Extensible Markup Language (XML) 1.0 (Second Edition). `www.w3.org/TR/REC-xml`. Current as of June 30th, 2003.

[15] W3C. XML Protocol Comparisons. `www.w3.org/2000/03/29-XML-protocol-matrix`. Current as of June 30th, 2003.

[16] L. Xyleme. Xyleme: A Dynamic Warehouse for XML Data of the Web. In *Proc. of IDEAS*, pp. 3–7, 2001.

[17] Y. Zhuge and H. Garcia-Molina. Graph Structured Views and Their Incremental Maintenance. In *Proc. of ICDE*, pp. 116–125, 1998.