**Noname manuscript No.**
(will be inserted by the editor)

# Active XML-based Web Data Integration

**Rashed Salem · Omar Boussaïd · Jérôme Darmont**

Received: date / Accepted: date

**Abstract** Today, the Web is the largest source of information worldwide. There is currently a strong trend for decision-making applications such as Data Warehousing (DW) and Business Intelligence (BI) to move onto the Web, especially in the cloud. Integrating data into 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.
In this paper, we propose a generic, metadata-based, service-oriented, and event-driven approach for integrating Web data timely and autonomously. Beside handling data heterogeneity, distribution and interoperability, our approach satisfies near real-time requirements and realize active data integration. For this sake, we design and develop a framework that utilizes Web standards (e.g., XML and Web services) for tackling data heterogeneity, distribution and interoperability issues. Moreover, our framework utilizes Active XML (AXML) to warehouse passive data as well as services to integrate active and dynamic data on-the-fly. AXML embedded services and changes detection services ensure near real-time data integration. Furthermore, the idea of integrating Web data actively and autonomously revolves around mining events logged by the data integration environment. Therefore, we propose an incremental XML-based algorithm for mining association rules from logged events. Then, we define active rules dynamically upon mined data to automate and reactivate integration tasks. Finally, as a proof of concept, we implement a framework prototype as a Web application using open-source tools.

**Keywords** Real-time Web data integration · metadata · integration services · active rules · event mining

R. Salem, O. Boussaïd, J. Darmont
Université de Lyon (ERIC Lyon 2)
5 av. P. Mendès-France, 69676 Bron Cedex, France
Tel.: +33 (0)4 78 77 31 54
Fax: +33 (0)4 78 77 23 75
E-mail: rsalem@eric.univ-lyon2.com

## 1 Introduction

Business intelligence (BI) is based on 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, on-line 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.

Nowadays, the Web becomes a commonly accepted development and delivery platform. Therefore, modern DW/BI applications currently have moved onto the Web, e.g., to integrate Web data to make better decision, or benefit from cloud computing facilities. In the

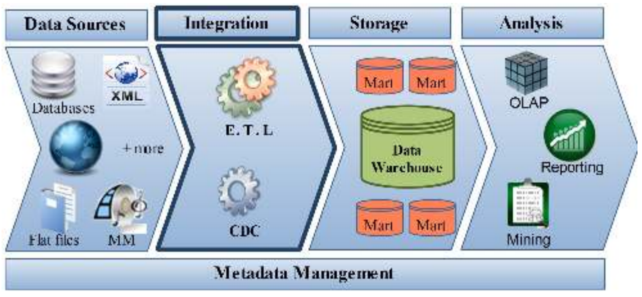

**Fig. 1** Data warehousing/business intelligence process

cloud, BI is accessible via any browser; there is no need to buy neither software nor hardware. With BI software running in the cloud, data integration must be carried out with back-end systems, far from end-users.

Moreover, the Web is the world's largest source of information. The Web is very rich with distributed and heterogeneous data (e.g., semi-structured/unstructured data, chat logs, e-mails, images, videos, feedback and surveys). Other valuable data types include relational databases (deep Web), HTML/XML documents, spreadsheet files, RSS feeds, on-line transaction records, external data feeds, sensor and streaming data. We term such 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).

One of major problematic issues that we address in this paper is to warehouse complex data into a unified storage to be later analyzed for decision-support purposes. Another problematic issue is that the data flow in traditional data integration systems in one-way. The dominant way of populating data warehouses and data marts is to operate ETL processes off-line, 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. Today's requirements impose integrating complex data in near real-time rather than with traditional snapshot and batch ETL. 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. Since traditional data integration systems are passive in nature, data integration tasks conform to a static scheduling plan, whereas real-time and near real-time processing require more active ETL processes. Therefore, today's DW/BI applications must react in an intelligent, i.e., active and autonomous way, to encountered changes in the data integration environment, especially data sources.

In this paper, we propose an approach for integrating complex Web data in near real-time, actively and autonomously. We indeed provide a generic metadata-based, service-oriented and event-driven approach for integrating Web data. The main contributions of our approach follow.

Firstly, the complexity of Web data is handled using our metadata-based and service-oriented features of our approach. For instance, heterogeneous Web data are unified and integrated into a common repository by exploiting the XML language. The XML language bears several interesting features for representing, exchanging and storing Web data. Moreover, data distribution and interoperability are tackled using Web services. Beside exchanging data among different applications and platforms, Web services provide transparent access to a variety of relevant data sources as if they were a single source. Additionally, Web data are integrated into a unified repository of AXML documents. AXML documents are XML documents where some of the data are given explicitly, while other parts are given implicitly by embedding calls to Web services (Abiteboul et al., 2008a; Phan et al., 2012). The AXML technology is mainly used in peer-to-peer mediation, but we reuse it in data warehousing.

Secondly, our service-oriented feature ensures integrating data in near real-time using a bi-directional strategy, i.e., forward integration (push) and backward integration (pull). Beside employing integration services to carry out traditional ETL tasks, some management services (i.e., change detection services) are employed to track data changes and push them to integration services. Indeed, integration services (change consumers) subscribe to change detection services (change providers) to be notified with only relevant near real-time changes. Moreover, by integrating data into an AXML repository, not only data are warehoused, but also functions (or services) for integrating data on-the-fly. In a pull strategy, embedded AXML services are invoked implicitly or explicitly to enrich and refresh the repository with data in near real-time.

Thirdly, we provide an event-driven approach for integrating data actively and autonomously. The simple idea of this approach is to warehouse all logged events that describe integration activities into an XML-based event repository. Then, the warehoused events are mined and analyzed to maintain, automate and reactivate data integration tasks. Therefore, we propose an efficient incremental algorithm for mining association rules from logged events in XML format (Salem et al., 2011). A novel compressed tree structure, namely FXT (Frequency XML-based Tree), which contains most critical information about logged events is proposed. An FXT is built with a single-pass over logged event and enables transaction insertion alongside transaction deletion. Both frequent events and association rules are discovered from FXT using XQuery. Furthermore, active rules are incorporated to reactive data integration tasks. Such rules are defined either explicitly by the user or implicitly by querying frequent events and association rules from the FXT.

Finally, to validate our approach for managing Web data integration, we develop a high-level software framework using open-source tools. The framework prototype is implemented as Web application.

The rest of this paper is organized as follows. In section 2, we review the state of the art of data integration, Web warehousing, real-time/active warehousing, and AXML and active rules for XML. Section 3 introduces our generic framework for complex Web data integration. The role of active metadata in data integration is presented in section 4. Our service-oriented feature for near real-time data integration is presented in section 5. Moreover, our event-driven feature that incorporates active rules to reactive data integration tasks is discussed in section 6. Implementation issues are presented in section 7. Finally, we conclude and highlight future trends in section 8.

## 2 State of the Art

In this section, we review and discuss the literature related to data integration approaches, Web warehousing, active XML and active rules, and real-time/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 and Dittrich (2004). In general, there are two main families of approaches to integrate data: virtual views (mediation) and materialized views (warehousing).

#### *2.1.1 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 and 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, Wu (2006) propose to build a virtual integration system over *Deep Web* sources, the sources that are only accessible through their query interfaces. The proposed integration system provides a uniform access to sources, thus freeing users from the details of individual sources. Additionally, the PAYGO architecture is proposed for achieving web-scale data integration, focusing on integrating structured data from the Deep Web (Madhavan et al., 2007). Unlike in traditional data integration, PAYGO has not a single mediated schema since web scale requires everything to be modeled. PAYGO has a repository of schemata that are clustered by topic. Due to heterogeneity at Web-scale, approximate schema mapping is supported by PAYGO rather than semantic mapping between data sources. Mechanisms that incrementally improve semantic relationships between data sources over time are also provided in PAYGO.

XML and Web services are also employed for integrating Web data in federated systems (Zhu et al., 2004), and in peer-to-peer (P2P) and mediated systems, e.g., in the AXML project (Abiteboul et al., 2002, 2008a). 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. Furthermore, mashup-based data integration provides a programmatic, data-flow-like and service-oriented approach for Web data integration (Lorenzo et al., 2009). In addition to the fact that mashup tools are mainly designed for handling Web data, such tools are mostly simple and do not develop advanced programmatic data integration, e.g., to analyze large sets of Web data. Therefore, Thor and Rahm (2011) present an advanced script language that provides operators for data integration tasks such as query generation and entity matching. This language is presented in CloudFuice, which follows a mashup-like specification of advanced dataflows for data integration in the cloud.

#### *2.1.2 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 (Kimball and Ross, 2002; Inmon, 2002). 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 (Darmont and Boussaïd, 2006). 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 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, either.

Beside relational data warehousing approaches, 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. Hümmer et al. (2003) propose a family of XML-based templates 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 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 Web Warehousing

The problem of warehousing 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 (Bhowmick et al., 2003). In Xyleme (2001), a dynamic warehouse is built for massive volumes of XML data from the Web, several issues have been addressed in this project 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 (Bhowmick et al., 2003). This project is mainly focused on the definition of a formal data model and algebra to represent and manage Web documents, their physical storage, and change detection. In Whoweda project, the Web warehouse consists of a collection of constructs, namely Web tables, which represent sets of interlinked Web documents. The tuples of Web tables are multigraphs where each node represents a document, and edges represent hyper-links between documents (Pérez et al., 2008).

Kimball and 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. Moreover, Vrdoljak et al. (2003) propose a semi-automated methodology for designing Web warehouses from XML sources modeled by XML Schemes. In this methodology, design is carried out by first creating a schema graph, then navigating its arcs in order to derive a correct multidimensional representation. This approach was implemented through a prototype that reads an XML schema and outputs the logical star schema of the warehouse. Finally, several innovative approaches exploiting XML are proposed for warehousing Web data (Boussaid et al., 2008; Bentayeb et al., 2011). These approaches address Web data integration, multi-dimensional modeling and storage into XML data warehouses, as well as enabling user-driven analysis via coupling data mining and OLAP.

### 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 with embedded calls to Web services. When one of these calls is invoked, its result is returned to enrich the original document (Abiteboul et al., 2008a). 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 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., 2008b).

Active rules are widely known in active databases (Paton, 1999). They follow the 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. (2002b) 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. Rekouts (2005) also describes a general form of active rules for XML based on XQuery and previously defined update languages.

### 2.4 Active and Real-Time Warehousing

To improve data freshness and to realize real-time decision support systems, Tho and Tjoa (2003) propose a framework for building Zero-Latency Data Warehouses (ZLDWs). 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.

Moreover, Karakasidis et al. (2005) propose an architectural framework for the implementation of active data warehousing and develop a theoretical framework for the problem, by utilizing queue theory for the prediction of performance of the refreshment process. 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. Additionally, Polyzotis et al. (2007) address the problem of refreshing active warehouses on-line to ensure a higher degree of consistency between the stored information and their latest update. The authors propose a novel join operator called MESHJOIN for joining a fast stream S of source updates with a large warehouse relation R under the assumption of limited memory. Naeem et al. (2011) extends MESHJOIN by designing an adaptive algorithm called Extended Hybrid Join (X-HYBRIDJOIN) that can adapt to data skew and stores parts of the master data in memory permanently, reducing the disk access overhead significantly.

Furthermore, Naeem et al. (2008) discuss an event-based near real-time ETL for transferring and transforming data from operational databases to a data warehouse. The authors propose an event-driven and near-real time architecture for ETL that uses a Database Queues (DBQ), works on a push technology principle and supports content enrichment. A detailed state of the art study of near real-time ETL is discussed by Vassiliadis and Simitsis (2009). The authors also detail the infrastructure of near real-time ETL presenting alternative topologies, issues related to parallelism and partitioning techniques, and issues concerning the communication of the different parts of the architecture.

Finally, several technological issues and considerations when building real-time data warehouses are discussed by industrial communities (Oracle, 2010; Brobst and Ballinger, 2003).

### 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 to represent static and passive parts of the document, but the second set of nodes is defined implicitly by calling services to integrate data on the fly, virtually, to represent the dynamic and active part of the document. 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. Indeed, calling information of AXML services may or may not occupy larger storage than their results, but they can certainly be reused infinitely to refresh the repository. Unlike in the virtual data integration approach, 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 by saving response time when querying data, especially if data sources are physically located far from the integration system. Beside data historization facilities, the data warehousing approach enables complex analyses unlike in virtual and mashup-based approaches. Therefore, we conclude that our approach is a hybrid of virtual and warehousing approaches.

Compared to existing Web integration approaches, our approach aims at integrating not only XML data, like in Xyleme (2001); Vrdoljak et al. (2003), but also other types of complex data including Web data (e.g., structured, semi-structured, images, videos, textual data, etc.), and unifying them into an XML format. While Xyleme and Whoweda are very large-scale, requiring unbounded resources due to the increasing volumes of XML documents and Web data, medium-scale or small-scale Web data may need to be warehoused for a specific decision support system. Moreover, our approach aims at integrating Web data in an innovative real-time and autonomous manner, different than Abiteboul et al. (2006); Wu (2006); Vidal et al. (2008).

Data warehousing, particularly in the analysis phase, tends to satisfy active features (Thalhammer et al., 2001), while the data integration phase tends to satisfy real-time requirements (Tho and Tjoa, 2003; Naeem et al., 2008). However, ZLDWs must update data frequently to improve data freshness using push/pull technologies 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 no either 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. The event-driven data integration approach (Naeem et al., 2008) deals mainly with operational data sources and does not handle data complexity issues. Beside handling data complexity issues thanks to a service-oriented feature, our integration approach automates integration tasks via an event-driven feature by mining and analyzing logged events, which are incorporated with active rules. Therefore, only initial or trained integration tasks need to be integrated by the 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 a generic framework, namely AX-InCoDa (**A**ctive **X**ML-based framework for **In**tegrating **Co**mplex **Da**ta). Through AX-InCoDa we validate our approach (i.e., metadata-based, service-oriented and event-driven) for integrating complex data including complex Web data (figure 2). Integrating Web data for DW/BI poses different and new requirements to data integration technologies, over those posed by conventional data integration systems (Kimball and Ross, 2002; Inmon, 2002). Therefore, more than one tool is required for combining, transforming, and organizing data into a common syntax. Web standards (e.g., XML, Web services and related technologies) can help enterprises to integrate such complex data. Typically, Web standards are the core technologies that are utilized in AX-InCoDa for integrating complex Web 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 can solve the data distribution and interoperability problems by exchanging data between different applications and different platforms. Web services are self-contained, self-describing modular applications that can be published, located, and invoked across the Web. Indeed, a service provides transparent access to a variety of relevant data sources as if they were a single source (Erl, 2004; Zhu et al., 2004; Schlesinger et al., 2005; Utomo, 2011). Moreover, according to informatica[1], *Software-as-a-Service* (SaaS) integrate cloud-based data easily with on-premise systems to ensure users can access accurate, complete, up-to-date data whenever and wherever they need it. *Integration-as-a-Service* (IaaS) is a flexible, scalable and reusable integration approach where the core integration technology that performs integration functions, such as semantic mediation, data migration, connectivity, and other core integration facilities, is delivered from the Web as a service (Linthicum, 2010).

Therefore, we employ metadata-based services to integrate data from distributed data sources. 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., 2008a). Using XML and Web service standards also provides a less expensive and more efficient data integration infrastructure, typically by leveraging services to extract and detect changed

[1] http://www.informatica.com

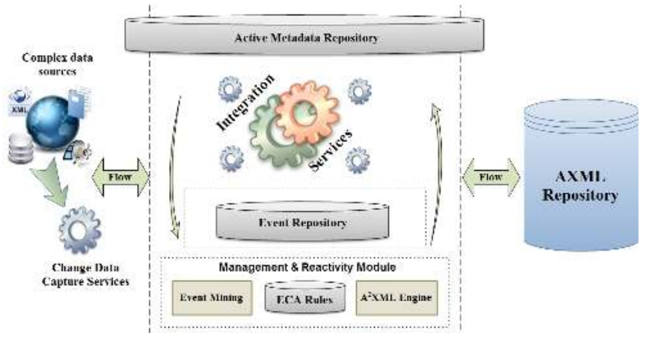

**Fig. 2** AXML data integration framework

data, providing transformation services to unify data, as well as loading services to move data from sources to targets (section 5). Integrated data are targeted to a native-XML based repository, namely an AXML repository; while integration services integrate relevant 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 to self-maintain, self-manage and automate the execution of integration services, specifically when incorporating the framework with active rules paradigm (section 6).

## 4 Metadata Management for Data Integration

Metadata are a critical element in effective data integration. They are considered as a potential integration tool providing useful information that facilitates communication between information systems. While metadata management was not considered as a significant part in traditional data integration systems, today's metadata management plays a critical role in developing and maintaining current data integration environments. Integration services in AX-InCoDa 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. Since AX-InCoDa is a metadata-driven, we address in this section the active role of metadata to facilitate data integration.

### 4.1 Active Metadata

#### *4.1.1 Active metadata management*

Traditionally, metadata are used passively and are created and maintained as documentation about the structure, the development and the use of a data integration system. However, there is now a trend in data integration toward metadata taking on a more active role (William Inmon, 2008). In AX-InCoDa, metadata management helps deal with the heterogeneity, variety, and complexity of data sources, as well as the automation of data integration tasks. For example, specifying source-to-target mappings (under the form of metadata) serves as the basis for automating extraction and transformation tasks. The well-organized metadata, the easier administration of the data integration environment. Therefore, metadata can minimize the efforts for developing and administrating a data integration platform. Beside supporting and automating data integration, metadata improve the flexibility of the platform and the reuse of existing integration services. Moreover, metadata enforce security mechanisms by providing access permission rules and user rights. When integrating data, metadata can also improve data quality through enforcing consistency, validation, accuracy, completeness and business rules.

Furthermore, AX-InCoDa integration and management services are metadata-driven. Such services need metadata about data sources, targets, mapping and rules to carry out their functionalities. Since these services are metadata-driven, they can be reused for different integration tasks and do not require re-engineering. We indeed distinguish between reusable metadata-based services and specific-domain services. Some examples of metadata-based services include extracting HTML table structure, XML patterns, image attributes, etc. These services can be reused everywhere for the same types of sources by changing the service arguments, i.e., source URL. Examples of specific-domain services include extracting a specific product specification, price, review, etc. These services are designed to get their results from pre-determined sources. This does not mean that they cannot be reused, but they can be reused for a similar source structure and may require re-engineering when the structure of the source changes. Moreover, metadata management integration services (e.g., change detection, logging, etc.) are usually reusable for any domain.

#### *4.1.2 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. In AX-InCoDa, we follow the principles of active metadata by always querying the metadata repository on-line when developing and maintaining data integration tasks. 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. Examples of metadata include data about data sources, integration services, target AXML repository, source-to-target mappings, active rules, and event repository.

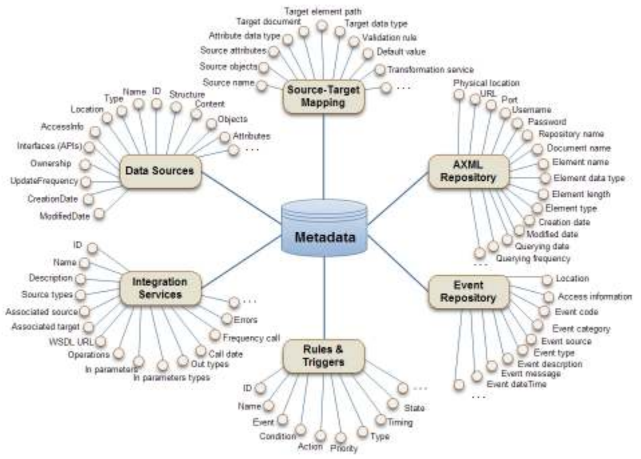

**Fig. 3** Metadata samples of our data integration framework

### *4.1.3 Integration schema evolution*

The integration schema specifies the structure of relevant data sources, specifies 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. A detailed workflow that describes the sequence of the framework processes is presented in Salem et al. (2010). The workflow illustrates how relevant changes of data sources are monitored and detected, and then how these changes update the integration schema implicitly. 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 change detection service to notify the integration system with data source structure changes). This schema can be continuously queried during system development, for instance to validate whether data source changes are relevant or not.

## 4.2 Conceptual Data Model

In order to illustrate associations between our framework's entities, we present AX-InCoDa conceptual data model in figure 4 as a UML class diagram. Classes in this model represent *metadata* such as entity, attribute, table, column, and index. This model describes the relationships between the main entities of the framework, such as data sources, integration services, AXML document, and logged events. The Complex_Source class contains metadata descriptions of complex data sources. Since we handle several types of data sources, each data source follows a specific source type (e.g., relational database, XML, text, image, etc.). A data source is composed of one or more objects, where each object is composed of one or more fields. 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.

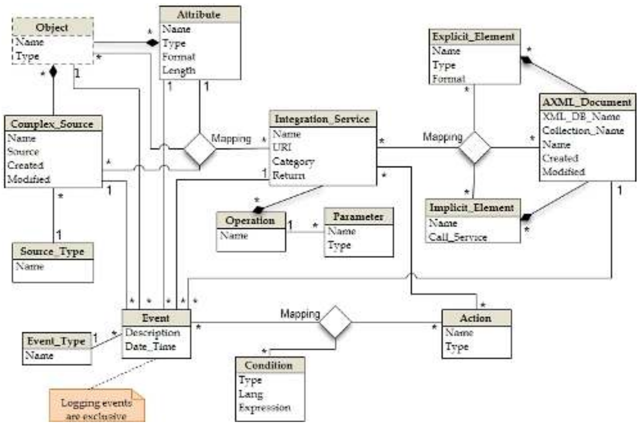

**Fig. 4** The framework conceptual data model

Data sources, their objects and their attributes are mapped with their associated integration services. The Integration_Service class contains metadata descriptions of 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 bearing one or more parameters. Metadata description of service operations and operation parameters are specified in the 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 when defining data sources, objects, fields, executing integration services, and querying AXML documents. Although there are several types of events (section 6), an event superclass is specified for simplicity. Logged events are exclusive; each event type has its own description. Events, conditions, and actions are mapped together to form active rules. The Event, Condition and Action classes contain the description of events, conditions, and actions, respectively. Actions are always associated to integration services.

## 5 Service-oriented, Near Real-time Integration

The Web has become a common platform for the delivery of business applications. In particular, Web service technologies and their extensions, e.g., SaaS, IaaS, Everything-as-a-Service (Li and Wei, 2012), etc., enable the development of complex business applications on the Internet/in the cloud by integrating and exchanging Web services from different providers. In AX-InCoDa, we present integration services as a set of reusable Web services, which are designed for integrating data from heterogeneous and distributed data sources. Due to the heterogeneity of data sources, integration services utilize several types of interfaces to extract data, transport and store integrated data (Martens and Teuteberg, 2012). In addition to our custom interfaces for integrating Web data, we benefit from existing Web APIs[2] and exploit them for accessing different databases on the Web, as well as integrating heterogeneous Web content. Such Web APIs offer several formats of data, e.g., CSS, GeoRSS, HTML, Text, XML, JSON, RDF, etc.

In this section, we begin by distinguishing between different roles of integration services and data services. We then present two complementary service-oriented directional strategies to integrate data in near real-time. Finally, we enumerate the functionalities of management services along with services for integrating data in near real-time, focusing on change detection services.

### 5.1 Data Services as Data Sources

We hereby distinguish between two main different categories of services: integration services and ready-data services. On the one hand, integration services are designed to integrate data and manage different activities of the data integration system. On the other hand, data services are a set of Web services that help share a provider's ready-data with consumers, delivering so-called *Information-as-a-Service*. Nowadays, an increasing amount of enterprises implement their core business and deploy their application services over the Internet. Thus, these data services can be considered as data sources, and we term them *third-party services* in opposition to the internal services of AX-InCoDa. Such third-party services have a direct access to their providers' underlying data. Some common examples of external services include currency exchange, products on stock, news streams, weather reports, and stock quote reports. Finally, beside re-usability of data services, they provide low-cost approach for integrating data from heterogeneous data sources.

### 5.2 Push/Pull Data Integration

In AX-InCoDa, there are two complementary service-oriented directional strategies for integrating and refreshing the AXML repository in near real-time, i.e., *push* and *pull* strategies. The push direction means that relevant data move from data sources toward the AXML repository in a "push" way. In this strategy, beside applying the traditional "initial data integration" using integration services, change detection services play a significant role to incrementally refresh the AXML repository with data in near real-time. Change detection services are employed to capture data changes. Using the subscription-notification paradigm, data integration services can subscribe to change detection services to be notified with only near real-time changes. Figure 5 illustrates the forward direction strategy for pushing data in near real-time onto the AXML repository.

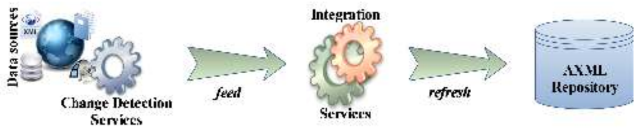

**Fig. 5** Push strategy for refreshing data

By contrast, in the pull direction strategy, the AXML repository is refreshed with data in near real-time by pulling data from sources (read figure 6 from right to left). Pulling data is attained by invoking embedded AXML services. Such embedded services are invoked to refresh the repository with data in near real-time, achieving *on the fly* data integration upon request. The timing of invoking embedded services may be defined explicitly (e.g., triggering temporal events), or implicitly (e.g., when querying AXML documents/fragments). The pull direction strategy for integrating data in near real-time heavily depends on the AXML language's capabilities.

[2] http://www.programmableweb.com/apis/directory/

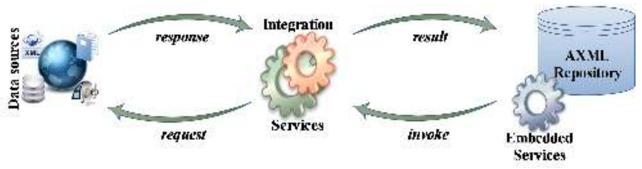

**Fig. 6** Pull strategy for refreshing data

### 5.3 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, AX-InCoDa includes other services for managing integration tasks as follows.

- **Metadata extraction services.** Metadata extraction services are sets of services designed for generating metadata from heterogeneous data sources, including metadata of Web services residing at data sources, so-called third-party services. Metadata are extracted using a library of adapters, each one dealing with a specific type of data sources. Then, metadata extraction processes are deployed as Web services. The metadata extraction process is an interactive process, where the data steward selects relevant metadata from service results. According to the selected metadata, other services are called to retrieve further metadata details. Beside generating metadata automatically using metadata extraction services, some useful semantic metadata can be enriched using a GUI, such as source description, delimiters of textual sources, etc.
- **Change detection services.** Change detection services are sets of services that monitor relevant data sources for changes. We distinguish between two types of change events, namely "structure changes" and "content changes". Structure changes are related to the structure of sources. Content changes are 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 changes. Only relevant changes are taken into consideration to incrementally update integrated data. Section 5.4 comes up with further details on how change detection services work.
- **Refreshing services.** Refreshing services are sets of services designed for incrementally updating integrated data with data changes, detected by change detection services. Therefore, these services act as intermediate services between change detection services and data integration services. Moreover, such services follow the scheduling plan for refreshing data integration.
- **Logging services.** Logging services are sets of services that log description information about different framework activities into the event repository. These services are attached to all framework modules, dealing with message catalogs for each module, and then logging the occurred events into the event log repository using an XML streaming interface. One challenge that faces logging services is where to physically locate logged events, i.e., where to locate the event repository. This challenge is raised because AX-InCoDa is a service-oriented framework, which is distributed across multiple systems. But since the framework mines and analyzes logged events for self-tuning and self managing purposes, the best location for logged events in the framework on the server that performs mining services.
- **Mining services.** Mining services are sets of services that extract interesting knowledge from logged events. Mining services are significant services that are continuously invoked by other management services, such as refreshing and reactive services. Mining logged events heavily depends on handling and structuring incremental logged events in an innovative way (section 6), to be mined easily. Moreover, such services can be scheduled to be explicitly invoked to self-configure the framework's setting. It is worth noting that mined information is evaluated against sets of defined active rules, which are managed by reactive services. In addition, mining services use XQuery to extract the relevant knowledge from the innovative new structure of logged events.
- **Querying services.** Querying services are a set of services that query either metadata by other integration management services or query integrated data by the business user. Thus, the first type of queries is carried out implicitly, without any user intervention. The second type of queries is explicitly formulated by business users or an external analysis module. Queried metadata and AXML data are both in XML format. Therefore, querying services use XQuery to query both the metadata and the AXML repository. XQuery is presented as an integral part of the Web service infrastructure (Onose and Siméon, 2004).
- **Reactive services.** Reactive services are sets of services that apply the ECA paradigm to respond to encountered events with appropriate actions. Reactive services are able to manage active rules through detecting events, evaluating conditions and invoking actions. Therefore, such services can drive the behavior of the framework's activities. Such services

can enhance the framework's capabilities, providing a mechanism that can be used to support the interaction among different framework modules. For instance, these services can be used to update the integration schema, validate integrity constraints, and coordinate the workflow of services. Implementing reactive rules is mainly based on XQuery triggers, or Active XQuery (Bonifati et al., 2002a).
– **Utility services.** Utility services include services to maintain the system. They include services to configure different settings of the framework such as the frequency of explicit invocation of AXML embedded services, the frequency of executing change detection services, the scheduling of integration services, the repositories settings, the Web service security protocol, etc. Utility also include versioning services to deal with different versions of loaded data. Finally, they include services to handle errors.

### 5.4 Change Detection Services

#### *5.4.1 Back-end services*

The common methods for capturing near real-time data changes include applying triggers in relational or XML databases, adding time-stamp attributes to database records, scanning log files of transactions (log sniffing), and/or monitoring transactions for changes and then targeting these changes to a specific message queue. These methods run on the back-end layer of data sources. In AX-InCoDa, such methods are developed and exposed as services, called back-end services. The data steward defines the appropriate method for detecting changes from each data source as part of metadata, i.e., metadata enrichment. Then, s/he maps data sources to change detection services, and adapts scheduling configurations for change detection. Scheduling configurations can be different from one data source to another. For instance, changes are detected implicitly when there is a possibility to apply triggers, and changes are detected recursively with more or less frequent intervals according to changing nature of data source. Notice that mining logged events can be employed to adapt the scheduling configuration automatically.

#### *5.4.2 Front-end services*

Since AX-InCoDa integrates data from Web data sources, we can assume that data source modifications (e.g., insert, update, or delete contents of Web data sources) can be conducted through a set of services. Such services are installed at data sources' as interface for modified data entries. The same modifiable services can be subscribed by multiple consumers (i.e., data integration services). These services then notify the consumers, in near real-time, with only relevant data changes according to the subscription policy. This method of wrapping (or enveloping) data sources with services can be considered as a choice for capturing data changes. In practice, this choice requires modifying the design and implementation of data sources. Thus, this choice is not always guaranteed to be applied for different data sources where modifying data source design is not allowed.

#### *5.4.3 Subscription-notification mechanism*

Publish-subscribe and/or event-notification mechanisms (Zhao and Liu, 2006) can play an important role in integrating near real-time data changes. This mechanism is applied in AX-InCoDa to integrate near real-time changes. There is usually two components in the distributed architectures that use publish-subscribe or event-notification mechanisms, i.e., event sources and event consumers. In AX-InCoDa, the Web services designed to monitor different data sources for changes are considered as event sources. Integration services, particularly extraction services and refreshing services, are considered as event consumers. Change detection services define change events and publish them whenever a change occurs using the PUBLISH operation. A particular integration service can subscribe to one or more interesting change events using the SUBSCRIBE operation identifying the event type to be subscribed. In addition, change events can also be subscribed by one or more of integration services. Such integration services are notified using the NOTIFY operation when their subscribed events occur, in other words when published events are matched with subscribed events. Figure 7 illustrates change detection workflow with the publish-subscribe-notify mechanism.

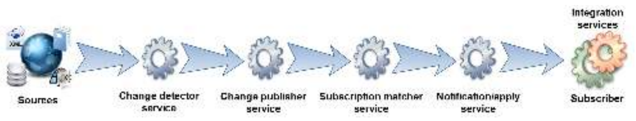

**Fig. 7** Publish-subscribe-notify mechanism for detecting changes

## 6 Event-driven Reactive Integration

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

### 6.1 Event Warehousing

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 adding, altering, and/or dropping data sources, objects, and/or fields. Nevertheless, examples of content changes events include data manipulation events such as inserting, updating, and/or deleting values at a 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 and Ross, 2002), where a single fact document represents all events (figure 8). However, there are multiple dimension documents (e.g., user dimension, machine dimension, session dimension, data source dimension, AXML document dimension, date and time dimension). Warehousing events allows data steward to apply XML On-line Analytical Processing (X-OLAP) (Park et al., 2005), in order to explore and analyze logged events information.

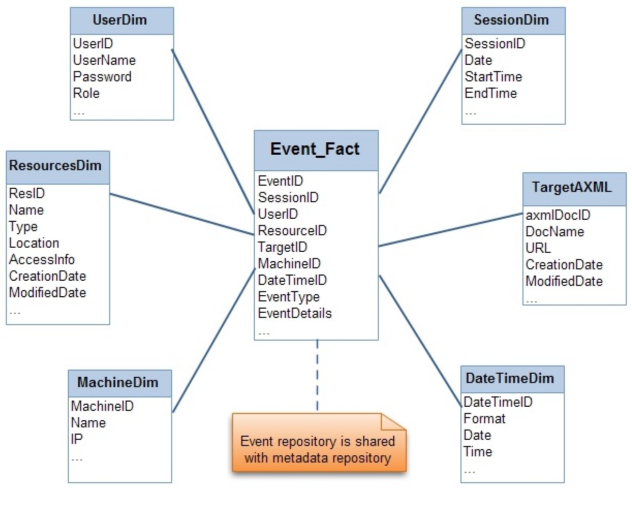

**Fig. 8** Logical star schema of event repository

### 6.2 Event Mining

#### *6.2.1 Incremental event mining*

AX-InCoDa logs a large amount of events incrementally and continuously in XML format. Logged events are essential to understand the activities of AX-InCoDa. 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.

Mining logged events is the process of extracting knowledge from continuously and rapidly logged events. Extracting interesting knowledge from logged events is an emerging need for auditing, maintaining, and self-managing systems. One relevant data mining technique in this context is association rule mining. Association rule mining discovers interesting association and/or correlation relationships among large sets of logged events, and predicts upcoming events based on the occurrence of previous ones. Mining association rules from incremental XML-formatted logged events is different from mining traditional static data, due to several specific issues and challenges either related to data arrival (Gaber et al., 2005; Jiang and Gruenwald, 2006; Yu and Chi, 2009), or the nature of the XML format (Feng and Dillon, 2004; Zhao et al., 2006). When logging events, events arrive continuously at moderate or high speed, in unbounded amount, and with varying data distributions. Unlike in traditional data mining, there is not enough time to rescan the whole database whenever an update occurs. Therefore, a single-pass over events is required. Logged events need to be processed incrementally as fast as possible. Processing speed should be faster than event arrival rate. Moreover, mined data should not need to be recalculated each time they are requested. The unbounded amount of logged events and limited system resources, such as disk storage, memory usage, and CPU power, lead to the need for event mining algorithms that adapt themselves to available resources, otherwise result accuracy decreases. Moreover, traditional data mining techniques mine frequent itemsets and discard non-frequent itemsets. This property is not valid for logged events, where the frequency of itemsets is changing over time. Finally, extracting knowledge from XML data is more difficult than on relational data, because of the flexible, irregular, and semi-structured nature of XML.

To the best of our knowledge, there is no XML-based algorithm in the literature to discover interesting knowledge from incremental XML-formatted logged events. Therefore, we propose an incremental algorithm for this purpose. Our algorithm is composed of two main phases. Firstly, we construct and populate a novel tree structure, i.e., FXT, which stores the frequencies of events to be mined. Secondly, we query frequent event-sets and association rules efficiently from the constructed FXT using XQuery. Our algorithm handles most logged event processing issues. It does a single-pass over data transactions to construct the compact FXT structure. Although the FXT is processed using XML technologies and constructed in XML format, its construction time is fast enough. Association rules with different minimum supports are queried at any time without re-constructing the FXT from scratch.

### *6.2.2 Event log sample*

As an example, let us address an e-commerce support system; one of its functionalities is to provide price comparisons. It extracts product data from different merchants, compares their prices and finally reports the best price. The system logs events that usually include much descriptive information about each activity, e.g., event identification, event occurring time, event source, event description, event origin activity, etc. The event origin activity helps formulate dynamic active rules (section 6.3) and their values are either event, condition or action. Note that the more event information is logged, the more interesting knowledge can be discovered. Events with origin activity "event" usually correspond to data modification operations (e.g., evt:addProduct, evt:modifyProduct, evt:deleteProduct, evt:addMerchant, evt:modifyMerchant, evt:deleteMerchant, etc.). Other events with origin activity "condition" usually denote Boolean queries (e.g., cnd:checkMerchant, cnd:checkProduct, cnd:checkPrice, cnd:checkInStock, etc.). Moreover, events with origin activity "action" are usually executing integration services, sending or notifying messages (e.g., act:getProductInfo, act:getMerchantInfo, act:getProductPrice, act:comparePrices, act:sendMessage, act:notifySubscriber, etc.). The aforementioned events and integration services are provided from application-oriented perspective. However, there are metadata-based services that can also be invoked (e.g., accessDeepDB, retrieveDeepDB, retrieveXMLSchema, retrieveHTMLStructure, etc.). In order to apply our incremental event mining algorithm, we need to pre-process logged events (figure 9) and organize them into transactions (figure 10).

Each transaction has an identifier, an occurring time, and a set of items that represent framework events. The set of events are assumed to be logged in a window-size of time (time + window). The corresponding format of logged transactions can be obtained from AX-InCoDa by applying a pre-processing step for defining transactions according to windows of time. The most impor-

```
<events>
 <event id="A">
  <dateTime>2012-02-14 09:16:03</dateTime>
  <category>source structure</category>
  <sourceID>s010</sourceID>
  <userID>user001</userID>
  <evtName>modifyProduct</evtName>
  <originAcitivity>event</originAcitivity>
 </event>
 <event id="B">
  <dateTime>2012-02-14 09:16:03</dateTime>
  <category>source structure</category>
  <sourceID>s010</sourceID>
  <userID>user001</userID>
  <evtName>addMerchant</evtName>
  <originAcitivity>event</originAcitivity>
 </event>
 <event id="C">
  <dateTime>2012-02-14 09:16:09</dateTime>
  <category>integration services</category>
  <sourceID>s010</sourceID>
  <userID>user001</userID>
  <evtName>getProductPrice</evtName>
  <originAcitivity>action</originAcitivity>
 </event>
 <event id="D">
  <dateTime>2012-02-14 09:16:15</dateTime>
  <category>integration services</category>
  <sourceID>s010</sourceID>
  <userID>user001</userID>
  <evtName>comparePrices</evtName>
  <originAcitivity>action</originAcitivity>
 </event>
 <!-- more events -->
</events>
```

**Fig. 9** Logged event sample

```
<transaction id="1" time="2012-02-14 09:16:00">
 <item>A</item><item>B</item><item>C</item><item>D</item>
</transaction>
<transaction id="2" time="2012-02-14 09:16:20">
 <item>C</item><item>E</item>
</transaction>
<transaction id="3" time="2012-02-14 09:16:40">
 <item>B</item><item>C</item>
</transaction>
<transaction id="4" time="2012-02-14 09:17:00">
 <item>C</item><item>D</item><item>E</item>
</transaction>
<transaction id="5" time="2012-02-14 09:17:20">
 <item>B</item><item>C</item><item>D</item>
</transaction>
<transaction id="6" time="2012-02-14 09:17:40">
 <item>A</item><item>C</item><item>E</item>
</transaction>
<transaction id="7" time="2012-02-14 09:18:00">
 <item>A</item><item>B</item><item>D</item>
</transaction>
<transaction id="8" time="2012-02-14 09:18:20">
 <item>E</item><item>F</item>
</transaction>
```

**Fig. 10** Sample event transactions

tant thing to our algorithm is to define the listing of items of each transaction, which should be sorted alphabetically for performance purposes.

### *6.2.3 Frequency XML-based Tree (FXT)*

In order to mine frequent itemsets or association rules, the frequency of events (or items) needs to be calculated. Hence, we propose a novel tree structure named Frequency XML-based Tree (FXT), which contains the frequency of all logged items. Before introducing our FXT structure, let us define $\mathcal{I}$ as a set of items ($\mathcal{I} = \{I_a, I_b, ...., I_z\}$) where letters (*a, b* and *z*) refer to item ordering. In addition, we denote $N_{trans}$ the total number of transactions, $N_k$ the count of any item $k$ across all logged transactions, and $N_{v|...|k}$ $\forall v, k \in [a, z]$ is the conditional count of $I_v$ given $I_k$ descendants. There are some facts can be deduced from the FXT structure, i.e., $N_{trans} \geq N_k$ and $N_k \geq N_{v|...|k}$.

FXT nodes, except the root node, consist of two entries: *item name* and *counter*, where *item name* registers what item this node represents (e.g., $I_m$), and *counter* registers the number of transactions represented by the portion of the path reaching this node (e.g., $N_m$ or $N_{m|...|i}$). FXT is composed of three main levels of nodes (figure 11). Firstly, the *Root node* represents the total number of logged transactions ($N_{trans}$). This root counter is used to calculate any item support. Secondly, the *Breadth nodes* level refers to all children of the root. It represents the count of each item in any logged transaction across all transactions (e.g., $N_i$). Note that the occurrence frequency of an item is the number of transactions that contain the item. This is also known, simply, as the frequency, support, or count of the item (Han and Kamber, 2005). In this paper, *frequency* refers to the number of occurrences of an item, and any *item* support can be easily calculated via dividing item frequency/counter by $N_{trans}$. Thirdly, the *Depth nodes* level refers to all root's grandchildren nodes. It represents the relative or conditional count of a specific item given other related items (e.g., $N_{m|...|i}$). Depth nodes are represented as sets of paths, each path corresponding to specific transaction itemsets. In figure 11, straight lines annotated by a single slash "/" mean that breadth nodes are children of the root node. Dashed lines annotated by double slashes "//" mean that there is zero or more descendant nodes in a specific depth path. Further details on how the algorithm constructs the FXT, and the performance study of the algorithm are discussed in Salem et al. (2011). Figure 12 shows the FXT constructed from the sample transactions given in figure 10.

Finally, the constructed FXT in XML format is shown in figure 13.

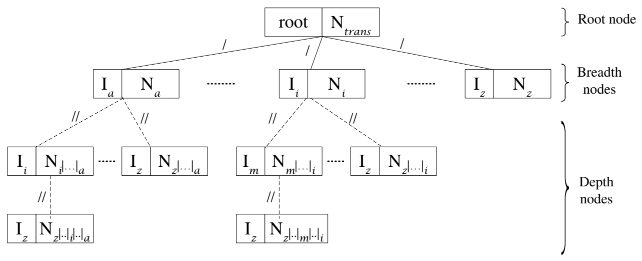

**Fig. 11** FXT structure

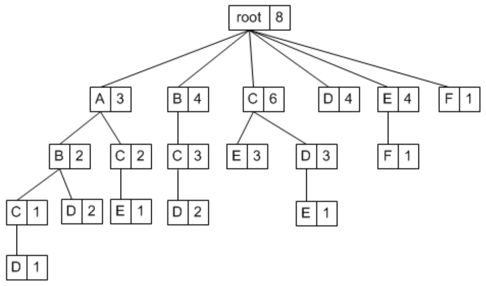

**Fig. 12** Sample FXT

```
<? 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. 13** Output FXT document

### *6.2.4 Discovering frequent events*

The main objective of constructing the FXT is to mine frequent itemsets and association rules easily using the XQuery language[3]. Frequent itemsets are queried by traversing the FXT from breadth nodes to specific nodes (portion of paths), or to leaf nodes (complete paths). Frequent itemsets are usually filtered using a statistical measure called *support*. Support measures the proportion of transactions that contains a specific item (or itemset). A frequent itemset is an itemset whose support is greater than some user-specified minimum support. Frequent itemsets 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 and Srikant, 1994).

Let $\mathcal{I}$ be a set of items within a set of transactions $\mathcal{S}$. Transactions $\mathcal{T}(\mathcal{T} \in \mathcal{S})$ contain the itemset $X(X \subset \mathcal{T})$. In addition, $Y$ and $Z$ are itemsets, where $(Y \subset \mathcal{T})$ and $(Z \subset \mathcal{T})$. Support of itemsets $X$ and $Y$ measures the proportion of transactions in $\mathcal{S}$ that contain both items in $X$ and $Y$. Support of any item can be obtained by querying the counter attribute value of the specified item at breadth nodes, then dividing it by $N_{trans}$. Support of frequent itemsets can be obtained by querying the counter attribute value of the last ordering item of an interesting portion of path, or the leaf item of the interesting path in depth nodes, then dividing this value by $N_{trans}$. Formulas for calculating the support of sample items and itemset from the FXT follow.

$$\begin{aligned} Support(X) &= \frac{root/X/@counter}{root/counter} \\ Support(X,Y,Z) &= \frac{count(X \cup Y \cup Z)}{N_{trans}} \\ &= \frac{root/X/Y/Z/@counter}{root/counter} \end{aligned}$$

Figure 14 introduces the function for generating frequent itemsets from the FXT of figure 13.

The result of calling function getFrequentItemsets on document tree.xml from figure 13 with minimum support=0.25 is shown in figure 15.

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

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

let $path := concat($parent,'/',name($element))
 where $element/@counter div $rootCounter>=$minSupport
return
 (<frequent path="{$path}" count="{$element/@counter}"
  support="{$element/@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. 14** XQuery function for mining frequent itemsets

```
<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. 15** Result of XQuery function for mining frequent itemsets

### *6.2.5 Discovering association rules*

Association rules have been first introduced in the context of retail transaction databases (Agrawal and Srikant, 1994). An association rule is an implication of the form $X \Rightarrow Y$, where $X$ is the *rule body*, $Y$ is the *rule head* and $X \cap Y = \phi$. A rule $X \Rightarrow Y$ states that the transactions $\mathcal{T}$ that contain the items in $X$ are likely to also contain the items in $Y$. Beside *support*, association rules are characterized by *confidence*. Confidence measures the proportion of transactions in $\mathcal{S}$ 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:

$$\begin{aligned} Support(X \Rightarrow Y) &= \frac{count(X \cup Y)}{N_{trans}} \\ &= \frac{root/X/Y/@counter}{root/counter}, \qquad (1) \\ Confidence(X \Rightarrow Y) &= \frac{support(X \Rightarrow Y)}{support(X)} \\ &= \frac{count(X \cup Y)}{count(X)} \\ &= \frac{root/X/Y/@counter}{root/X/@counter}. \qquad (2) \end{aligned}$$

Figure 16 introduces the XQuery function for generating a set of association rules related to a given event from the FXT constructed in figure 15.

The result of calling function getAssRules on document tree.xml from figure 13 is shown in figure 17.

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

```
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="{$y_given_x}"
   sup="{$supp_xy}" confidence="{$confidence}"/>)

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

**Fig. 16** XQuery function for mining association rules

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

**Fig. 17** Result of XQuery function for mining association rules

## 6.3 Active rules

Active rules are incorporated into AX-InCoDa to enrich it 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. Events are either simple or composite. Simple events (also called, primitive or atomic events) correspond to a data modification operation, the execution of an integration service, or the query of 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)*. 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 a condition and executing an action can be logged as events. Although the general form of active rules conform to the ECA paradigm, 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. The omission of the event part leads to a Condition-Action rule (CA-rule), where the compiler generates the event definition. The omission of both the event and condition parts is not allowed for the same rule.

To formulate active rules, there are two types of active rule definitions, i.e., *static active rules* and *dynamic active rules*. Static active rules are defined by the data steward, and their definitions are explicitly changed via the data steward's intervention. Dynamic active rules are defined by the data steward using sets of querying expressions on different components of the rule. Dynamic active rule definitions are implicitly changed from time to time according to the result of embedded queries, as shown in the following example.

In this example, the active rule is formulated dynamically by mining frequent events and association rules and by embedding AXML services into the ECA paradigm (figure 18). We herein assume that the identifiers of different events, conditions and actions are still the same when they are logged as events, for simplicity. For instance, if an action execution (e.g., with identifier $\mathcal{C}$) is logged, its logged event identifier remains $\mathcal{C}$. We are also interested in formulating rules for frequent events. Event frequency is checked using the isFrequent function (figure 19), e.g., with a minimum support equal to 0.05. The getAssRules function is called to get the associated events of the detected one (figure 16), e.g., with a minimum support equal to 0.1 and a minimum confidence equal to 0.65. Then, the origins of detected and associated events are tested to formulate the active rule. In the action part, the confidence between detected and associated events is set. For instance, if confidence is equal to 1, the rule is formulated immediately and saved into the rule repository. The user is notified only if confidence is greater than 0.65. The action is always invoking an AXML service.

## 6.4 Active Active XML (A$^2$XML) Engine

We utilize active (or ECA) rules to activate AXML service invocation. We merge the ECA paradigm into the AXML language to control the flow behavior of the data integration environment. We term the merging of both technologies Active Active XML (for short A$^2$XML). The A$^2$XML engine plays an important role to control the evaluation of the active rules. It manages event triggering, evaluates conditions, and executes actions. The A$^2$XML engine manages the evaluation of integration services. All integration services are registered to the A$^2$XML engine. It is also supplied with business rules to handle the flow and timely 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 service

```
<eca:rules>
 <eca:variable name="eventID" lang="xpath" expr="./@ID"/>
 {
 if local:isFrequent($eventID,0.05)
  {
   for $rule in local:getAssRules($eventID, 0.1, 0.65)
   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>
  }
 else()
 }
</eca:rules>
```

**Fig. 18** Formulating an active (ECA) rule with A$^2$XML

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

declare function local:isFrequent($element as element(*,
 xs:untyped), $min_supp as xs:decimal) as xs:boolean  {
 let $path := $input/*[name(.) = $element]
 let $evt_support := $element/@counter div $rootCounter
return
 boolean ($evt_support >= $$min_supp)
};
```

**Fig. 19** XQuery function for checking event frequency

results are written, either replacing the calling service or appending to it.

## 7 Implementation and Validation

### 7.1 Implementation Issues

We implemented our framework prototype mostly using standard open-source software. It is implemented as a Web-based application[4] using Oracle, XML and Java technologies. Figure 20 shows the deployment diagram, which visualizes the hardware, middle-ware and software used in our implementation. The diagram is composed of multiple tiers: data source, integration management, Web, client and target tiers. The tiers communicate via several protocols and interfaces. The data source tier enables accessing, defining, updating and managing data sources. There are Web APIs and Java APIs that we exploit in AX-InCoDa to facilitate extracting metadata schema from heterogeneous data sources. Relevant metadata schema of data sources can be easily determined via the application's GUI. Recall that AX-InCoDa integrates complex data including Web data from heterogeneous and distributed data sources. Examples of supported data sources include database connections (e.g., JDBC and ODBC), relational database files, spreadsheets, CSV, XML, Web services, text, images, audios and videos.

The integration management tier is the main core of our application. It is composed of two main sets of cooperative components: application manager and native XML database (management repository). The integration manager components (i.e., change detection, ETL and A$^2$XML engines) are implemented using Java. From an implementation viewpoint, using an open source tool for performing ETL tasks saves implementation time. Our prototype utilizes the Java library of *Pentaho Data Integration* (PDI)[5]. 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. The XML file of a specific transformation involves metadata about the input data source and its detailed characteristics, metadata about transformation rules, and metadata about output targets. Such metadata describe the ETL operation. The XML database is implemented using *Sedna*[6], which supports XQuery triggers. It involves several collections of databases, such as metadata of framework components, defined and mined

[4] `http://eric.univ-lyon2.fr/~rsalem/axdi/`
[5] `http://kettle.pentaho.com/`
[6] `http://www.modis.ispras.ru/sedna/`

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.

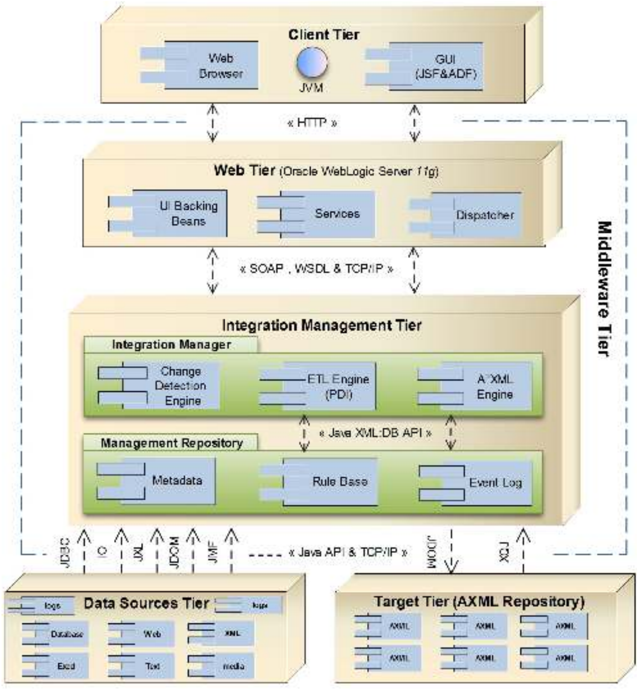

**Fig. 20** Deployment diagram of our Web platform

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 interesting data sources, and then extraction services bring out the metadata from which to select a relevant schema. Beside defining data sources via the 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 querying the AXML repository, browsing changes of data sources, and defining ECA rules. The GUI also allows applying the general settings of 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 the native XML database.

## 7.2 Validation Example

To validate our framework prototype, we reconsider the product pricing comparison example (section 6.2.2). Fortunately, there are several Web APIs that gather data from thousands of on-line stores, e.g., DataFeedFile.com API, Amazon Product Advertising API, Shopping.com (eBay) API, Shopzilla API, PriceGrabber API, etc. Such APIs enable the data steward searching the product using one of several criteria, e.g., keywords, sku, upc or ISBN number. All these APIs can retrieve product information including the price in XML format. In our prototype, we utilize such APIs to extract product specifications, their merchants and their prices. Additionally, we enrich the prototype with services for retrieving customer reviews, images and videos of products. We develop Web services to compare prices from different merchants returning the lowest and highest prices. We also apply services exploiting DiffX[7] to compare resulting XML data and detect event of changes. According to change events, the range of prices may be changed from time to time. Our prototype supports multiple adaptive strategies for refreshing the repository with recent data either by change detection services or invoking embedded AXML services.

In this example, we integrate product data from a variety of on-line stores using the aforementioned Web APIs and store these data into AXML documents. Figure 21 illustrates the target AXML document layout, wherein hollow tag indicators refer to static elements and solid tag indicators refer to service call (dynamic) elements. Contents of AXML static elements are rarely updated (e.g., product description, merchant logo and merchant URL), while contents of dynamic elements (i.e., call to services) are updated frequently, e.g., lowest price, highest price and quantity in stock. A sample of full target AXML document is provided in appendix A, which shows the AXML document before and after invoking embedded services, respectively.

## 7.3 Implementation Remarks

We herein discuss some remarks on the qualities of our prototype implementation, e.g., maintainability, reusability, interoperability, scalability and performance. AX-InCoDa is a metadata-driven framework that allows a data steward to specify what data need to be integrated, without regarding how to integrate them. Accordingly, it makes easy for the steward to maintain the data integration environment for any application. How-

[7] http://www.topologi.com/diffx/

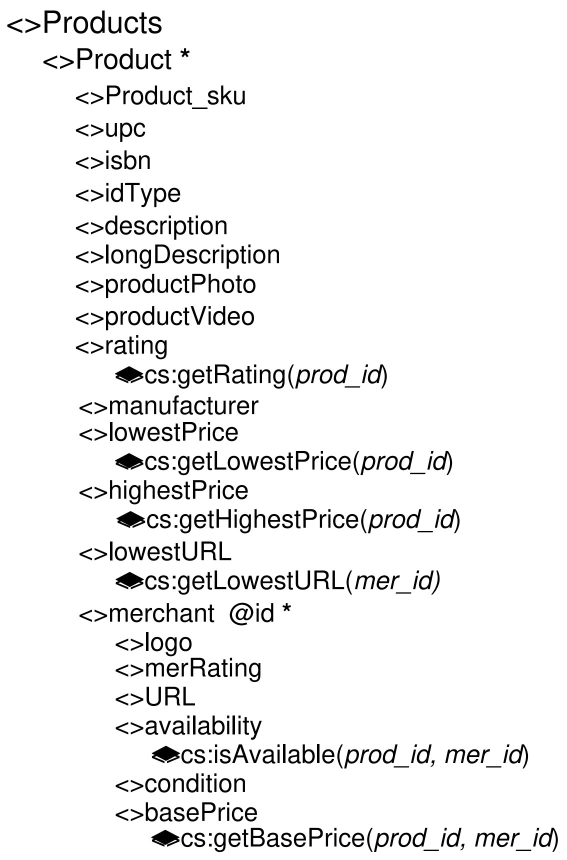

**Fig. 21** AXML document layout

ever, a steward must still be a person who has enough expertise in data integration and its processes. Furthermore, our prototype is service-oriented, and manages its processes and integration tasks using the services technology. These services are designed to be reused by different applications. For instance, a specific service can be reused to integrate the metadata of any database object. Moreover, employing both XML and Web services in developing our framework tackles interoperability problems among diverse systems. Since the entire framework is services-oriented, it can also benefit from the myriad of options offered by WS-Extensions. For instance, the framework can be augmented with security and reliability services. It is advantageous to use open-source and free software/pieces of software for implementing our prototype. However, the scalability and performance of our prototype is related to that of underlying technologies, such as Sedna, PDI, and the WebLogic server infrastructure. Note that there are other alternative technologies that may enhance performance. Nevertheless, our choice of technologies bases on their compatibility and satisfaction of our requirements. For instance, oracle ADF components (user interface components) run smoothly with the WebLogic server. Finally, we conclude that our prototype is implemented efficiently enough to validate the concepts of integrating data from complex data sources in real-time and autonomously.

## 8 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 Web-based DW/BI applications. Our framework addresses three main issues of complex Web data integration: 1) handling complexity issues of Web data; 2) satisfying near real-time data integration; and 3) achieving autonomous and reactive data integration. Our framework noticeably tackles limitations of traditional data integration systems by 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 Web standards in data integration helps integrate 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 responsible for detecting, capturing, and notifying only real-time data changes. Furthermore, logged events in our framework are mined to discover interesting knowledge to help self-maintain, self-configure, and automate data integration tasks. Active rules are also enriched by 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[4].

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 and exploit argumentative reasoning to realize autonomous semantic mediation between data sources, integration services and target data repositories (Janjua et al., 2012). Therefore, we can ensure achieving a more automated data integration workflow. Although the XML-formatted event mining approach proposed in this paper is feasible and efficient, the actual exploitation of the knowledge we mine in the integration process, and the evaluation of its efficiency, currently remains a perspective of our work. 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 framework 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, and a critical one when data come from the Web.

**Acknowledgements** The authors thank the anonymous reviewers of this paper for their thoughtful comments, which greatly helped improving our present work.

## 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, IEEE Computer Society, Washington, DC, USA, WISE '02, pp. 3–6.

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 .

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

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

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

Bailey J., Poulovassilis A., Wood P. T. (2002). An event-condition-action language for XML. In: The 12th International World Wide Web Conference, WWW, Hawaaii, pp. 486–495.

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.

Bentayeb F., Maiz N., Mahboubi H., Favre C., Loudcher S., Harbi N., Boussaid O., Darmont J. (2011). Innovative Approaches for efficiently Warehousing Complex Data from the Web, Business Science Reference, pp. 26–52. Business Intelligence Applications and the Web: Models, Systems and Technologies, m. Zorrilla, J. Mazón, Ó. Ferràndez, I. Garrigós, F. Daniel, J. Trujillo, Eds.

Bhowmick S. S., Madria S. K., Ng W. K. (2003). Web Data Management: A Warehouse Approach. Springer-Verlag New York Inc.

Bonifati A., Braga D., Campi A., Ceri S. (2002a). Active XQuery. In: Proceedings of the 18th International Conference on Data Engineering(ICDE'02), San Jose, CA, p. 403.

Bonifati A., Ceri S., Paraboschi S. (2002b). 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.

Boussaid O., Darmont J., Bentayeb F., Loudcher S. (2008). Warehousing complex data from the web. International Journal of Web Engineering and Technology 4:408–433.

Brobst S., Ballinger C. (2003). Active data warehousing: why Teradata warehouse is the only proven platform. NCR Teradata, white paper URL `http://whitepapers.zdnet.co.uk/`.

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., Boussaïd O. (2006). Processing and managing complex data for decision support. Idea Group Inc (IGI).

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.

Erl T. (2004). Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services. Prentice Hall.

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.

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

Han J., Kamber M. (2005). Data Mining: Concepts and Techniques, Second Edition. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA.

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. (2002). Building the data warehouse, Second Edition. New York: John Wiley & Sons.

Janjua N., Hussain F., Hussain O. (2012). Semantic information and knowledge integration through argumentative reasoning to support intelligent decision making. Information Systems Frontiers pp. 1–26, URL http://dx.doi.org/10.1007/s10796-012-9365-x, 10.1007/s10796-012-9365-x.

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

Karakasidis A., Vassiliadis P., Pitoura E. (2005). ETL queues for active data warehousing. In: Proceedings of 2nd international workshop on Information Quality in Information Systems (IQIS'05), Baltimore, USA, pp. 28–39.

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

Kimball R., Ross M. (2002). The data warehouse toolkit: The complete guide to dimensional modeling, Second Edition. New York: 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.

Li G., Wei M. (2012). Everything-as-a-service platform for on-demand virtual enterprises. Information Systems Frontiers pp. 1–18, URL http://dx.doi.org/10.1007/s10796-012-9351-3, 10.1007/s10796-012-9351-3.

Linthicum D. S. (2010). Approaching SaaS integration with data integration best practices and technology. White paper URL http://www.informaticacloud.com/images/whitepapers/WP-Approaching_SaaS_Integration.pdf.

Lorenzo G. D., Hacid H., Paik H.-Y., Benatallah B. (2009). Data integration in mashups. SIGMOD Record 38(1):59–66.

Madhavan J., Cohen S., Dong X. L., Halevy A. Y., Jeffery S. R., Ko D., Yu C. (2007). Web-scale data integration: You can afford to pay as you go. In: CIDR, www.crdrdb.org, pp. 342–350.

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.

Martens B., Teuteberg F. (2012). Decision-making in cloud computing environments: A cost and risk based approach. Information Systems Frontiers 14:871–893, URL http://dx.doi.org/10.1007/s10796-011-9317-x, 10.1007/s10796-011-9317-x.

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.

Naeem M., Dobbie G., Weber G. (2011). X-hybridjoin for near-real-time data warehousing. In: Fernandes A., Gray A., Belhajjame K. (eds.) Advances in Databases, Lecture Notes in Computer Science, vol. 7051, Springer Berlin / Heidelberg, pp. 33–47.

Naeem M. A., Dobbie G., Webber G. (2008). An event-based near real-time data integration architecture. In: Proc. 12th Enterprise Distributed Object Computing Conf. Workshops, pp. 401–404.

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.

Onose N., Siméon J. (2004). XQuery at your web service. In: Feldman S. I., Uretsky M., Najork M., Wills C. E. (eds.) WWW, ACM, pp. 603–611, URL http://doi.acm.org/10.1145/988672.988754.

Oracle W. P. (2010). Real-time data integration for data warehousing and operational business intelligence. Oracle White Paper p. 17, URL http://www.oracle.com/us/products/middleware/data-integration/goldengate11g-realtimedw-wp-168215.pdf.

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.

Pérez J. M., Llavori R. B., Aramburu M. J., Pedersen T. B. (2008). Integrating data warehouses with web data: A survey. IEEE Trans Knowl Data Eng 20(7):940–955.

Phan B., Pardede E., Rahayu W. (2012). On the improvement of active xml (axml) representation and query evaluation. Information Systems Frontiers pp. 1–20, URL http://dx.doi.org/10.1007/s10796-012-9363-z, 10.1007/s10796-012-9363-z.

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

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

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., Boussaïd O., Darmont J. (2010). Conceptual workflow for complex data integration using AXML. In: International Conference on Machine and Web Intelligence (ICMWI 10), Algiers, Algeria.

Salem R., Darmont J., Boussaïd O. (2011). Efficient incremental breadth-depth xml event mining. In: 15th International Database Engineering & Applications Symposium (IDEAS'11), Lisbon, Portugal, ACM.

Schlesinger L., Irmert F., Lehner W. (2005). Supporting the ETL-process by web service technologies. Int J of Web and Grid Services 1:31–47.

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.

Thor A., Rahm E. (2011). Cloudfuice: A flexible cloud-based data integration system. In: Auer S., Díaz O., Papadopoulos G. A. (eds.) ICWE, Springer, Lecture Notes in Computer Science, vol. 6757, pp. 304–318.

Utomo W. H. (2011). B2B integration based on SOA using web service. Foundation of Computer Science (FCS) (2).

Vassiliadis P., Simitsis A. (2009). Near real time etl. In: Kozielski S., Wrembel R. (eds.) New Trends in Data Warehousing and Data Analysis, Annals of Information Systems, vol. 3, Springer US, pp. 1–31.

Vidal V., Lemos F., Feitosa 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.

William Inmon G. N. Derek Strauss (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 B., Liu C. (2006). Efficient SIP-specific event notification. In: ICN/ICONS/MCL, IEEE Computer Society, p. 1, URL http://doi.ieeecomputersociety.org/10.1109/ICNICONSMCL.2006.85.

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: Zhu et al. (2004), pp. 262–269.

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

## Appendix A

```xml
<products xmlns:axml="http://axml.net" xmlns:cs="http://axml.net/call_service">
 <product>
  <product_sku/>
  <upc>8806071418728</upc>
  <isbn/>
  <idType>upc</idType>
  <category>smartphones</category>
  <description>Samsung i9100 Galaxy S II - BLACK </description>
  <longDescription>Samsung i9100 Galaxy S II Unlocked GSM Smartphone with 8MP Camera,
   Android OS, 16GB Internal Memory, Touchscreen, Wi-Fi, and GPS (Noble Black)
  </longDescription>
  <productPhoto>http://localhost:17101/resources/images/products/smartphones/samsung-galaxy-S2-black.png</productPhoto>
  <productVideo>http://localhost:17101/resources/videos/products/smartphones/samsung-galaxy-S2-black.mov</productVideo>
  <rating>
   <axml:cs serviceURL="http://localhost:17101/services/rating?WSDL"
    serviceName="rating" methodName="getRating" mode="Replace" frequency="86400000">
    <parameters>
     <param name="product_id" value="/upc/text()"/>
    </parameters>
   </axml:cs>
  </rating>
  <manufacturer>Samsung</manufacturer>
  <lowestPrice>
   <axml:cs serviceURL="http://localhost:17101/services/pricing?WSDL"
    serviceName="pricing" methodName="getLowestPrice" mode="Replace" frequency="3600000">
    <parameters>
     <param name="product_id" value="/upc/text()"/>
    </parameters>
   </axml:cs>
  </lowestPrice>
  <highestPrice>
   <axml:cs serviceURL="http://localhost:17101/services/pricing?WSDL"
    serviceName="pricing" methodName="getHighestPrice" mode="Replace" frequency="3600000">
    <parameters>
     <param name="product_id" value="/upc/text()"/>
    </parameters>
   </axml:cs>
  </highestPrice>
  <lowestURL>
   <axml:cs serviceURL="http://localhost:17101/services/merchants?WSDL"
    serviceName="merchants" methodName="getLowestURL" mode="Replace" frequency="3600000">
    <parameters>
     <param name="merchant_id" value="/merchant[basePrice=/lowestPrice]/@id"/>
    </parameters>
   </axml:cs>
  </lowestURL>
  <merchant id="2973">
   <logo>http://www.amazon.com/ref=gno_logo</logo>
   <merRating>6.7/10</merRating>
   <URL>http://www.amazon.fr/dp/B0052EWH32</URL>
   <availability>
    <axml:cs serviceURL="http://localhost:17101/services/availabilty?WSDL"
     serviceName="availabilty" methodName="isAvailable" mode="Replace" frequency="3600000">
     <parameters>
      <param name="product_id" value="./upc/text()"/>
      <param name="merchant_id" value="/merchant/@id"/>
     </parameters>
    </axml:cs>
   </availability>
   <condition>New</condition>
   <basePrice>
    <axml:cs serviceURL="http://localhost:17101/services/pricing?WSDL"
     serviceName="pricing" methodName="getBasePrice" mode="Replace" frequency="3600000">
     <parameters>
      <param name="product_id" value="./upc/text()"/>
      <param name="merchant_id" value="/merchant/@id"/>
     </parameters>
    </axml:cs>
   </basePrice>
  </merchant>
```

**Fig. 22** Sample AXML document before invoking embedded AXML services (1 of 2)

```
  <merchant id="6538">
   <logo>http://www.pixmania.com/fr/fr/home.html</logo>
   <merRating>6.4/10</merRating>
   <URL>http://www.pixmania.com/fr/fr/11676708/art/samsung/
    i9100g-galaxy-s-ii-androi.html?srcid=17</URL>
   <availability>
    <axml:cs serviceURL="http://localhost:17101/services/availabilty?WSDL"
     serviceName="availabilty" methodName="isAvailable" mode="Replace" frequency="3600000">
     <parameters>
      <param name="product_id" value="./upc/text()"/>
      <param name="merchant_id" value="/merchant/@id"/>
     </parameters>
    </axml:cs>
   </availability>
   <condition>New</condition>
   <basePrice>
    <axml:cs serviceURL="http://localhost:17101/services/pricing?WSDL"
     serviceName="pricing" methodName="getBasePrice" mode="Replace" frequency="3600000">
     <parameters>
      <param name="product_id" value="./upc/text()"/>
      <param name="merchant_id" value="/merchant/@id"/>
     </parameters>
    </axml:cs>
   </basePrice>
  </merchant>
  <merchant id="55119">
   <logo>http://www.darty.com/static/r4u3/img/logo.gif</logo>
   <merRating>7.2/10</merRating>
   <URL>http://www.darty.com/nav/achat/telephonie/telephone_mobile/mobile/
    samsung_galaxy_s_ii_noir.html</URL>
   <availability>
    <axml:cs serviceURL="http://localhost:17101/services/availabilty?WSDL"
     serviceName="availabilty" methodName="isAvailable" mode="Replace" frequency="3600000">
     <parameters>
      <param name="product_id" value="./upc/text()"/>
      <param name="merchant_id" value="/merchant/@id"/>
     </parameters>
    </axml:cs>
   </availability>
   <condition>New</condition>
   <basePrice>
    <axml:cs serviceURL="http://localhost:17101/services/pricing?WSDL"
     serviceName="pricing" methodName="getBasePrice" mode="Replace" frequency="3600000">
     <parameters>
      <param name="product_id" value="./upc/text()"/>
      <param name="merchant_id" value="/merchant/@id"/>
     </parameters>
    </axml:cs>
   </basePrice>
  </merchant>
  <merchant id="53657">
   <logo>http://www.fnac.com/</logo>
   <merRating>5.8/10</merRating>
   <URL>http://www.fnac.com/Samsung-Galaxy-S2-sous-Android-Noir/a4044819/
    w-4?Origin=CMP_CLUBIC</URL>
   <availability>
    <axml:cs serviceURL="http://localhost:17101/services/availabilty?WSDL"
     serviceName="availabilty" methodName="isAvailable" mode="Replace" frequency="3600000">
     <parameters>
      <param name="product_id" value="./upc/text()"/>
      <param name="merchant_id" value="/merchant/@id"/>
     </parameters>
    </axml:cs>
   </availability>
   <condition>New</condition>
   <basePrice>
    <axml:cs serviceURL="http://localhost:17101/services/pricing?WSDL"
     serviceName="pricing" methodName="getBasePrice" mode="Replace" frequency="3600000">
     <parameters>
      <param name="product_id" value="./upc/text()"/>
      <param name="merchant_id" value="/merchant/@id"/>
     </parameters>
    </axml:cs>
   </basePrice>
  </merchant>
 </product>
</products>
```

**Fig. 23** Sample AXML document before invoking embedded AXML services (2 of 2)

```
<products>
 <product>
  <product_sku/>
  <upc>8806071418728</upc>
  <isbn/>
  <idType>upc</idType>
  <category>smartphones</category>
  <description>Samsung i9100 Galaxy S II - BLACK </description>
  <longDescription>Samsung i9100 Galaxy S II Unlocked GSM Smartphone with 8MP Camera,
   Android OS, 16 GB Internal Memory, Touchscreen, Wi-Fi, and GPS (Noble Black)
  </longDescription>
  <productPhoto>http://localhost:17101/resources/images/products/smartphones/
   samsung-galaxy-S2-black.png</productPhoto>
  <productVideo>http://localhost:17101/resources/videos/products/smartphones/
   samsung-galaxy-S2-black.mov</productVideo>
  <rating>8.9/10</rating>
  <manufacturer>Samsung</manufacturer>
  <lowestPrice>465  TTC</lowestPrice>
  <highestPrice>577  TTC</highestPrice>
  <lowestURL>http://www.amazon.fr/dp/B0052EWH32</lowestURL>
  <merchant id="2973">
   <logo>http://www.amazon.com/ref=gno_logo</logo>
   <merRating>6.7/10</merRating>
   <URL>http://www.amazon.fr/dp/B0052EWH32</URL>
   <availability>Yes</availability>
   <condition>New</condition>
   <basePrice>465  TTC</basePrice>
  </merchant>
  <merchant id="6538">
   <logo>http://www.pixmania.com/fr/fr/home.html</logo>
   <merRating>6.4/10</merRating>
   <URL>http://www.pixmania.com/fr/fr/11676708/art/samsung/
    i9100g-galaxy-s-ii-androi.html?srcid=17</URL>
   <availability>Yes</availability>
   <condition>New</condition>
   <basePrice>577  TTC</basePrice>
  </merchant>
  <merchant id="55119">
   <logo>http://www.darty.com/static/r4u3/img/logo.gif</logo>
   <merRating>7.2/10</merRating>
   <URL>http://www.darty.com/nav/achat/telephonie/telephone_mobile/mobile/
    samsung_galaxy_s_ii_noir.html</URL>
   <availability>Yes</availability>
   <condition>New</condition>
   <basePrice> 492  TTC</basePrice>
  </merchant>
  <merchant id="53657">
   <logo>http://www.fnac.com/</logo>
   <merRating>5.8/10</merRating>
   <URL>http://www.fnac.com/Samsung-Galaxy-S2-sous-Android-Noir/a4044819/
    w-4?Origin=CMP_CLUBIC</URL>
   <availability>Yes</availability>
   <condition>New</condition>
   <basePrice> 529  TTC</basePrice>
  </merchant>
 </product>
</products>
```

**Fig. 24** Sample AXML document after invoking embedded AXML services