Toward Active XML Data Warehousing

Rashed Salem, Jérôme Darmont, Omar Boussaïd

Toward Active XML Data Warehousing

Rashed SALEM, Jérôme DARMONT and Omar BOUSSAID

Université de Lyon (ERIC - Lyon 2)
5 av. P. Mendès-France, 69500 Bron, France
firstname.lastname@univ-lyon2.fr

Abstract. Warehousing data is not a trivial task, particularly when dealing with huge amounts of distributed and heterogeneous data. Moreover, traditional decision support systems do not feature intelligent capabilities for integrating such complex data. Therefore, we propose an approach for intelligent decision support based on active XML warehousing. We exploit XML as a pivot language in order to unify, model and store complex data. Furthermore, Web services tackle the distribution and interoperability problems of data sources. They are employed for complex data integration and react with active rules for realizing intelligent ETL. In this paper, we focus on the integration phase and propose an architecture for integrating complex data into a repository of Active XML documents, based on Web services and event-driven rules. We have finally developed a software prototype to validate this approach.

1 Introduction

Nowadays, huge volumes of heterogeneous data (e.g., operational data, Web data, text data, multimedia data, etc.) are available over networks. We term these data *complex data* (Darmont et al., 2005). Data may be qualified as complex if they are: diversely structured, represented in various formats, originating from several different sources, described through different perspectives and/or changing in terms of definition or value over time. Complex data must be warehoused into specific storage to be later analyzed for decision-support purposes. The complexity of data renders their structuring and exploitation difficult. However, the classical warehousing approach (Inmon, 1996; Kimball, 1996) is not very adequate when dealing with complex data, particularly in the data integration phase. It is technically difficult, crucial and ill-adapted for handling complex data. Neither classical ETL (Extracting, Transforming and Loading), nor dynamic ETL is alone sufficient for integrating complex data. Therefore, traditional decision support systems need to be extended with intelligent capabilities in order to: increase business competitiveness, automate decision-making, support near real-time decisions, improve consistency in decisions, improve management of data distributed throughout heterogeneous sources and support on-line analysis of complex data. Intelligent ETL is well-suited for achieving most of these promised issues. Intelligent ETL is constituted of autonomous services that are based on an event-driven environment for integrating complex data. It posses a set of services allowing to accomplish in autonomous way the different tasks of the ETL process.

We address these issues by developing an active XML warehousing approach for intelligent decision support purposes. This approach is achieved by utilizing different technologies. The data heterogeneity motivates us to exploit XML for standardizing data into a unified format. XML, a self-describing semi-structured language, is the standard format for storing, modeling and exchanging semi-structured data. It is also becoming the standard to represent business data (Beyer et al., 2005). Moreover, Web services (WSs) can solve the data distribution and interoperability problems by exchanging data between different applications and different platforms. In order to add more functionality to warehoused data, we embed them with calls to WSs. Embedding WSs into warehoused XML documents results in so-called Active XML (AXML) documents. Typically, the goal of AXML is to integrate information provided by any number of autonomous, heterogeneous sources and to query it uniformly (Abiteboul et al., 2008). Thus, XML and WSs have dramatically changed distributed data management by overcoming the heterogeneity, distribution issues and implying some form of dynamicity.

Incorporating active rules or ECA (Event-Condition-Action) rules (Paton, 1999), which are widely accepted in active databases, into the warehousing environment enriches the system with reactive and intelligent features. We can indeed assimilate different integration tasks as Web services. The work of integration services is based on event occurrence. If the rule condition holds to true, actions of invoking integration services are executed. Thus, intelligent ETL services are different from classical and dynamic ETL, by their responsive to ECA mechanism.

In this paper, we propose a general AXML-based integration architecture enriched by active features utilizing WSs. It handles complex data for intelligent decision support. XML and WSs with ECA rules overcome the heterogeneity, distribution and non-interoperability problems of complex data, and keep the warehouse with excellent data freshness. Calls to WSs embedded in AXML documents can also be invoked to refresh their documents with up-to-date data. We finally implement a software prototype to validate our proposal.

The rest of this paper is organized as follows: Related works are surveyed in section 2. Section 3 demonstrates the proposed architecture for integrating complex data into a repository of AXML documents. An example is given in section 4. Section 5 highlights some implementation issues and challenges. Finally, we conclude and discuss future trends in section 6.

2 State of the Art

In this section, we survey the state of art related to XML and active warehousing. These researches are based mainly on Data Warehouse (DW) and XML concepts. They involve XML Warehousing (XW), Active XML (AXML), Active rules (ECA rules), and Active Data Warehousing (ADW).

2.1 XML Warehousing

Numerous researchers address the problem of designing and building XML data warehouses. XML is used for managing, modeling or representing facts and dimensions. Facts are the metrics to be analyzed (e.g., sales), and dimensions are those attributes that describe facts (e.g., product, time, region or salesperson). The main purpose of XML warehousing approaches is to enable a native storage of the warehouse and allow querying it with XML query

languages, mainly XQuery. XML Warehousing research is divided into three families (Mahboubi et al., 2008): Web data integration for decision-support purposes, adapting approaches based on classical warehouse logical models and document warehousing.

Web Data Warehouses. Integrating Web data into warehousing environments is becoming increasingly common. Xyleme's authors build a dynamic warehouse for massive volumes of XML data from the Web (Xyleme, 2001). Several semi-automated approaches are proposed for designing and building Web warehouses, starting from XML data sources and modeled by XML Schemes (Golfarelli et al., 2001; Vrdoljak et al., 2003). However, we consider XML data over the Web as only one source among different sources of complex data . A framework to generate AXML WSs is proposed for materializing the dynamic content of Web sites (Vidal et al., 2008). Yet, AXML WSs can play different roles in active warehousing environments, which we discuss in Sect. 3.

XML Data Warehouses. Several researchers adapt the traditional star-like schemas for XML environments. For instance, the snowflake schema is adapted with explicit dimension hierarchies (Pokorný, 2002). Mahboubi et al. (2008) propose an architecture model to translate the classical constellation schema into XML. This reference DW involves *dw-model.xml*, $facts_f.xml$ and $dimension_d.xml$, for representing the warehouse metadata, facts and dimensions, respectively. Furthermore, Hummer et al. (2003) propose a family of XML-based templates so-called XCube to store, exchange and query data warehouse cubes. Rusu et al. (2005) propose an XQuery-based methodology for building a data warehouse of XML documents. It transfers data from an underlying XML database into an XML data warehouse. Finally, Boussaid et al. (2006) propose X-Warehousing for warehousing complex data and preparing XML documents for future analysis.

XML Document Warehouses. Baril and Bellahsène (2003) introduce the View Model for building an XML warehouse called DAWAX (Data Warehouse for XML). Other authors propose a conceptual design methodology to build a native XML document warehouse, called xFACT (Nassis et al., 2005). It is improved in GxFACT (Rajugan et al., 2005), a view-driven approach for modeling and designing a Global XML FACT repository.

2.2 Active XML

Active XML is considered as a useful paradigm for distributed data management on the Web. AXML documents are XML documents where some of the data are given explicitly, while other parts are given only intentionally by embedding calls to WSs. When one of these calls is invoked, its result is returned to enrich the original document (Abiteboul et al., 2008). There are several issues studied in P2P architectures, such as distribution, replication of dynamic XML data, semantics of documents and queries, confluence of computations, terminations and lazy query evaluation.

Several performance and security issues for Active XML are addressed (Milo et al., 2003), e.g., the problem of guiding the materialization process. Ruberge and Mattoso (2008) handle materialization performance issues, when the result of some service calls can be used as input of other calls. Another issue is continuous XML queries, which are evaluated incrementally as soon as there are new data items in any of their input streams (Abiteboul et al., 2007).

2.3 Active Rules for Databases and XML

Active database behavior (Machani, 1996; Paton, 1999) is characterized by the definition of a set of ECA rules as part of the database, which describes actions to be taken upon encountering an event in a particular database state. These rules are then associated with objects, making them responsive to a variety of events. Events can be simple or composite. A composite event is a logical combination of simple events or other composite events, defined by logical operators such as disjunction, conjunction, sequence, and negation. A simple event corresponds to a data modification operation and is specified by the event's type and the modified object.

Active rules are also proposed for XML. Bonifati et al. (2000, 2001) propose an active extensions of the Lorel and XSLT languages, for using active rules on the implementations of e-services. Bailey et al. (2002) investigate ECA rules in the context of XML data . Other authors describe a general form of active rules for XML based on XQuery and previously defined update languages (Rekouts, 2005).

2.4 Active Data Warehousing

Traditional DWs are passive and all tasks related to analyzing data and making decisions must be carried out manually by analysts. They also offer little support to automate decision tasks that occur frequently. In this context, decision making can be automated for routine decision tasks and the routinizable elements of semi-routine decision tasks, through a novel architecture called Active Data Warehouses (ADW) (Thalhammer et al., 2001). The traditional data warehouse architecture is also extended with analysis rules, that mimic the work of an analyst during decision-making. From different perspective, Tho and Tjoa (2003) deal with the problem of allowing organizations to deliver relevant information as fast as possible to applications that need a near real-time action to new information captured by organization's information system. Also, problems of time consistency (Bruckner and Tjoa, 2001), feeding Real-Time Data Warehouses (RTDW) (Araque, 2003), and refreshing ADW online (Polyzotis et al., 2007) are handled in ADW environments.

Finally, Active XML and WS technologies (Abiteboul et al., 2006) are utilized for building and maintaining domain specific content warehouses, which differ from classical DWs by managing "content" and not only numerical data. Their active content warehouse is used to store data for management purpose.

3 AXML-based Integration Architecture

This architecture pertains complex data integration, the process of combining data residing at distributed and heterogeneous data sources into a unified repository. This repository is fed from different sources of complex data utilizing some ETL services, and yields the store of AXML documents. The workflow of integration tasks is managed and controlled via a set of modules that include metadata, event log, AXML and ECA engine (Fig. 1). XML is the main representation language of this architecture. We here differentiate between two different types of XML documents, which are commonly utilized throughout this architecture. One type is used for representing the warehoused data themselves, i.e., facts and dimensions, which

are intended to be analyzed by analysis modules. The other type is used for representing architecture-aided information such as event log, metadata, and active rules.

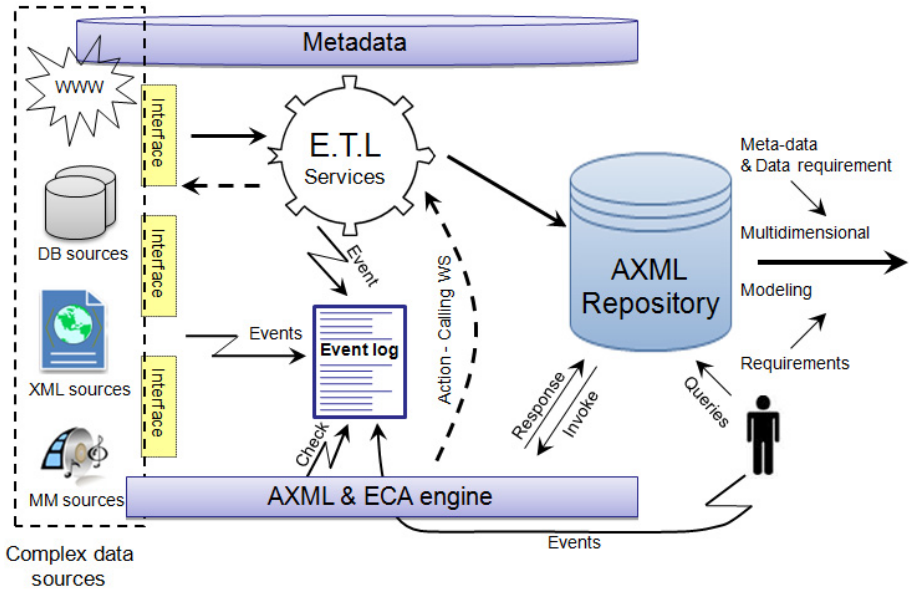

FIG. 1 – *Complex data integration architecture*

In the following sections, we discuss the main components of this architecture, active rules and the flow of data.

3.1 Architecture Components

Sources of Complex Data. Most traditional DWs integrate data from operational and structured data sources. One of the objectives of our architecture is to deal with complex data (Darmont et al., 2005) residing in distributed and heterogeneous data sources.

Data sources can be defined as an *input schema* via a data stewards' Web-based GUI (Graphical User Interface). The input schema can then be translated automatically into XML format. A data steward is someone who understands the business and its processes, and has also enough technical expertise for maintaining the data integration environment. It worth to be noted that input schema can be updated at any time either by adding, altering or dropping data sources. The input schema aims at profiling complex data and defining the relevant entities, elements and/or attributes for each data source. Let us define S as a set of n data sources $(s_1, s_2, \ldots, s_n)$. For each s_i, let E_i be the set of interesting entities and/or elements $(e_1^i, e_2^i, \ldots, e_m^i)$. Some samples of interesting data follow.

- In operational and structured data sources, the input schema defines interesting tables, fields, and/or specific records, etc.
- In Web data sources, the input schema defines texts, links, images, etc.

- In XML and semi-structured data sources, the input schema defines relevant elements, attributes, etc.
- In text and flat-file data sources, the input schema defines relevant keywords, summaries, concepts, entities, etc.
- In multimedia data sources, the input schema defines media (e.g., image, video, movie, audio, etc.) object type, color, shape, content, description, etc.

Technically, when a data steward accesses a database via the Web-based GUI, he can browse the database structure and select interesting tables. He can also specify interesting records graphically or using a query language (i.e., SQL). The same procedure is applied for other data sources. For example, when identifying an XML data source, the data steward can browse the document hierarchy and then select interesting paths.

ETL Services. ETL services are responsible for extracting data from data sources, transforming them into XML and loading them into the repository of AXML documents. These services deal with heterogeneous data sources via different interfaces. The role of these services is not only to integrate low-level characteristics (e.g., image color, image texture, image shape, file size, file location, creation date, update date, etc.), but also to integrate semantic characteristics of complex data (e.g., image content, relationship between objects, etc.). However, integrating semantic features from complex data requires semantic data retrieval and mining techniques. We exploit WSs on developing ETL tasks, which results in multiple advantages. WSs can solve data distribution and interoperability problems. WSs can also be embedded in AXML documents and then can be invoked to refresh the documents on the fly.

Furthermore, extraction tasks face challenges due to the disparity of sources, which results in the need for developing complex services in order to deal with different structures of data sources. Some of these challenges are: How to access appropriate data sources? How to unify the various formats of data? How to define the accurate mapping between source schemas and integration schemas? How to ensure the quality, accuray, completeness and consistency information?. There are also other challenges related to security, accessing Web services, licensing and developing integration services. To overcome these issues, we propose developing several source handler services, each of which deal with a specific type of data sources.

Transformation and cleaning tasks face some conflicts related to both structure and data levels. Structure level conflicts have different natures:

- *Domain conflicts* where different sources of complex data describe the same domain using different structures;
- *Naming conflicts* where names assigned to the same objects, elements and/or attributes vary among data sources;
- *Schema conflicts* where the same objects and/or entities may have different schema models in different data sources.

Data level conflicts can be arised when similar data are represented using different ways:

- *Data value conflicts* where different ways to instantiate a certain element through different data sources (e.g., France vs. FR);
- *Data unit conflicts* where the same element can be instantiated using different measuring units, originating from heterogeneous data sources (e.g., Dollar vs. Euro);

- *Data representation conflicts* where different representations are used for the same element through different sources of complex data (e.g., date format).

Such conflicts require accurate mapping and matching between contents and structure of heterogeneous data sources. Moreover, developing transformation services that based on metadata can avoid such conflicts. Loading transformed data and writing them into AXML documents is not always simple. It should discriminate between "new data" and "update data" at loading time and determine their target documents. Also, the most updatable data should be written and expressed as calls to WSs in order to be refreshed later with up-to-date data.

Active XML Repository. The AXML repository is the target where data are loaded through ETL services. The output of integration tasks are AXML documents. Some issues such as: What data of AXML documents should be given explicitly? and what others should be given implicitly? must be tackled by the administrator/data steward to determine updating frequencies. The explicit (or static) parts of AXML documents are meant to be defined like traditional XML elements (i.e., <elementTag>text content</elementTag>), but the implicit (or dynamic) parts are calls to WSs (i.e., <call-ws>contents include WS URL and its parameters</call-ws >). Moreover, XML schema supports complex types and further additions. It is considered as a grammar for the warehoused AXML documents. AXML documents induce multiple benefits. Firstly, we exploit XML as a pivot language to unify, model and store complex data. Secondly, WSs are the best solution to overcome the problem of non-interoperability and distribution of data over distributed and heterogeneous data sources, where WSs interfaces can be applied to evaluate specific queries in remote, independent and heterogeneous data sources. Thirdly, when querying AXML documents, their embedded WSs are invoked to refresh contents with up-to-date data. Lastly, AXML documents are stored in a native XML database (NXD). NXD systems support managing, storing, querying and updating XML data. On the contrary, no browser can read AXML documents yet. As well, AXML finds difficulty to be supported and processed by end-user applications. To parse AXML documents, we propose scanning requested AXML document in order to find the embedded WSs, then invoke them and return with results in the same place of the document.

AXML & ECA Engine. The AXML & ECA engine plays an important role with metadata and event log for managing the work of our architecture. It is employed for several purposes:

- Ensuring the availability of data sources,
- Ensuring accurate data elements and accurate sources-to-targets mapping,
- Managing event triggering and evaluating their conditions throughout the architecture (Sect. 3.2),
- Checking event log for events against their impact on the AXML repository,
- Invoking specific actions when conditions are met, i.e., execute specific ETL services,
- Managing the evaluation of AXML WSs, then return their results to feed and refresh the documents with up-to-date data.

An important issue is timing for invoking WSs, which can be done either *explicitly* (e.g., daily, weekly or after occurring some eventšššššššššs) or *implicitly* when data are requested. Evaluating WSs may depend on data update rate, which differs w.r.t. the nature of various applications.

```
<?xml version="1.0" encoding="UTF-8" ?>
<EventLog >
   <session sessionID="1234" machineID="127.0.0.1" userID="user00" start="2009-09-03 09:14:57 ">
    <etl_event status="success" />
      <extract_evt eventID="00001" time="09:16:12" source="http://www.shopping.fr/"
            fact_key="5000001" fact="product_price" />
      <transform_evt eventID="00002" time="09:16:56" fact_key="5000001" fact="product_price"
            type="currency" from="euro" to="dollar" />
      <load_evt eventID="00003" time="09:17:43" fact_key="5000001" fact="product_price"
            destination="http://127.0.0.1/repository/products.axml" />
    <etl_event />
    <structure_event status="success">
      <add_web time="09:20:17" url="http://www.amazon.fr/" content="text/html" />
      <add_attribute time="09:21:57" base="dsn:retail-sales" domain="intranet"
           attribute="quantity_sold" />
      <modify_element time="09:22:46" doc="../products.xml" domain="local" old="product_SKU"
           new="prd_ref" />
      <remove_doc time="09:23:24" domain="intranet" path="../spreadsheets/comparing.xls" />
      <associate_ws time="09:24:49" provider_ws="local" ws_path="../getLowestPrice( int id)"
        target="../repository/ products.axml" t_element="lowest_price" />
    </structure_event >
    <err_event time="09:20:33" status="failed" />
      <message value="Cannot open remote source 'http://img.shopping.fr/img/101/1011100.gif  " />
    <err_event />
     <!-- etc. -->
  </session>
   <session><!-- etc. --> </session>
</EventLog >
```

FIG. 2 – *Event log*

Event Log. The main purpose of the *event log* is to maintain all events detected throughout the architecture, either by data sources, ETL services modules or AXML repository. Users' queries are also important events that are maintained by the event log. Descriptions of events are logged in the event log into XML format (i.e., event type, event description, date, time, status, error message, and so on. Fig. 2). Thus, the event log is not only intended for handling exceptions, but also provides history of integration tasks and user's event descriptions. This log is checked regularly by the AXML & ECA engine. Furthermore, it can be analyzed later to build data and user profiles in order to enhance system performance. Such profiles can be used for recommendation. Some examples of events that result in logging are: changing the list of data sources specified in the input schema, changing attributes of a specific data source, integrating data from their sources (extracting, transforming or loading), and so on.

Metadata. We believe that the better metadata are defined, the easier automation is achieved for different processes of the architecture. An XML Schema describes the structure of an XML document, including metadata. Figure 3 represents an XML Schema as a metadata grammar. However, metadata of Web resources are modeled using Resource Description Framework (RDF). Metadata can be of various natures: data source descriptions, ETL services, storage structures of the AXML repository and data refreshing policies. Hence, metadata can be considered as a grammar for AXML documents and as configuration files for data sources and

ETL services. Metadata may contain the following information:

- Description of data sources, types, interfaces, descriptive information such as ownership, source, format, structure, update frequency, access methods, and so on;
- Description of ETL services, including sources-to-extracting services interfaces, transformation and cleaning rules, data elements-mapping names, targeted AXML documents, updating policies, data lineage (e.g., tracing warehoused data back to their sources), and so on;
- Description of AXML repository's structure, physical location of AXML documents, indexing, refreshing plan, and so on.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema ">
  <xs:element name="image" type="imagetype"/>
  <xs:complexType name="imagetype">
    <xs:sequence>
      <xs:element name="sourceID " type="xs:string"/>
      <xs:element name="MIMEType " type="xs:string"/>
      <xs:element name="Host " type="xs:string"/>
      <xs:element name="sourceLocation " type="xs:string"/>
      <xs:element name="DateTimeCreated " type="xs:date"/>
      <xs:element name="imageWidth " type="xs:int"/>
      <xs:element name="imageHieght " type="xs:int"/>
      <xs:element name="compression " type="compressionType "/>
      <!-- etc. -->
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="compressionType ">
    <xs:sequence>
      <xs:element name="compressionScheme " type="xs:byte"/>
      <xs:element name="compressionLevel " type="xs:byte"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

FIG. 3 – *XML Schema as a metadata grammar*

3.2 Active Rules

Although the general and preferred forms of active rules are ECA rules, other variations may occur. For example, the omission of the condition part leads to an Event-Action rule (EA-rule), where the condition is considered to be always true, and the omission of the event part leads to a Condition-Action rule (CA-rule), where the compiler generates the event definition (Machani, 1996). Most active rules defined in our integrating architecture are EA-rule.

Active rules in databases are employed for detecting changes of measurable units and detailed values in relational databases. Per contra, the embedded WSs of AXML documents are employed for retrieving directly up-to-date data from their sources. Table 1 compares actives rules in ADB versus our integrating architecture. Active rules are also proposed for analysis purposes in warehousing environments, namely analysis rules (Thalhammer et al., 2001).

In our architecture, active rules are associated with different sources of complex data, ETL services and the AXML repository, making them responsive to a variety of events. Events

TAB. 1 – *Comparison of active rules in ADB and in AXML integration*

	Active Databases	AXML Integration
Event	- Insert, update and delete statements - Create, delete and modify objects in OODB - Method invocation - Time-based events - Application defined events	- Modification events (ADD, ALTER and DROP data source) - Resource-structure events (DEFINE, UPDATE& DELETE entity| element| att.) - Temporal events (Absolute, Periodic and Relative events) - Explicit events - Request AXML documents
Condition	- Database predicate | queries	- COMPARE different schemas - CHECK logs for encountering events
Actions	- Sequence of insert| update| delete operations - Transaction statement	- NOTIFY a server with events - ENQUEUE events into the log - EXECUTE resource events - INVOKE ETL service | AXML WSs

range from modification of data resources' input schema (e.g., add new data resource, alter or drop existing data resource), to resource-specific events (e.g., defining new entities such as database tables, measurable elements or attributes). Thus, these events focus on altering the structure and the description of sources.

Temporal events are proposed in ADBs (Machani, 1996; Paton, 1999) and ADW (Thalhammer et al., 2001). Such events are suggested in our integrating architecture to detect changes in the structure of data sources (e.g., every one hour; check addition, modification, or deletion of entities, elements or attributes in input data sources). They are also suggested for maintenance purposes (e.g., every day at 20:00; ensure source input availability, every Sunday at 8:00; backup the AXML repository incrementally, after the insertion or deletion of any object; ensure the consistency of the integrity constraints, etc.). Another type of events is explicit events (e.g., querying embedded calls to WSs).

Recall that most active rules in the proposed architecture are EA-rules. However, conditions in our architecture might be specified as simple boolean expressions. For instance, compare the input schema of data sources with their latest state in order to detect changes in sources' description, and check the event log for encountering changes against their impact into the repository in order to ensure the consistency of the AXML repository.

Finally, there are many forms of actions in our architecture. They may notify a specific server with occurring changes in input data sources' description, which results in invoking the corresponding ETL services to integrate the changes. They may affect the AXML repository itself by invoking embedded WSs, when querying AXML documents in order to refresh them with up-to-date results.

3.3 Active ETL Dataflow

Integrating relevant complex data into the AXML repository in an automatic and updatable manner is not a trivial task. From a dataflow point of view, our architecture can be seen

as a bidirectional approach. In "forward direction", relevant complex data are extracted from sources, then transformed and loaded into the active XML repository via ETL services. Although this direction is the same than in classical data warehousing, there are some differences: utilizing ETL tasks as WSs to cope with the problems of data distribution and heterogeneity of data sources, and applying a form of intelligence on the work of ETL services due to the mechanism of ECA. In "backward direction", the AXML repository is refreshed by calling AXML WSs. The evaluation of WSs is managed by the AXML & ECA engine to call and execute specific ETL services or other third-party Web service.

4 Motivating Example

The main power of the proposed architecture is to integrate complex data relying on metadata and on an event-driven mechanism. To demonstrate this concept, there are several case studies including: healthcare, banking, geographic information systems, as well as systems for managing products, comparing prices, promotion and marketing.

Suppose we have system for marketing products, such as digital cameras and camcorders. The marketed products are supplied by different vendors. Camera or camcorders specifications are ranging from imaging (i.e., resolution, sensor, bit depth, file formats, file size, color spaces, image stabilization, crop factor), optics (i.e., lens, zoom, lens mount, focus type, focus range, accessory lens, filter mount), shooting controls (i.e., sensitivity, shutter speeds, metering, exposure modes, white balance, mirror lock-up, burst capability, self timer, interval recording, remote control), flash (i.e., built-in flash, effective flash range, external flash connection, max sync speed), memory (i.e., built-in memory, memory card type, still images per GB), A/V recording (i.e., video recording, video resolution, video clip length, audio resolution), viewfinder (i.e., viewfinder type, viewfinder coverage, display), connectivity (i.e., connectivity, system requirements, software requirements), environmental (i.e., operating temperature, Weatherproofing, durability), power (i.e., battery type, power adapter) to physical (i.e., dimensions, weight). Besides specifications, sample images captured by camera and sample clips recorded by camcorder are very important in order to review the product. Such heterogeneous types of data are needed to be unified into XML. The specifications of marketed products are integrated from different vendors into the marketing data warehouse (MDW). MDW communicates with those vendors either by accessing their Web servers or by accessing their own databases. Notice that each vendor may use different Web server platform and different database platform. Accessing different Web servers, databases and networks are suggested to be carried out utilizing Web services. Typically, MDW is supposed to be implemented as repository of AXML documents.

In order to increase the value of competition, most vendors offer promotions to their stock during various periods throughout the year. On hand, an event-driven mechanism can be applied effectively to feed the reduced prices into MDW immediately. Events can also be applied to add new camera item into MDW, updating or deleting a specific camera specifications from MDW. More broadly, events can be applied to add new vendor with its family of products or drop existing vendor with its family of products. On the other hand, Web services can be applied to compare prices of the same camera from different vendors, returning the cheapest one. Also, they can be applied for currency conversion onto unified format. Indeed, Web services can be applied to integrate product specifications into MDW.

Purchasing events, their timing, promotions, products and their vendors are queried and analyzed to: define the most frequently product purchased, find association rules between a specific product and a particular promotion, get the most vendor selling, and study if customers are most interesting with price, quality, or vendor due its credibility of after-sales services.

5 Implementation Issues

Different modules are implemented separately in order to validate our architecture, using mostly standard open-source and free software:

- Eclipse and Sun's Java JDK 6,
- Apache Tomcat 6 Server engine,
- Apache Axis2/Java, Java2WSDL, WSDL2Java and Apache Ant,
- JSP-based interface.
- eXist-db Open Source Native XML Database.

A prototype is implemented in two main phases. The first phase is to build, deploy and test integration services. The second phase is to develop simple interface using Web forms and JSP in order to access data sources, manage and invoke integration services. The existing Application Programming Interfaces (APIs) can facilitate integration tasks, either for reading from heterogeneous sources or writing into the target repository. For instance, when implementing *extraction services*, there are many helpful APIs for extracting the relevant data from heterogeneous data sources. Java Database Connectivity (JDBC) can be used for databases, I/O for flat files, Image I/O for images, sound and Java Media Framework (JMF) for audio, video and other time-based media, Java API for XML Processing (JAXP), XQuery API for Java (XQJ), etc. *Loading services* utilize different APIs in order to load and write transformed data into the AXML repository, such as Streaming API for XML (StAX), Writing API for XML (WAX), XML Writer, etc. AXML repository is implemented using eXist-db, an open source database management system built using XML technology. It stores XML data according to the XML data model and features efficient, index-based XQuery processing.

The developed services are deployed on a *Tomcat*[1] Web application server via *AXIS2*, which generates Web Service Definition Language (WSDL) files from Java classes or interfaces using Apache Java2WSDL tool. Then, the generated WSDL files are used to build the appropriate bindings for the Web services using Apache WSDL2Java tool under Eclipse environment. AXIS2[2] is the core engine for WSs, which has capability to add Web services interfaces to Web applications. It allows the interaction among distant machines over a networked environment. Then, WSs can be invoked from anywhere by specifying the complete Uniform Resource Locator (URL) of host containing AXIS2 (e.g., *http://88.186.240.140/axis2/services/...*).

Moreover, ECA rules are implemented as listener modules to detect changes in input schema of data sources' structure. They are implemented as an if-then structure to control rule conditions and actions. Rule actions often invoke specific WSs and activate when their condition is satisfied. However, data sources' contents can be detected and notified using third-party open source software.

[1] http://tomcat.apache.org/
[2] http://ws.apache.org/axis2/

6 Conclusions and Future Trends

In this paper, we surveyed the state of the art concerning XML warehousing and active warehousing. Then, we presented a general architecture for integrating complex data from heterogeneous and distributed data sources into an AXML repository. In this architecture, the integration tasks are expressed as WSs. The work of such WSs conforms to the work of ECA rules realizing some form of intelligence on integrating complex data. We integrate complex data into AXML documents for multiple reasons. XML has extensibility and flexibility features to represent, model and store complex data from heterogeneous data sources into a well-formed format. WSs cope with the problem of distribution and interaction over different networks. Moreover, AXML documents do not require storing all data explicitly, wherever their most frequently updatable data can be loaded on the fly via evaluating embedded WSs.

Recall that AXML warehouse implementation effort is not complete yet. We still face some difficulties for handling some sources of complex data, such as unstructured data sources, multimedia and dynamic Web sources. In the near future, we plan to give more attention for dealing with these sources. To this end, we may embed some external modules into the implementation of the architecture. Such modules should mine relevant data from unstructured sources and detect changes in dynamic Web pages. Moreover, incorporating the semantic meaning of complex data into the warehoused data is another issue. It will lead to a complex architecture exploiting data mining and semantic data retrieval techniques. User requirements are so important for successful data integration. Thus, these requirements should be taken into account and thoroughly defined before and during data integration. We also intend to give more attention to intelligent ETL for handling several challenges raised by integrating complex data. In addition, the absence of browser for reading AXML documents and the difficulty of processing such documents are other challenges. Finally, we aim at developing the AXML warehouse as a Web-based application, to help data stewards run and manage it entirely through a browser.

References

Abiteboul, S., O. Benjelloun, and T. Milo (2008). The active XML: an overview. In *VLDB Journal*, pp. 1019–1040.

Abiteboul, S., I. Manolescu, and S. Zoupanos (2007). Optimax: optimizing distributed continuous queries. *In Bases de Données Avancées (BDA'07)*.

Abiteboul, S., B. Nguyen, and G. Ruberg (2006). Building an active content warehouse. *In Processing and Managing Complex Data for Decision Support (Darmont and Boussaïd, eds.), Idea Group*.

Araque, F. (2003). Real-time data warehousing with temporal requirements. *15th Conference on Advanced Information Systems Engineering (CAiSE'03), Austria*.

Bailey, J., A. Poulovassilis, and P. T. Wood (2002). Analysis and optimisation of event-condition-action rules on XML. *Computer Networks 39*, 239–259.

Baril, X. and Z. Bellahsène (2003). Designing and managing an XML warehouse. *In XML Data Management: Native XML and XML-enabled Database Systems, Addison Wesley*, 455–473.

Beyer, K., D. Chamberlin, L. Colby, F. Özcan, H. Pirahesh, and Y. Xu (2005). Extending xquery for analytics. *Proceedings of International ACM Special Interest Group for the Management of Data (SIGMOD'05), Baltimore, USA*, 503–514.

Bonifati, A., S. Ceri, and S. Paraboschi (2000). Active rules for XML: A new paradigm for E-Services. In *Proceedings of TES Workshop (VLDB'00), Cairo, Egypt*.

Bonifati, A., S. Ceri, and S. Paraboschi (2001). Reactive services to XML repositories using active rules. In *Proceedings of World Wide Web(WWW'01)*.

Boussaid, O., R. Ben Messaoud, R. Choquet, and S. Anthoard (2006). X-warehousing: An XML-based approach for warehousing complex data. *10th East-European on Advances in Databases and Information Systems (ADBIS'06), Thessaloniki, Greece*, 39–54.

Bruckner, R. and A. Tjoa (2001). Managing time consistency for active data warehouse environments. In *Proceedings of Data Warehousing and Knowledge Discovery(DaWaK'01), Munich, Germany*, pp. 254–263.

Darmont, J., O. Boussaid, J. Ralaivao, and K. Aouiche (2005). An architecture framework for complex data warehouses. *7th International Conference on Enterprise Information Systems (ICEIS'05), Miami, USA*, 370–373.

Golfarelli, M., S. Rizzi, and B. Vrdoljak (2001). Data warehouse design from XML sources. *4th International Workshop on Data Warehousing and OLAP (DOLAP'01), Atlanta, USA*, 40–47.

Hummer, W., A. Bauer, and G. Harde (2003). Xcube: XML for data warehouses. *6th International Workshop on Data Warehousing and OLAP (DOLAP'03), New Orleans, USA,*, 33–40.

Inmon, W. (1996). *Building the Data Warehouse.* John Wiley & Sons.

Kimball, R. (1996). *The data warehouse toolkit*. John Wiley & Sons.

Machani, S. (1996). Events in an active object-relational database system. *Linköping Studies in Science and Technology, Master Thesis, Linköping University, Sweden*.

Mahboubi, H., M. Hachicha, and J. Darmont (2008). XML warehousing and OLAP. *Encyclopedia of Data Warehousing and Mining, 2nd Edition, IGI Publishing, USA*, 2109–2116.

Milo, T., S. Abiteboul, B. Anman, O. Benjelloun, and F. Ngoc (2003). Exchanging intentional XML data. In *Proceedings of International ACM Special Interest Group for the Management of Data (SIGMOD'03)*, pp. 289–300.

Nassis, V., R. Rajugan, T. Dillon, and J. Rahayu (2005). Conceptual and systematic design approach for XML document warehouses. *International Journal of Data Warehousing & Mining 1*(3), 63–86.

Paton, N. (1999). *Active Rules in Database Systems*. Springer, New York.

Pokorný, J. (2002). XML data warehouse: Modelling and querying. *5th International Baltic Conference (BalticDB&IS'02)*, 267–280.

Polyzotis, N., S. Skiadopoulos, P. Vassiliadis, A. Simitsis, and N. Frantzell (2007). Supporting streaming updates in an active data warehouse. *23rd International Conference Data Engineering(ICDE'07), Istanbul, Turkey*, 476–485.

Rajugan, R., E. Chang, and T. Dillon (2005). Conceptual design of an XML FACT repository

for dispersed XML document warehouses and XML marts. *5th International Conference on Computer and Information Technology (CIT'05), Shanghai, China*, 141–149.

Rekouts, M. (2005). Incorporating active rules processing into update execution in XML database systems. *16th International Workshop on Database and Expert Systems Applications(DEXA'05), Copenhagen, Denmark*.

Ruberge, G. and M. Mattoso (2008). XCraft: Boosting the performance of active XML materialization. *11th International Conference on Extending Database Technology (EDBT'08), Nantes, France*, 299–310.

Rusu, L., J. Rahayu, and D. Taniar (2005). A methodology for building XML data warehouses. *International Journal of Data Warehousing & Mining 1*(2), 67–92.

Thalhammer, T., M. Schrefl, and M. Mohania (2001). Data warehouses: Complementing OLAP with active rules. *Data and Knowledge Engineering 39*(3), 241–269.

Tho, M. N. and A. Tjoa (2003). Zero-latency data warehousing for heterogeneous data sources and continues data streams. In *Proceedings of 5th International Conference on Information and Web-based Applications Services (iiWAS'03), Jakarta, Indonesia*, pp. 55–64.

Vidal, V., F. Lemos, and F. Porto (2008). Towards automatic generation of AXML Web services for dynamic data integration. *3rd International Workshop on Database Technologies for Handling XML Information on the Web (DataX-EDBT'08), Nantes, France*, 43–50.

Vrdoljak, B., M. Banek, and S. Rizzi (2003). Designing Web warehouses from XML schemas. *5th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'03), Prague, Czech Repblic*, 89–98.

Xyleme, L. (2001). A dynamic warehouse for XML data of the Web. *International Database Engineering & Applications Symposium(IDEAS'01), Grenoble, France*, 3–7.

Résumé

L'entreposage de données n'est pas une tâche triviale, notamment lorsqu'il s'agit de grand volumes de données distribuées et hétérogènes. En outre, les systèmes traditionnels d'aide à la décision ne possèdent pas de capacités intelligentes pour l'intégration de ces données complexes. Par conséquent, nous proposons une approche pour l'aide à la décision intelligente basée sur l'entreposage Active XML. Nous exploitons XML comme langage pivot dans le but d'unifier, de modéliser et de stocker des données complexes. Par ailleurs, les services Web s'attaquent aux problèmes de distribution et l'interopérabilité des sources de données. Ils sont utilisés pour l'intégration des données complexes et réagissent avec des règles actives pour la réalisation d'ETL intelligent. Dans ce papier, nous nous concentrons sur la phase d'intégration et nous proposons une architecture d'intégration de données complexes au sein d'un référentiel de documents Active XML, basée sur les services Web et les règles orientées-événements. Enfin, nous avons mis au point un prototype de logiciel pour valider cette approche.