Active XML-based Web Data Integration

Rashed Salem. Jérôme Darmont. Omar Boussaïd

Université de Lyon (ERIC Lyon 2), 5 av. P. Mendés-France, 69500 Bron, France.

Tel.: +33 (0)4 78 77 31 54

Fax: +33 (0)4 78 77 23 75

E-mail: {first.last}@univ-lyon2.com

Abstract Today, the Web is the largest source of information worldwide. There is currently an increasing demand that decision-making applications such as Data Warehousing (DW) and Business Intelligence (BI) move onto the Web, especially in the cloud. Integrating data into the DW/BI applications is a critical and time-consuming task. To make better decisions in DW/BI applications, next generation data integration poses new requirements to data integration systems, over those posed by traditional data integration.

We propose in this paper a metadata-based, event-driven, and service-oriented framework for integrating real-time data autonomously. Our framework utilizes Web standards to integrate data from heterogeneous and distributed sources. It exploits the XML language to address data heterogeneity, Web services to tackle data distribution, and Active XML (AXML) to store integrated data. Our framework is also subscribed to web services for real-time change data capture. Moreover, beside logging different framework events into a specified repository for on-line analysis and reporting, we propose a novel Frequency XML-based Tree (FXT) structure for mining association rules from logged event streams using XQuery. Our framework is also incorporates active rules to automate and activate different integration services. Finally, we present a web application prototype as a proof of concept.

Keywords Real-time data integration, Web data, integration services, active rules, event stream mining.

1 Introduction

Business intelligence (BI) is a set of applications and technologies for gathering, storing, analyzing, and providing access to data in order to help enterprise users make better business decisions. BI applications include the activities of decision-support systems, querying and reporting, online analytical processing (OLAP), statistical analysis, forecasting, and data mining. BI applications typically use data stored in a data warehouse or data marts extracted from the data warehouse. A data warehouse is a central repository for all or significant parts of the data that an enterprise's various business systems collect. It provides the base to perform refined reporting and analytics. Thus, data warehousing (DW) is a vital aspect of BI, and both DW/BI today play an important role in decision making. DW processes include integrating, storing and analyzing business data (Figure 1). Data integration is a crucial task for DW/BI applications. It consumes a large fraction of the effort (70% by some estimators). Data integration systems consolidate data from various data sources into a target warehouse, performing extraction-transformation-loading (ETL) tasks.

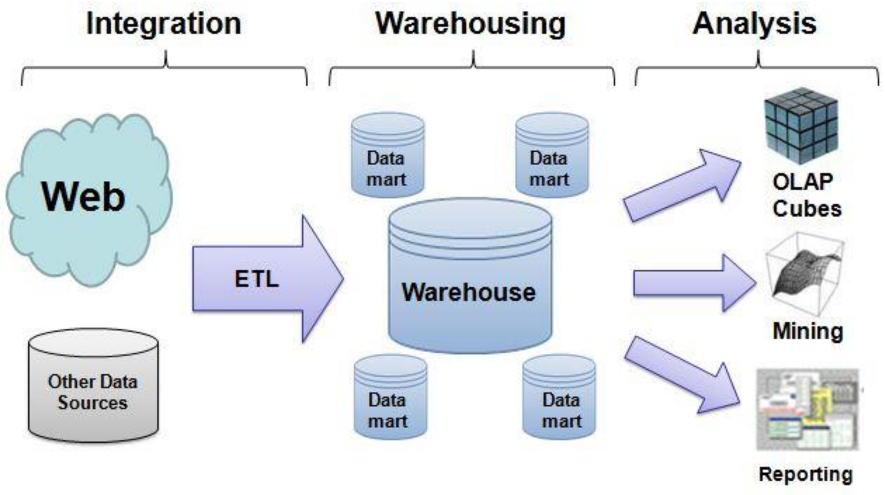

Fig 1 Data warehousing/business intelligence process

Nowadays, the Web is the world's largest source of information. The Web is very rich with heterogeneous and distributed data (e.g., semi-structured/unstructured data, review chats, e-mails, images, videos, voice call transcripts, feedback and surveys). Other valuable data types include relational databases, XML documents, flat files, online transaction records, external data feeds, sensor and streaming data. We term such heterogeneous and distributed data *complex data*. Data may be qualified as complex if they are: diversely structured, represented in various formats and models, originating from several different sources, described through different perspectives and/or changing in terms of definition or value over time (Darmont et al. 2005). Therefore, modern DW/BI applications currently move onto the Web, e.g., to integrate Web (complex) data to make better decision, or benefit from cloud computing facilities. In the cloud, BI is accessible via any browser; there is no need to buy any software nor hardware. With BI software running in the cloud, data integration must be carried out with back-end systems, far from end-users.

By studying the literature, we observed that data flow in traditional data integration systems is one-way. The dominant way of populating data warehouses and data marts is to operate ETL processes offline, in batch mode, usually at regular interval of downtime (e.g., at the end of a day, week, or month). However, demand for fresh data in either DW or BI is strongly desired in many businesses. For example, enterprises need to integrate real-time changing customer needs, customer click stream data, up-to-minute inventory and pricing data. Real-time acquisition of radar stream and weather data is necessary for forecasting. E-banks need to integrate real-time data to react against fraudulent transactions. Therefore, “real-time” or “near real-time” data should be integrated to minimize latency delays between BI systems and business operations for better decision making. Furthermore, traditional data integration systems are passive in nature, with data integration tasks conforming to a static scheduling plan. However, today’s DW/BI applications demand to detect and react to different events in an autonomic way.

Hence, we propose in this paper to integrate Web (complex) data in a near real-time, autonomic and active way, using Web standards (XML, Web services and related technologies). Rather than dealing with Web data heterogeneity and distribution, we also deal with two main Web data integration issues, which are 1) integrating Web data in near real-time, and 2) integrating Web data in an *autonomous* and *active* way (i.e., with minimum user intervention). Firstly, near real-time data is guaranteed to be integrated via maximizing the role of integration services to integrate Web data. For instance, integration services can be embedded in so-called Active XML (AXML) documents to integrate real-time data *on the fly* upon request. Other services can be employed to change data capture (CDC), then the integration system can be subscribed to CDC services to be notified with only data near real-time data changes. Secondly, the *actively* and *autonomously* integrating data can be achieved by mining logged events and enriching the system with some active (Event-Condition-Action, or ECA) rules. Therefore, beside proposing to warehouse logged events of the data integration system for further analysis and mining, we also propose a novel Frequency XML-based Tree (FXT) structure for mining association rules from the framework’s logged event streams. The discovered rules can then be employed to maintain, self-configure and automate data integration tasks. Our framework incorporates active rules to automatically activate integration services according to encountered events. Finally, integration services query and interact with active metadata repository to maintain different integration system activities.

The rest of this paper is organized as follows. In section 2, some of the related works about data integration, Web-warehousing, real-time and active warehousing, AXML and active XML rules are reviewed. Section 3 introduces our framework for real-time Web (complex) data integration and discusses the role of metadata in data integration. Data integration using Web services is detailed in section 4. Mining logged events and incorporating active rules to reactive data integration tasks are discussed in section 5. Implementation issues are presented in section 6. Finally, we conclude and discuss future trends in section 7.

2 Related Work

In this section, we review and discuss the literature related to data integration approaches, Web warehousing, active XML and active rules, and real-time and active data warehousing.

2.1 Data Integration

Over the past decade, many data integration approaches have been proposed, most of them being surveyed in (Halevy et al. 2006; Ziegler & Dittrich 2004). We focus here on data integration approaches using web services and data integration approaches supplied with feedback and active features. There are two main families of approaches to integrate data: virtual views and materialized views (warehousing).

Virtual Data Integration

In a virtual view approach, data are accessed from the sources on demand when a user submits a query to the information system. Examples of integration systems that use the virtual view approach are federated database systems and mediated systems. A Federated Database System (FDBS) is a collection of autonomous database components that cooperate for sharing data (Sheth & Larson 1990). Mediated systems integrate data from heterogeneous data sources by providing a virtual view of all data sources (e.g., as in the TSIMMIS (Chawathe et al. 1994) and Ariadne (Knoblock et al. 2001) projects). They handle not only operational databases, but also legacy systems, web sources, etc. Mediated systems provide the single mediated schema to the user, and users pose their queries w.r.t. this schema.

Moreover, XML and Web services are employed for integrating web data in federated systems (Zhu et al. 2004) , and in peer-to-peer (P2P) and mediated systems (e.g., as in the AXML project (Abiteboul et al. 2002; Abiteboul et al. 2008). AXML 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 Web services (WSs). When one of these calls is invoked, its result is returned to enrich the original document.

Data Warehousing

In a materialized view approach, relevant data are filtered from data sources and pre-stored (materialized) into a repository (namely a warehouse) that can be later queried by users (Inmon 1996; Kimball et al. 1998). Although the materialized view approach provides a unified view of relevant data similar to a virtual view, it physically stores these data in a data warehouse. This approach is mainly designed for decision-making purposes and supports complex query operations (Boussaïd et al. 2006; Darmont et al. 2005). Compared to virtual data integration approaches, classical warehousing approaches lack of data freshness when dealing with data sources that may update their contents very frequently. They also cannot handle a large number of heterogeneous and distributed data sources that need to store their data in a central repository and keep them always up-to-date.

On the other hand, there are a variety of approaches proposed for XML data warehousing. The main purpose of such approaches is to enable a native XML storage of the warehouse, and allow querying it with XML query languages, mainly XQuery. Several researchers address the problem of designing and building XML warehouses. For instance, the snowflake schema is adapted with explicit dimension hierarchies (Pokorný 2002). Mahboubi et al. (2008) also propose an architecture model to translate the classical constellation schema into XML. Furthermore, Hümmer 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. Boussaïd et al. (2006) propose X-Warehousing for warehousing complex data and preparing XML documents for future analysis. Baril & 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 Web Warehousing

The problem of integrating Web data is not trivial, mainly because data sources are dynamic and heterogeneous. In this context, some researchers focused on the construction of dynamic warehouse for XML (Xyleme 2001) or Web documents (Whoweda 2011). In (Xyleme 2001), a dynamic warehouse is built for massive volume of XML data from the Web, several issues have been addressed by its authors such as: *Efficient storage* for huge quantities of XML data over the Web, considering the repository for efficient updateable storage of XML data; *query processing* with indexing at the element level for such a heavy load of pages, by implementing a complex algebraic model, named *PatternScan* that captures so-called tree queries; *data acquisition strategies* to build the repository and keep it up-to-date, by *crawling* the Web in search of XML data; *change control* with services such as query subscription; and finally, *semantic data integration* to free users from having to deal with many specific DTDs when expressing queries. The Whoweda (Warehouse of Web Data) project aims to design and implement a warehouse for relevant data extracted from the Web (Whoweda 2011). This project is mainly focused on the definition of a formal data model and an algebra to represent and manage Web documents, their physical storage, and change detection.

Golfarelli et al. (2001) propose a semi-automatic approach for building the conceptual schema for a data mart starting from XML data sources. The authors showed how the design of the data mart can be carried out starting directly from an XML source, and how the semi-structured nature of the source increases the level of uncertainty on the structure of data as compared to structured sources such as database schemas. Their approach was described by reference to the case of sources constrained by a DTD using sub-elements, but it can be adopted equivalently when XML schemes are considered.

Vrdoljak et al. (2003) propose a semi-automated methodology for designing Web warehouses from XML sources modeled by XML Schemes. In their methodology, design is carried out by first creating a schema graph, then navigating its arcs in order to derive a correct multidimensional representation. Their approach was implemented through a prototype that reads an XML schema and outputs the logical star schema of the warehouse.

Nørvåg (2002) discusses some challenges of temporal XML data warehouses, in order to query historical document versions, or query changes between document versions. Some of these challenges relate to consistency and the problems that are difficult to solve automatically in a temporal XML-DW, such as transaction time, valid time, time approximation, inconsistency between versions, summarizing data, and vacuuming to control the deletion of old document versions.

Abiteboul et al. (2006) utilize Active XML and Web service technologies to present an approach for building and maintaining domain specific content warehouses, which differs from classical data warehouses by managing “content” and not only numerical data. Vidal et al. (2008) propose a framework for generating AXML Web services for materializing the dynamic content of Web sites whose primary goal is publishing and integrating data stored in multiple data sources. In this framework, the dynamic content of a Web page is defined as a (virtual) view, and the view is specified with the help of a set of correspondence assertions, which specify the semantic mapping between the XML view schema and the base sources schema.

Kimball & Merz (2000) introduce the marriage of data warehouse and the Web to build Web-enabled data warehouse (or Webhouse). They address the problem of bringing data warehouse to the Web in order to deliver Web information not only to managers, executives, business analysts, and other higher level employees, but also to customers, suppliers and other business partners. They also discuss the importance of bringing the Web to data warehouse.

Wu (2006) propose to build a *virtual* integration system over *Deep Web* sources, the sources that are only accessible through their query interfaces. Their proposed integration system provides a uniform access to the sources, thus freeing the users from the details of the individual sources.

2.3 Active XML and Active Rules for 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. Ruberg & 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. 2008).

Active rules are widely known in active databases (Machani 1996; Paton 1999). They are set of rules follow Event-Condition-Action (ECA) paradigm, 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. Active rules are also proposed for XML. Bonifati et al. (2000, 2002) propose an active extension 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 and Real-Time Warehousing

To improve data freshness and to realize real-time decision support systems, Tho & Tjoa (2003) propose a framework for building Zero-Latency Data Warehouse (ZLDW). They capture and load data from heterogeneous data sources using a continuous data integration technique. Moreover, they combine their integration technique with an Active Data Warehousing (ADW) approach (Thalhammer et al. 2001). ADW exploits active rules to achieve auto-decision-making by minimizing user intervention in processing a series of complex analysis tasks, so-called *analysis rules.*

2.5 Discussion and Positioning

Although our approach is designed for warehousing systems, it is supplied with virtual approach features. Particularly, AXML documents consist of two types of XML nodes. The first set of nodes is defined and stored explicitly, but the second set of nodes is defined implicitly by calling services to integrate the data on the fly, virtually. It has virtual approach advantages of data integration from a large number of heterogeneous and distributed data sources that are likely to be updated frequently, due to using AXML, but there are no predefined or expected user queries. Calling information of AXML services may or may not occupy larger storage than their results, but certainly they can be reused infinitely to refresh the repository. Furthermore, our approach gains advantage from the data warehousing approach by storing relevant data into a unified repository. The target repository contains not only the integrated data but also calls to integration services. Accordingly, better performance can be achieved from saving response time when querying data, especially if data sources are physically located far from the integration system.

Compared to existing Web warehousing approaches, our approach aim at integrating not only XML data sources, like in (Xyleme 2001; Golfarelli et al. 2001; Vrdoljak et al. 2003; Nørvåg 2002) , but also other types of Web data sources (e.g., structured, semi-structured, image, video, text, etc.), and unifying them into XML format. Moreover, our approach aims at integrating Web data in an innovative real-time and autonomous manner comparing to (Abiteboul et al. 2006; Vidal et al. 2008; Wu 2006).

Furthermore, if we compare our approach to related works, i.e., ZLDW and ADW, ZLDW must update data more frequently to improve data freshness using push/pull technology via a message queuing system. The ZLDW and ADW approaches use traditional batch data integration to integrate data that do not need to be continuously updated and integrated. There is also no mention on how to handle heterogeneity and distributed issues in data sources. These approaches use active rules for automating routine analysis tasks, but feedback decisions are always taken by analysts. Our integration approach automates integration tasks via mining and analyzing logged events that are incorporated with active rules. Therefore, only initial or trained integration tasks need to be integrated by user via a Graphical User Interface (GUI).

3 AXML-based Data Integration Framework

Data integration is the process of consolidating data from heterogeneous and distributed sources into a unified repository as in data warehousing, or providing the user with a unified view of these data as in virtual data integration. We thus present in this section a metadata-based, event-driven and service-oriented framework for realizing near real-time and active data integration based on Web standards (Fig. 2). Integrating Web complex data for DW/BI poses different and new requirements to data integration technologies, over those posed by conventional data integration systems (Kimball et al. 1998; Inmon 1996). Therefore, more than one tool is required for combining, transforming, and organizing data into a common syntax. Today Web standards (e.g., XML, Web services and related technologies) can help enterprises to integrate such complex data. XML is indeed the *de facto* standard for representing, exchanging, and modeling semi-structured data. Thus, we exploit XML as a pivot language for standardizing data into a unified format. XML is also utilized for modeling and storing complex data. Moreover, Web services and *Software as a Service* (SaaS) can solve the data distribution and interoperability problems by exchanging data between different applications and different platforms. Indeed, a service provides transparent access to a variety of relevant data sources as if they were a single source. Thus, we employ metadata-based services to integrate data from distributed data sources. XML and services have dramatically changed distributed data management by overcoming the heterogeneity and distribution issues and implying some form of dynamicity. Embedding calls to services into XML data results in so-called *Active XML* (AXML). 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). Using XML and Web services standards also provide a less expensive and more efficient data integration infrastructure, typically by leveraging services to extract and detect changed data, providing transformation services to unify data, as well as loading services to move data from sources to targets (section 4). Integrated data are targeted to a native-XML based repository, namely an AXML repository. While integration services integrate data from sources to target, description of different framework activities are logged into event repository. Such events are mined on-line in order to extract interesting knowledge helps in self-maintaining, self-management and automating execution of integration services, specifically when incorporating the framework with active rules paradigm (section 5). Beside introducing our framework, we discuss in this section the role of metadata in the data integration framework.

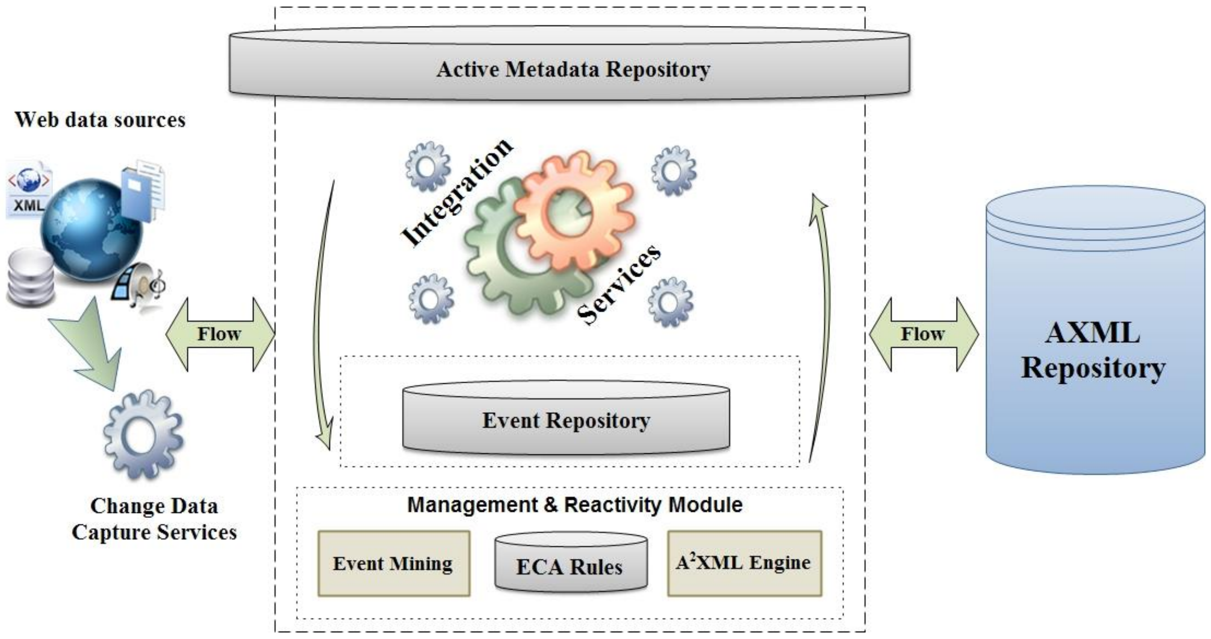

Fig. 2 AXML data integration framework

3.1 Active Metadata Management

Metadata, or data about framework's data, are a critical element in effective data integration. Thus, there is no data integration, data warehousing or business intelligence application without metadata repository. Metadata have a vital role to maintain data integration. There is now a trend in data integration toward metadata taking on a more active role. For example, specifying source-to-target mappings (under the form of metadata) serves as the basis for automating extraction and transformation tasks. The better the metadata, the better and easier the administration of the data integration environment (William Inmon 2008). Metadata are classified as either business or technical metadata (Inmon 1996; Kimball et al. 1998). Technical metadata are useful for system administrators to maintain and manage the system, and business data are useful for end-users when using the system.

Active Metadata Repository

There are two main types of metadata repositories: active repositories and passive repositories (William Inmon 2008). Metadata from an active repository interact in an ongoing manner during development, maintenance, and/or query activities of the system. On the other hand, metadata from a passive repository do not interact in any direct way. For example, source-to-target mapping can be queried continuously to determine which sources are mapped to integration services and/or targets. An active metadata repository should be always up-to-date with changes, unlike a passive metadata repository. Therefore, changes need to be reflected continuously in an active data repository to keep up with the typical evolution and maintenance of the data integration system. Figure 3 shows some examples of metadata for different framework components.

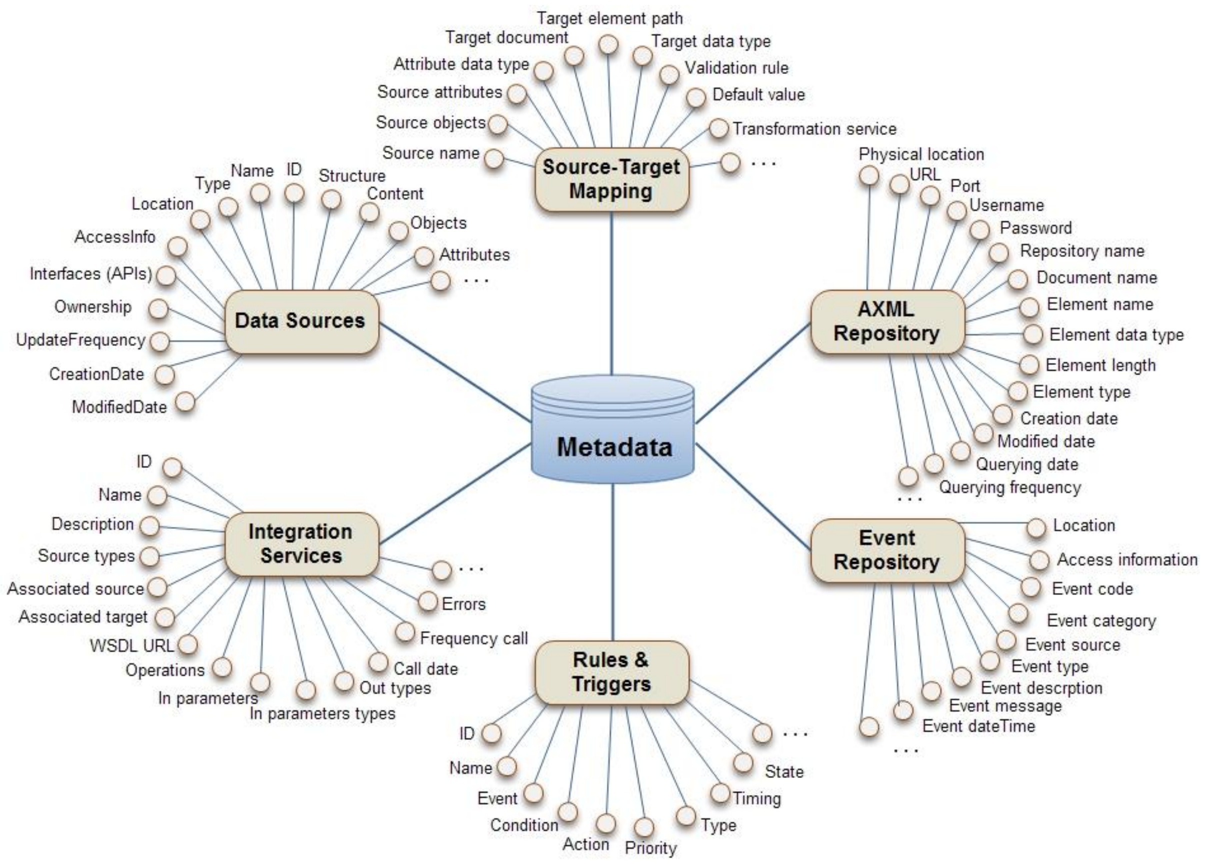

Fig. 3 Metadata samples of our data integration framework

Integration schema evolution

The integration schema specifies the structure of relevant data sources, the structure of target documents and lists integration services. Hence, the integration schema is considered as a catalog of input sources, target structures and service descriptions. Because there is an explosive growth of information and new data sources, particularly in the Web arena, these new data or changes of data sources should be reflected in to the data integration schema. However, only relevant structures are stored into the data integration schema. This schema can be continuously queried during system development, for instance to validate whether data source changes are relevant or not. The data integration schema can be updated either explicitly (i.e., by user when adding/modifying a specified data source), or implicitly (i.e., by applying Web service to notify the integration system with data source structure changes).

3.2 Data Integration Metamodel

In order to illustrate associations between framework entities, we present our complex data integration metamodel in figure 4. Classes in this metamodel represent metadata such as entity, attribute, table, column, and index. The corresponding metamodel describes relationships between the main entities of our complex data integration framework such as data sources, integration services, AXML document, and logged events. The "Complex_Source" class contains metadata descriptions of complex data sources. Due to handling several types of data sources, each data source is of a specific source type (i.e., relational database, XML, temporal, text, image, etc.). A data source is composed of one or more objects, where each object is composed of one or more attributes. The "Object" class describes the metadata of objects. Object is an optional entity and may not be available for non-relational data sources. Finally, the "Attribute" class describes the metadata of attributes.

Data sources, their objects, and their attributes are mapped with their associated integration services. The "Integration-Service" class contains metadata description of data integration services. Moreover, integration services are mapped with their target in AXML documents, explicit elements, and their implicit elements. Integration services are composed of several operations, each of which has one or more several parameters. Metadata description of service operations and operation parameters are specified in "Operation" and "Parameter" classes, respectively. The "AXML-Document" class contains the description of AXML documents' metadata. Such documents are composed of explicit elements and implicit elements. The

"Explicit-Element" and "Implicit_Element" classes describe metadata of explicit and implicit elements, respectively.

The "Event" class describes the logged events raised due to defining data sources, objects, fields, executing integration services, and querying AXML document. Although there are several types of events, one single event entity is specified for simplicity. Logged events are exclusive; each event type has its own descriptions. Events, conditions, and actions are mapped together to form active rules. The "Event", "Condition", and "Action" classes contain metadata description of events, conditions, and actions, respectively. Actions are always associated with integration services.

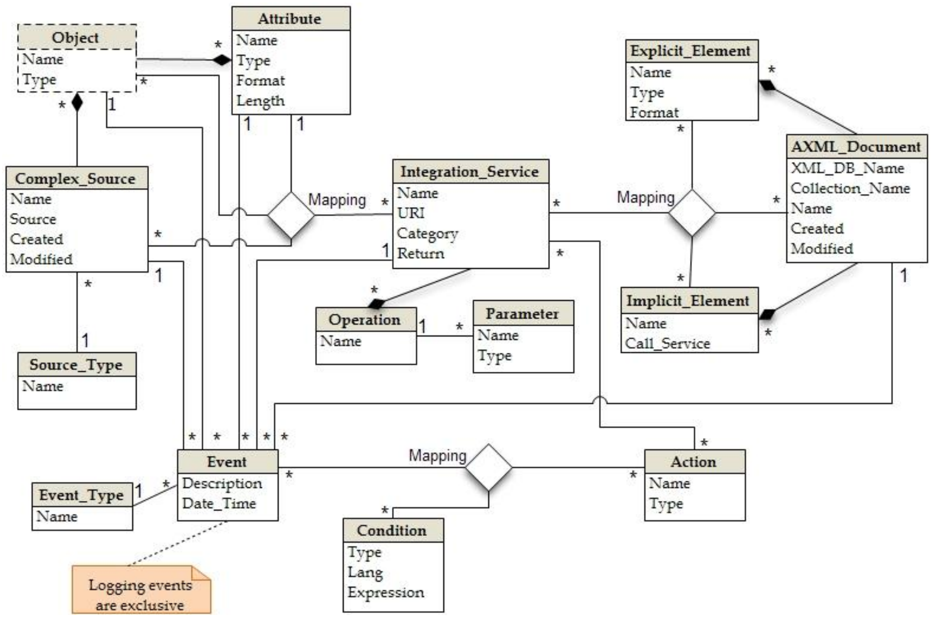

Fig. 4 Metamodel of main objects in our data integration framework

4 Service-oriented, Near Real-Time Integration

The Web has become a common platform for the delivery of business applications. Specifically Web service technologies and Software-as-a-Service (SaaS) enable the development of complex business applications on the internet/in the cloud by integrating Web services from different providers. Web services are self-contained, self-describing modular applications that can be published, located, and invoked across the Web. Integration services are set of Web services designed for integrating data from heterogeneous and distributed data sources. Integration services are based on a metadata-driven approach for integrating relevant data. The metadata-driven approach allows users to specify "what" data need to be integrated, without regarding "how" to integrate them. Due to the heterogeneity of data sources, integration services can utilize several types of interfaces in order to extract data, transport and store integrated data.

In this section, we discuss how integration services play an important role to integrate real-time data. Embedded AXML services can be invoked at any time to integrate real-time data *on the fly* upon request. Change detection services can be employed to CDC. CDC is a technology that enables to detect, capture and move only data changes in real-time, from data sources to integration systems. Using the subscription/notification paradigm, data integration systems can be subscribed to CDC services to be notified with only real-time changes data changes.

4.1 Data Integration Services

Although data integration includes several processes, ETL are still the most important services when integrating data. In addition to extraction, transformation, and loading services for performing ETL tasks, our data integration framework includes other services as follows.

- **Change detection services:** Change detection services are set of services that monitor relevant data sources for changes. We distinguish between two types of event changes, namely "structure changes" and "content changes". Structure changes are changes related to the structure of data sources. Content changes are changes related to the contents of data sources. Notice that not all encountered changes at data sources are relevant. Thus, change detection services can query the integration schema to check the relevance of data changes. Only relevant changes are taken into consideration to incrementally update the integrated data.
- **Refreshing services:** Refreshing services are set of services for incrementally update the integrated data with changed data detected by change detection services.
- **Logging services:** Logging services are set of services that log description of different activities into the event repository.
- **Mining services:** Mining services are set of services that extract interesting knowledge from logged event streams.
- **Querying services:** Querying services are set of services that query either metadata by other cooperated integration services or query integrated data by the business user.
- **Reactive services:** Reactive services are set of services that apply ECA paradigm to respond to encountered events with the appropriated actions.

- **Utility services:** Utility services include other services to maintain the system such as tuning service, configuring services, security services, scheduling services, monitoring services, error handler services, etc.

4.2 Front vs. Back Change Detection Service

The common methods for capturing real-time data changes include applying triggers in DBMS, adding timestamp attributes to database records, scanning log files of transactions, and/or monitoring transactions and then targeting changes to a specific message queues. These methods run in the back-end of data sources, and can be applied as services. Hence, we term these services as *back change detection services*.

In a different way, since our framework integrates data from Web data sources, we can assume that data source modifications can be conducted via a set of services (e.g., insert, update, or delete contents of Web data sources). The same modifiable services can be subscribed by multiple consumers (i.e., data integration services). These services then notify the consumers, in real-time, with only relevant data changes according to the subscription policy. We call this category of services *front change detection service.*

4.3 Integration Services vs. Data Services

We distinguish between two different categories of services: integration services and data services. Integration services, discussed in section (4.1), are designed to integrate data and manage the data integration system. Data services are a set of Web services that help share the provider ready data with the consumers. Nowadays, an increasing amount of enterprises implement their core business and outsource other application services over the Internet. Thus, these data services can be considered as data sources, and we term them as *external services*. Such external services have a direct access to their providers' underlying data. Some common examples of external services are currency exchange, products on stock, news streams, weather reports, and stock quote reports.

4.4 Integration Services Composition

With the complexity of business needs promised to be conducted over the Internet, Web services cannot satisfy all service requests individually. Therefore, there is an ever increasing demand to combine Web services together into a composite Web service to fulfill complex personal and business needs. A business typical example of a service request is to book room in a specified hotel through a Web site. It can be fulfilled by a composite service that integrates internal and external services such as checking room availability within a given period of time, credit card checking, payment and sending confirmation email to the customer. Similarly, the data integration process is complex and requires service composition to be conducted. For example, extracting operation names of an external Web service from the WSDL URL can be fulfilled by a composite service to get service ports, which also requires specifying service name and namespace. In figure 5(a) and 5(b), we show how AXML can be used as composition language in a data integration environment. AXML has first been introduced as composition language (Abiteboul et al. 2008), seen as equivalent to a subset of BPEL (Business Process Execution Language)[1]. An AXML document is an XML document specifying which services to call, how to build their input messages, and how the calls should be ordered.

```
<sc service="ExtractExternalServiceOperations">
        <input-param>$wsdlURL</input-param>
</sc>
```

(a)

```
<sc service="ExtractExternalServiceOperations">
        <input-param>
                <sc service="GetServicePort">
                        <input-param>
                                <sc service="GetServiceNameAndNameSpace">
                                        <input-param>$wsdlURL</input-param>
                                </sc>
                        </input-param>
                </sc>
        </input-param>
</sc>
```

(b)

Fig. 5 (a) de-composite service, and (b) composite service using AXML language

5 Event-driven Reactive Integration

Integrating data in an autonomous and reactive manner is the main problem that we address in this paper. We provide a solution by mining logged events on-line, and enriching the integration system with active rules to activate integration services when detecting specified events.

[1] http://www.ibm.com/developerworks/library/specification/ws-bpel/

5.1 Mining logged events

Data integration systems log a large amount of events as data streams every day into either simple text or XML format. We are particularly interested in events logged in XML format. Logged events are essential to understand the activities of integration systems. Our integration system logs events from its different modules, e.g., events related to detecting changes in data sources, executing integration services, and querying the AXML repository. Discovering interesting knowledge from logged events, which can be performed by data mining techniques, helps understand relationships among events and predict upcoming events. Therefore, such knowledge can be employed to self-maintain and activate the workflow behavior of these systems, autonomously.

Since events are logged continuously at high speed, in real-time, in unbounded amount, and with changing data distributions, events can be considered as streams. An *event stream* is defined as sequence of real-time, continuous and ordered events. Event stream mining is more difficult than mining traditional static data (Gaber et al. 2005; Jiang & Gruenwald 2006; Yu & Chi 2009). A single-pass over events is required. Event streams need to be processed incrementally as fast as possible. Processing speed should be faster than event arrival rate. Logging events in XML format poses another difficulty for mining events, due to the flexible, irregular, and semi-structured nature of XML data (Feng & Dillon 2004; Zhao et al. 2006).

Actually, logged events include much descriptive information about each activity (e.g., event identification, event occurring time, event source, event description, event category, etc.). The more we log event information, the more we can discover interesting knowledge. In order to apply the algorithm we propose for mining frequent events and association rules, we need to pre-process logged events and organize them into transactions as illustrated in the figure 6:

```
<transaction id="1" time="2011-01-12 09:16:00">
        <eventID>A</eventID><eventID>B</eventID><eventID>C</eventID><eventID>D</eventID>
</transaction>
<transaction id="2" time="2011-01-12 09:16:20">
        <eventID>C</eventID><eventID>E</eventID>
</transaction>
<transaction id="3" time="2011-01-12 09:16:40">
        <eventID>B</eventID><eventID>C</eventID>
</transaction>
<transaction id="4" time="2011-01-12 09:17:00">
        <eventID>C</eventID><eventID>D</eventID><eventID>E</eventID>
```

```
</transaction>
<transaction id="5" time="2011-01-12 09:17:20">
        <eventID>B</eventID><eventID>C</eventID><eventID>D</eventID>
</transaction>
<transaction id="6" time="2011-01-12 09:17:40">
        <eventID>A</eventID><eventID>C</eventID><eventID>E</eventID>
</transaction>
<transaction id="7" time="2011-01-12 09:18:00">
        <eventID>A</eventID><eventID>B</eventID><eventID>D</eventID>
</transaction>
<transaction id="8" time="2011-01-12 09:18:20">
        <eventID>E</eventID><eventID>F</eventID>
</transaction>
```

Fig. 6 Sample event transactions

Each transaction has an ID, an occurring time, and a set of items that represent event identifications. The set of events is logged in a window-size of time (time + window) are represented as a transaction. The corresponding format of transactions can be obtained or transformed using the XSLT language applied to original logged events. The most important thing to our algorithm is to define the listing of events of each transaction, which should be sorted alphabetically for performance purposes.

Frequency XML-based Tree (FXT)

We propose a single-pass algorithm for mining association rules from logged event streams. The algorithm firstly constructs a novel tree structure, called Frequency XML-based Tree (FXT), and then allows querying the tree using XQuery to discover frequent events and association rules. FXT nodes, except the root node, consist of two entries: *event ID* and *counter*, where *event ID* registers which event this node represents (i.e., I_i) and *counter* registers the number of transactions represented by the portion of the path reaching this node (i.e., N_i or $N_{m|\_|j}$). As illustrated in figure 7, the FXT is composed of three main levels of nodes. Firstly, the *Root node* represents the total number of logged transactions (N_{trans}). Secondly, *Breadth nodes* refer to all root's child nodes. It represents the count of each event that appeared in any logged transaction. Thirdly, the *Depth nodes* refer to all root's grandchildren nodes. It represents a relative or conditional count of a specific event given other related events. Depth nodes are represented as a set of paths, each path corresponding to specific transactions' event-sets. In figure 7, the dashed line annotated by double slashes "//"

means that there is zero or more in-between nodes in a specific depth path. Figure 8 shows the FXT constructed from sample transactions given in section 5.1. More details on how the algorithm constructs the FXT, and the performance study of the algorithm are discussed in (Salem et al. 2011).

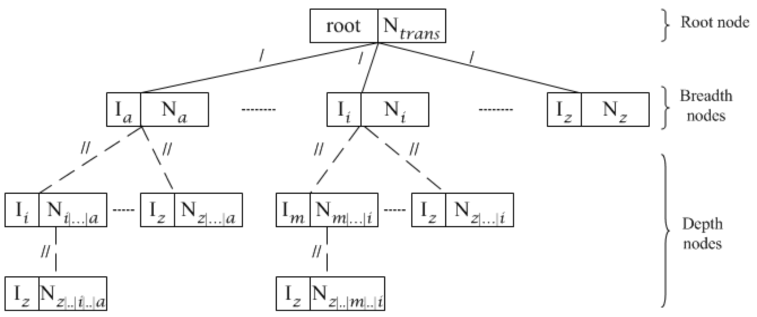

Fig. 7 Frequency XML-based Tree (FXT) structure

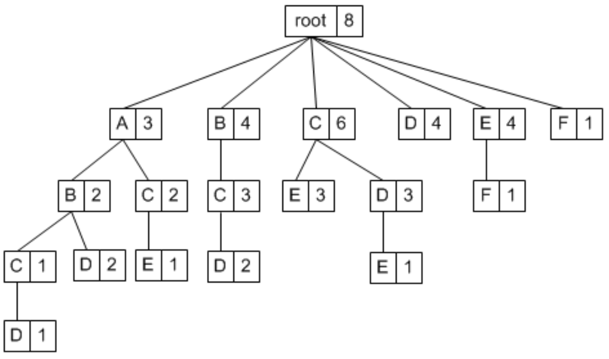

Fig. 8 Sample FXT

Finally, the constructed FXT in XML format is as featured in figure 9.

```
<? xml version="1.0" encoding="UTF-8"?>
<root counter = "8">
  <A counter = "3">
    <B counter = "2">
      <C counter = "1">
        <D counter = "1"/>
      </C>
      <D counter = "2"/>
    </B>
    <C counter = "2">
      <E counter = "1"/>
    </C>
  </A>
  <B counter = "3">
    <C counter = "3">
      <D counter = "2"/>
    </C>
  </B>
  <C counter = "6">
    <E counter = "3"/>
    <D counter = "3">
      <E counter = "1"/>
    </D>
  </C>
  <D counter = "4"/>
  <E counter = "4"/>
    <F counter = "1"/>
  </E>
  <F counter = "1"/>
</root>
```

Fig. 9 The constructed FXT document

Discovering frequent events and association rules

The main objective of constructing the FXT is to query frequent events and association rules easily using the XQuery language[2]. Frequent events are queried by traversing the FXT from breadth nodes to specific nodes (portion of paths), or to leaf nodes (complete paths). Frequent events are filtered using a statistical measure called *support*. Support measures the proportion of transactions that contains a specific event (or event-set). A frequent event-set is an event-set whose support is greater than some user-specified minimum support. Frequent event-sets satisfy the Apriori property, which states that if a given portion of path does not satisfy minimum support, then neither will any of its descendants (Agrawal & Srikant 1994). Support of frequent events can be obtained by querying the counter attribute value of the events at breadth nodes, and then divide this value by the counter of total number of transactions N_{trans}. Moreover, support of frequent event-sets can be obtained by querying the counter attribute value of the last ordering event of interesting portion of path, or the leaf event of the interesting path at depth nodes, and then divide this value by N_{trans}. Formula for retrieving the support of events and event-set from the FXT follow.

$$Support(A) = \frac{count(A)}{N_{trans}} = \frac{root/A/@counter}{root/@counter}$$

$$Support(B,C,D) = \frac{count(B \cup C \cup D)}{N_{trans}} = \frac{root/B/C/D@counter}{root/@counter}$$

[2] http://www.w3.org/TR/xquery/

Example: The following example introduces a function for generating frequent event-sets according to a specific minimum support from the FXT that constructed by our algorithm (figure 10).

```
declare variable $input := doc("tree.xml")/root;
declare variable $rootCounter := $input/@counter;

declare function local:getFrequentItemsets($parent as xs:string, $event as element(*, xs:untyped),
$minSupport as xs:decimal) {

let $path := concat($parent,'/', name($event))
where $event/@counter div $rootCounter>=$minSupport
return
 (<frequent eventItemset="{$path}"  support="{$event/@counter div $rootCounter}"/>,
  for $child in $element/*
  return
    local:getFrequentItemsets ($path, $child,  $minSupport))               };

(: call the function :)
for $child in $input/*
return
 local:getFrequentItemsets("", $child, 0.25)
```

Fig. 10 XQuery function for mining frequent event-sets

The result of calling this function is as follows (figure 11):

```
<frequent eventItemset="/A" support="0.375"/>
<frequent eventItemset="/A/B" support="0.25"/>
<frequent eventItemset="/A/B/D" support="0.25"/>
<frequent eventItemset="/A/C" support="0.25"/>
<frequent eventItemset="/B" support="0.5"/>
<frequent eventItemset="/B/C" support="0.375"/>
<frequent eventItemset="/B/C/D" support="0.25"/>
<frequent eventItemset="/C" support="0.75"/>
<frequent eventItemset="/C/E" support="0.375"/>
<frequent eventItemset="/C/D" support="0.375"/>
<frequent eventItemset="/D" support="0.5"/>
<frequent eventItemset="/E" support="0.5"/>
```

Fig. 10 Result of XQuery function for mining frequent event-sets

Association rules have been first introduced in the context of retail transaction databases (Agrawal & Srikant 1994). An association rule is an implication of the form $X \Rightarrow Y$, where the rule *body* X and *head* Y are subsets of the set I of items ($I = i_1, i_2, \ldots, i_n$) within a set of *transactions D* and $X \cap Y = \Phi$. A rule $X \Rightarrow Y$ states that the transactions *T* that contain the items in *X* are likely to also contain the items in *Y*. Association rules are characterized by two measures: the *support*, and the confidence, which measures the proportion of transactions in *D* containing items *X* that also contain items *Y*. *Confidence*($X \Rightarrow Y$) can be expressed as the conditional probability *p(Y|X)*. Thus, we define:

$$Support(X \Rightarrow Y) = \frac{count(X \cup Y)}{N_{trans}} = \frac{root/X/Y/@counter}{root/@counter}$$

$$Confidence(X \Rightarrow Y) = \frac{\sup port(X \Rightarrow Y)}{\sup port(X)} = \frac{count(X \cup Y)}{count(X)} = \frac{root/X/Y/@counter}{root/X/@counter}$$

Example: The following example introduces the XQuery function for discovering a set of association rules related to a given event, from the FXT constructed by our algorithm.

```
declare function local:getAssRules($x as xs:string){
let $path_x := $input/*[name(.) = $x]
return
  (for $y in $path_x/*
   let $y_given_x := name($y)
   let $supp_xy := $y/@counter div $rootCounter
   let $supp_x := $path_x/@counter div $rootCounter
   let $confidence := $supp_xy div $supp_x
   return
     <rule body="{$path_x/name()}" head="{$y_given_x}"
     support="{$supp_xy}" confidence="{$confidence}"/>)        };

(: call the function :)
local:getAssRules("A")
```

Fig. 12 XQuery function for mining association rules

The result of calling this function is as follows:

```
<rule body="/A" head="B" support="0.25" confidence="0.667"/>
<rule body="/A" head="C" support="0.25" confidence="0.667"/>
```

Fig. 13 Result of XQuery function for mining association rules

5.2 Event repository

The event repository involves a variety of events that are logged from different framework modules. It can be considered as part of metadata, because the administrator can query it to trace different framework activities. Events can arise from changing structures or contents of data sources. Examples of structure changes events include events of adding, altering, and/or dropping data sources, objects, and/or fields. Examples of content changes events include data manipulation events such as inserting, updating, and/or deleting values at specific data source. Such events of changes are logged into the event repository. Integration services log events about data sources that they extract, transform, and load. Main event information includes date processed, number of records read, written, input, output, updated, errors encountered, and services carried out. Another category of events is logged when querying AXML documents. These events include information about AXML documents, XQuery expressions, XPaths expressions, as well as date of querying.

Since events are warehoused into the event repository, the repository can be modeled and stored according to a *star schema* model (Kimball et al. 1998), where a single fact document represents all events (figure 14). However, there are multiple dimension documents (e.g., user dimension, machine dimension, session dimension, data source dimension, AXML document dimension, date and time dimension). Another solution is to model the repository according to constellation schema model (Kimball et al. 1998), where there are several fact documents; one fact document per each type of the event types such as data source contents or structure changes, invoking integration service, querying AXML repository, and messaging. However, there are still multiple dimension documents. Warehousing events allows administrator to apply XML Online Analytical Processing (X-OLAP) (Park et al. 2005), in order to explore and analyze logged events information.

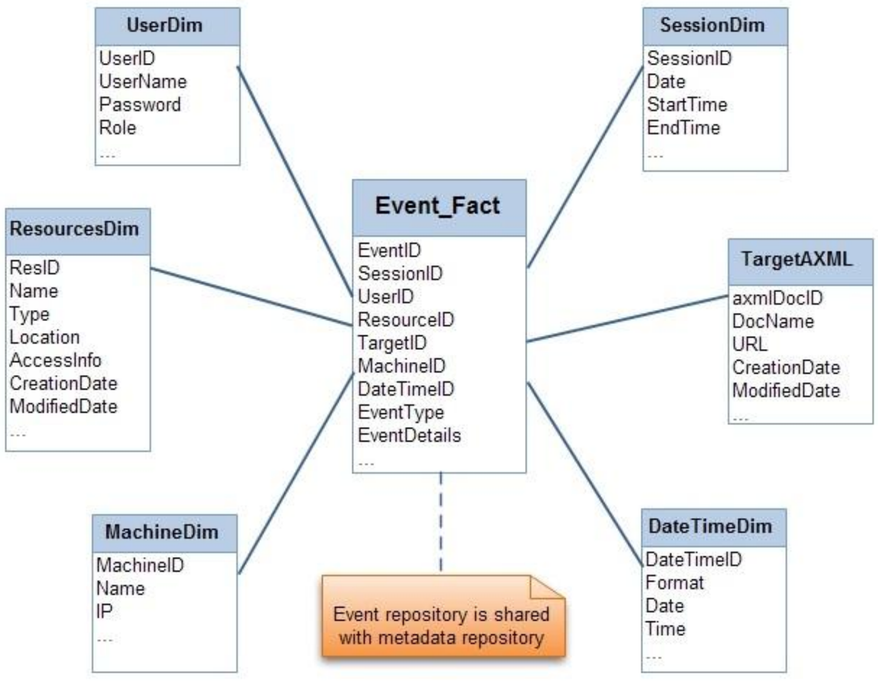

Fig. 14 Logical star schema of event repository

5.3 ECA rules

Active rules are incorporated into our data integration system to enrich them with automatic, reactive and intelligent features. These rules enable integration systems to respond automatically to encountered events. Once a given event is detected, the associated rules are fired. Events can be temporal events or data integration system events (section 5.2). Events are either simple or composite. Simple events (also called, primitive or atomic events) correspond to a data modification operation, execution of integration service, or querying an AXML document. A composite event is a logical combination of simple events or other composite events, defined by logical operators such as disjunction *(or),* conjunction *(and),* sequence *(and then),* and negation *(not).* We define such composite events by examples in XML (Salem et al. 2011). Conditions denote queries to one or several events and are expressed in XPath or XQuery. Actions can be notifying the integration server when a data source changes, sending message, or invoking integration service. Note that, evaluating condition and executing action can be logged as event. Thus, we assume that there is an attribute for each logged event indicates its origin whether event, condition, or action.

Although the general forms of active rules conform to 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). The omission of both event and condition parts are not allowed for the same rule.

Example: In the following example, we assume that identification of events, condition and actions are the same as their identification when logged as events, for simplicity. For each detected event, function is called to get associated events of the detected one. It then tests the origin of detected and associated events to formulate the active rule. In action part, if the confidence between detected and associated event is 1, the rule is formulated immediately, the user is notified for recommendation if confidence greater than 0.65 otherwise do nothing. The action is always invoking AXML service. In this example, the active rule is formulated dynamically by merging ECA paradigm and AXML (figure 15).

```
<eca:rules>
 <eca:variable name="eventID" lang="xpath"  expr="./@ID"/>
 {
  for $rule in local:getAssRules($eventID)
  let $ruleHead := data($rule/@head)
  let $eventBodyOrigin := doc("events") /event[@id=$eventID]/origin
  let $eventHeadOrigin := doc("events") /event[@id=$ruleHead]/origin
  where $eventBodyOrigin="event" or $eventBodyOrigin="condition"
  return
  <eca:rule>
  {
   if ($eventBodyOrigin="event") then
   <eca:event id="{$eventID}"/>
   else()
  }

  {
   if ($eventBodyOrigin="condition") then
   <eca:condition id="{$eventID}"/>
   else if ($eventHeadOrigin="condition") then
   <eca:condition id="{$ruleHead}"/>
   else()
  }

  {
   if ($rule/@confidence = 1 and $eventHeadOrigin="action") then
   <eca:action>
      <axml:cs service="{$ruleHead}"/>
   </eca:action>
   else if ($rule/@confidence = 0.65 and $eventHeadOrigin="action") then
   <eca:action>
      <axml:cs service="send-message">
        <input-param  value=" The action {$ruleHead} is suggested to be taken"/>
      </axml>
   </eca:action>
   else ()
  }
 </eca:rule>
 }
</eca:rules>
```

Fig. 15 Formulating active (ECA) rule by A^{2}XML

5.4 Active Active XML (A^{2}XML) engine

We utilize active (or ECA) rules to activate AXML services invocation; we term the merging of both technologies Active Active XML (for short A^{2}XML). The A^{2}XML merges the ECA paradigm into the AXML language to control the flow behavior of data integration environment. The A^{2}XML engine plays an important role to control the evaluation of the active rules. It manages event triggering, evaluates the conditions, and performs the associated rule actions. The A^{2}XML engine manages the evaluation and composition of integration services. All integration services are registered to the A^{2}XML engine. It is supplied with business rules to handle flow and timing activation of services. It also monitors querying the AXML documents, and then invokes the embedded services of the queried documents to refresh them with up-to-date information. Beside evaluating services implicitly when data are requested, services can also be evaluated explicitly (e.g., daily, weekly or after occurring some events). Evaluating services explicitly may depend on data update frequency, which differs w.r.t. the nature of various applications. Moreover, the engine manages where services' results are written, either replacing the calling service or appending to it.

6 Implementation Issues

We implemented our framework prototype mostly using standard open-source software. It is implemented as a Web-based application[3] using Oracle, XML and Java technologies. Figure 16 shows the deployment diagram, which visualizes the hardware, middleware and software used in our implementation. The diagram is composed of multiple tiers: data source, integration application, web, client and target tiers. Different tiers communicate via several protocols and interfaces. For instance, there are Java Application Programming Interfaces (APIs) that can facilitate extracting metadata schema from heterogeneous data sources. Relevant metadata schema of data sources can be easily determined via the application's GUI.

The integration application tier is the main core of our application. It is composed of two main sets of cooperative components: application manager and native XML database. The application manager components are implemented using Java. From an implementation viewpoint, using an open source tool for performing ETL tasks saves implementation time. Our framework utilizes the Java library of *Pentaho Data Integration* (PDI)[4]. PDI is a powerful, metadata-driven ETL tool. The framework adapts the structure of relevant data sources from the input schema into XML files that can be handled by PDI tools (e.g., *Pan* or *Kitchen).* Then, PDI tools are invoked to execute transformations or jobs, which are described in XML format. The XML file of a specific transformation involves metadata about the input data source and its detailed characteristics, metadata about transformation rules, and metadata

[3] http://eric.univ-lyon2.fr/~rsalem/axdi/

[4] http://kettle.pentaho.com/

about output targets. Such metadata describe the ETL operation. The XML database is implemented using *Sedna*[5]. It involves several collections of databases, such as metadata of framework components, defined and mined active rules, and logged events. Such databases are employed as utilities for application manager components. Active rules are implemented, triggered and fired using the XQuery trigger facility supported by Sedna.

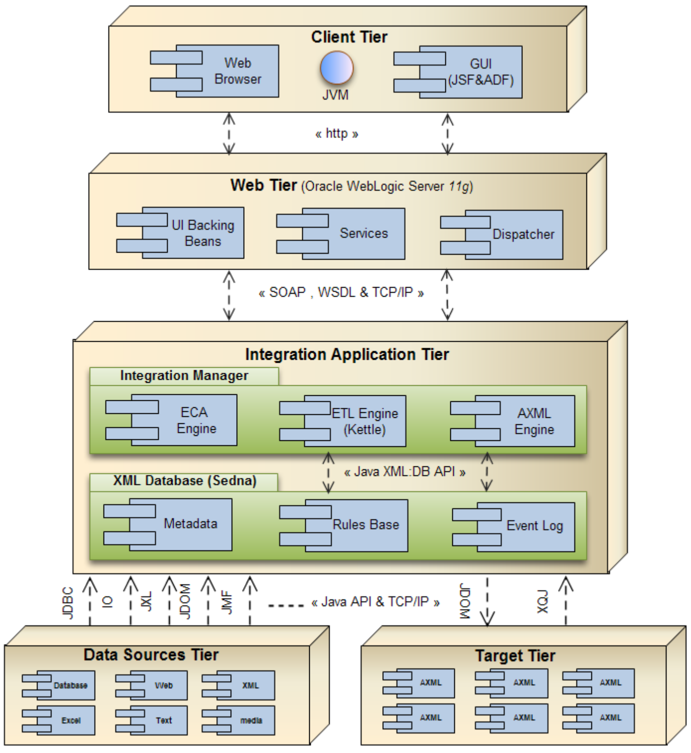

Fig. 16 Deployment diagram of data integration web application

In the web tier, integration services, dispatcher services, and user interface components bindings are implemented in Java. They are deployed to Oracle WebLogic server 11g. At the client tier, users can explore our application via a web browser. The application's GUI is developed using components of Oracle Application Development Framework (ADF) and Sun JavaServer Faces (JSF). The user specifies the interesting data source, and then extraction services bring out the metadata from which to select a relevant schema. Rather than defining data sources via application's GUI, users have access to several functionalities such as mapping data sources with integration services, scheduling the execution of integration services, mapping integration services with targets, browsing and query AXML repository, browsing changes of data sources, and defining ECA rules. It also allows applying the general setting of

[5] http://www.modis.ispras.ru/sedna/

the framework, and applying the configuration of change detection and native-XML database. Moreover, a demo of the AXML browser is implemented in order to navigate AXML documents. The browser is split to display the AXML documents before and after invoking embedded services. Finally, the target of the integration tasks is always a set of AXML documents that are also stored in a native XML database.

7 Conclusions and Perspectives

In this paper, we propose an innovative metadata-based, service-oriented, and event-driven data integration framework meeting next generation data integration, for better decision-making in DW/BI applications. Our framework addresses two main issues of Web (complex) data integration: 1) a near real-time data integration, and 2) an autonomous and active data integration. Our framework noticeably tackles limitations of traditional data integration systems utilizing Web standards for performing data integration tasks (for instance, it uses the XML language to handle data heterogeneity, Web services to tackle data distribution, and AXML to store integrated data). Employing such Web standards in data integration helps integrating real-time data, for example by invoking embedded services in the AXML documents to integrate real-time data on the fly, or by applying the subscription/notification paradigm to services that responsible for detecting, capturing, and notifying only real-time data changes. Furthermore, framework logged events are mined in innovative manner to discover interesting knowledge to help self-maintain, self-configure, and automate data integration tasks. Some of active rules are also enriched with the system to activate different integration tasks. Finally, as a proof of concept, we have implemented a web application prototype that is freely available on-line[6].

Recall that different framework events are logged and warehoused in a specified repository, called event repository. Thus, OLAP tools and other data mining techniques can be applied in the future to explore and analyze logged event information. We also aim to carry out machine learning to realize autonomous semantic mediation between the data sources, integration services and the targets. Therefore, we can ensure achieving more automated data integration workflow. Moreover, integration service composition is important to meet the complexity of business integration needs. We intend to study service composition for more complicated business data integration. We addressed in this paper the concepts of *Integration as a Service* (IaaS), we intend in the future to address different challenges for deploying our

[6] http://eric.univ-lyon2.fr/~rsalem/axdi/

framework of these concepts in the cloud. Moreover, data quality is another critical aspect to DW/BI applications and it goes hand in hand with data integration. The right data quality during the process of loading a data warehouse leads to better informed and more reliable decisions. Thus, addressing data quality is another interesting future trend.

References

Abiteboul, S., Benjelloun, O. & Milo, T. (2002), Web services and data integration, *in* 'Proceedings of the 3rd International Conference on Web Information Systems Engineering', WISE '02, IEEE Computer Society, Washington, DC, USA, pp. 3–6.

Abiteboul, S., Benjelloun, O. & Milo, T. (2008), The active XML: an overview, *in* 'VLDB Journal', pp. 1019–1040.

Abiteboul, S., Manolescu, I. & Zoupanos, S. (2008), OptimAX: Optimizing distributed activeXML applications, *in* D. Schwabe, F. Curbera & P. Dantzig, eds, 'ICWE', IEEE, pp. 299–310.

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

Agrawal, R. & Srikant, R. (1994), 'Fast algorithms for mining association rules', *Very Large DataBase, VLDB* pp. 487–499.

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

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

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

Bonifati, A., Ceri, S. & Paraboschi, S. (2002), 'Pushing reactive services to XML repositories using active rules', *Computer Networks* **39**(5).

Boussaïd, O., Messaoud, R. B., Choquet, R. & Anthoard, S. (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* pp. 39–54.

Chawathe, S. S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J. D. & Widom, J. (1994), The TSIMMIS project: Integration of heterogeneous information sources, *in* ‘IPSJ’, pp. 7–18.

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

Feng, L. & Dillon, T. (2004), Mining interesting XML-enabled association rules with templates, Springer.

Gaber, M. M., Zaslavsky, A. B. & Krishnaswamy, S. (2005), ‘Mining data streams: A review’, *ACM SIGMOD Record* **34**(2), 18–26.

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

Halevy, A. Y., Rajaraman, A. & Ordille, J. J. (2006), Data integration: The teenage years, *in* U. Dayal, K.-Y. Whang, D. B. Lomet, G. Alonso, G. M. Lohman, M. L. Kersten, S. K. Cha & Y.-K. Kim, eds, ‘Proceedings of VLDB’, pp. 9–16.

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

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

Jiang, N. & Gruenwald, L. (2006), ‘Research issues in data stream association rule mining’, *ACM SIGMOD Record* **35**(1), 14–19.

Kimball, R. & Merz, R. (2000), *The Data Webhouse Toolkit: Building the Web-Enabled Data Warehouse,* John Wiley & Sons.

Kimball, R., Reeves, L., Ross, M. & Thornthwaite, W. (1998), *The data warehouse toolkit,* John Wiley & Sons.

Knoblock, C. A., Minton, S., Ambite, J. L., Ashish, N., Muslea, I., Philpot, A. G. & Tejada, S. (2001), ‘The ariadne approach to web-based information integration’, *International Journal of Cooperative Information Systems* **10**(1 & 2), 145–169.

Machani, S. (1996), Events in an active object-relational database system, Master’s thesis.

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

Milo, T., Abiteboul, S., Anman, B., Benjelloun, O. & Ngoc, F. (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., Rajugan, R., Dillon, T. & Rahayu, J. (2005), 'Conceptual and systematic design approach for XML document warehouses', *International Journal of Data Warehousing & Mining* **1**(3), 63–86.

Nørvåg, K. (2002), Temporal XML data warehouses: Challenges and solutions, *in* 'Proceedings of Workshop on Knowledge Foraging for Dynamic Networking of Communities and Economies, Shiraz, Iran'.

Park, B., Han, H. & Song, I. (2005), 'XML-OLAP: A multidimensional analysis framework for XML warehouses', *7th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'05), Copenhagen, Denmark* pp. 32–42.

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)* pp. 267–280.

Rajugan, R., Chang, E. & Dillon, T. (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* pp. 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* .

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

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

Salem, R., Boussaid, O. & Darmont, J. (2011), An autonomic axml-based framework for integrating heterogeneous data, *in* 'submitted to The 13th International Symposium on Database Programming Languages (DBPL 2011), Seattle, Washington, USA'.

Salem, R., Darmont, J. & Boussaid, O. (2011), Efficient incremental breadth-depth xml event mining, *in* 'Submitted to 15th International Database Engineering & Applications Symposium (IDEAS'11), Lisbon, Portugal'.

Sheth, A. P. & Larson, J. A. (1990), 'Federated database systems for managing distributed and autonomous databases', *ACM Computing Surveys* pp. 183–236.

Thalhammer, T., Schrefl, M. & Mohania, M. (2001), ‘Active data warehouses: Complementing OLAP with active rules’, *Data and Knowledge Engineering* **39**(3), 241–269.

Tho, M. N. & Tjoa, A. (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., Lemos, F. & Porto, F. (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* pp. 43–50.

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

Whoweda (2011), ‘The web warehousing & mining group’. http://www.cais.ntu.edu.sg/ whoweda/index.htm

William Inmon, Derek Strauss, G. N. (2008), *DW 2.0: The Architecture for the Next Generation of Data Warehousing*, MORGAN KAUFMANN.

Wu, W. (2006), Integrating deep web data sources, PhD thesis, Champaign, IL, USA.

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

Yu, P. S. & Chi, Y. (2009), ‘Association rule mining on streams’, *In Encyclopedia of Database Systems, Springer US* pp. 136–139.

Zhao, Q., Chen, L., Bhowmick, S. S. & Madria, S. K. (2006), ‘XML structural delta mining: Issues and challenges’, *Data Knowl. Eng* **59**(3), 627–651.

Zhu, F., Turner, M., Kotsiopoulos, I. A., Bennett, K. H., Russell, M., Budgen, D., Brereton, P., Keane, J. A., Layzell, P. J., Rigby, M. & Xu, J. (2004), Dynamic data integration using web services, in *ICWS*, pp. 262–269.

Ziegler, P. & Dittrich, K. R. (2004), Three decades of data integration - all problems solved?, *in* R. Jacquart, ed., ‘IFIP Congress Topical Sessions’, Kluwer, pp. 3–12.