An Active XML-based Framework for Integrating Complex Data

Rashed Salem, Omar Boussaïd, Jérôme Darmont

An Active XML-based Framework for Integrating Complex Data

Rashed Salem, Omar Boussaïd and Jérôme Darmont
Université de Lyon (ERIC Lyon 2)
5 av. P. Mendès-France, 69676 Bron Cedex, France
{rashed.salem, omar.boussaid, jerome.darmont}@univ-lyon2.fr

ABSTRACT

Data integration is a critical problem in data warehousing and decision-support systems. Traditional data integration systems are very successful in integrating structured data, but structured data represent only a small subset of interesting data that could be warehoused by many enterprises. Current data integration systems also lack of self-managing capabilities. Therefore, we propose a data integration framework for integrating complex data *actively*. The purpose of our framework is twofold. Firstly, it integrates complex data using Web standards into an Active XML (AXML) repository. Secondly, beside warehousing logged events into event repository, it exploits active rules and framework events mining to self-manage, automate and activate different data integration tasks. Finally, we have implemented a prototype framework as a web application.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*Data warehouse and repository*; H.3.5 [**Online Information Services**]: Web-based services

General Terms

Design, Languages, Management

Keywords

Active XML, active integration, active rules, complex data, integration services.

1 Introduction

Data warehousing is a very successful approach for decision-support. Classical data warehousing is well adapted to integrate structured data, but structured data represent only a small subset of interesting data that could be warehoused by many enterprises [9]. Indeed, there are huge volumes of heterogeneous data (e.g., semi-structured/XML, emails, Web, text, and multimedia data) that are available and distributed over networks. Such data can be termed as *complex data*. Data may be qualified as complex if they are: diversely structured, represented in various formats, originating from several different sources, and/or changing in terms of definition or value over time [5]. As "simple" data, complex data must be warehoused into a unified storage to be later analyzed for decision-support purposes. However, the classical warehousing approach [6, 7] is not very adequate when dealing with complex data, particularly in the data integration phase. Therefore, there is an increasing demand to handle the complexity aspects of data sources.

Moreover, traditional data integration systems are based on batch ETL (Extraction, Transformation, and Loading), which is achieved at regular interval (e.g., nightly, weekly or monthly), conforming to a static scheduling plan. ETL processes perform integration tasks passively, and they cannot satisfy real-time requirements [11, 12], specifically when integrating complex data. Nevertheless, there is an increasing need to eliminate (or rather minimize) user intervention in processing complex and routine data integration tasks, to provide event-based autonomous integration systems, and to refresh integrated data dynamically, in real-time.

In this paper, we are motivated to tackle the two aforementioned problems of traditional integration systems, i.e., integrating complex data into a unified repository for supporting complex queries and analysis, and performing data integration tasks autonomously in reactive and dynamic way. Therefore, we propose an innovative framework for this purpose. To the best of our knowledge, there exists no such integration framework in the literature that combines reactive, real-time features and complex data context together into a warehousing environment. Complementing these features is necessary to meet a wide variety of today's operational business intelligence applications, which are typically based on active and real-time data warehousing. To integrate complex data, the framework exploits XML to cope with heterogeneity of data, and employs Web services (WSs) to tackle the distribution and interoperability problems of data sources. Unlike in traditional data integration systems, specifically mediation-based approaches, the integrated data are warehoused into so-called Active XML (or AXML) repository. AXML documents are XML documents where some of the data are given explicitly, while other parts are given implicitly by embedding calls to WSs [2]. In AXML repository, data are not only warehoused but also functions (or services) for integrating up-to-date information. Embedded AXML services are invoked implicitly or explicitly realizing *on-the-fly* data integration, for warehousing (not mediation).

The rest of this paper is organized as follows. Data integration approaches and positioning our approach are discussed in section 2. Section 3 introduces our framework for integrating complex data actively. The framework implementation is demonstrated in section 4. Section 5 discusses a case study to validate our framework concepts. Finally, we conclude and discuss future research trends in section 6.

SAC'12 March 26-30, 2012, Riva del Garda, Italy.

2 Data Integration Approaches

There are two main families of approaches to integrate data: virtual views and materialized views (warehousing).

Virtual Data Integration. In this approach, data are accessed from the sources on demand when a user submits a query to the information system (e.g., mediators). Mediated systems integrate data from heterogeneous data sources by providing a virtual view of all data sources (e.g., the TSIMMIS [4] and Ariadne [8] projects). Mediated systems provide the single mediated schema to the user, and users pose their queries w.r.t. this schema. In this context, XML and WSs are used efficiently for integrating web data in federated systems [13], and in peer-to-peer (P2P) and mediated systems (e.g., the AXML project [1, 2]). The major advantage of this approach is its flexibility, since mediators are able to reformulate and/or approximate queries to better satisfy the user needs. However, when the data sources are updated, modified data are lost, which is not pertinent in a decision-support context where historicity of data is important.

Data Warehousing. In this approach, relevant data are filtered from data sources and pre-stored into a repository (namely a warehouse) that can be later queried by users [6, 7]. This approach is mainly designed for decision-making purposes and supports complex query operations [5]. Compared to virtual data integration approaches, classical warehousing approaches lack of data freshness when dealing with data sources that may update their contents very frequently. They also cannot handle a large number of heterogeneous and distributed data sources that need to store their data in a central repository and keep them always up-to-date.

To improve data freshness and realize real-time decision support systems, a framework for building Zero-Latency Data Warehouse (ZLDW) is proposed [12]. It captures and loads data from heterogeneous data sources using a continuous data integration technique. Moreover, it combines the integration technique with an Active Data Warehousing (ADW) approach [11]. ADW exploits active rules to achieve auto-decision-making by minimizing user intervention in processing complex analysis tasks, so-called *analysis rules*.

Discussion and Positioning. Although our approach conforms to warehousing systems, it inspires some virtual approach features. For instance, it integrates data from a large number of heterogeneous and distributed data sources that are likely to be updated frequently, due to using AXML. Different than virtual approach whereas AXML can be used for mediation-based integration, our approach stores relevant data using AXML. The target repository contains not only the integrated data but also calls to integration services. Hence, both historical data and real-time data are integrated together in the AXML repository. Accordingly, a better performance can be achieved from saving response time when querying data, especially if data sources are physically located far from the integration system.

Comparing our approach to related works, i.e., ZLDW and ADW, ZLDW must update data more frequently to improve data freshness using push/pull technology via a message queuing system. The ZLDW and ADW approaches use traditional batch data integration to integrate data that do not need to be continuously updated and integrated. There is also no mention on how to handle heterogeneity and distributed issues of data sources. In a different way, while ZLDW and ADW approaches use active rules for automating routine analysis tasks, our approach employs active rules along with event mining to automate and reactivate data integration, the early phase of data warehousing process.

3 Complex Data Integration Framework

Data integration is a crucial phase in data warehousing. The complexity issues of data and the autonomous integration requirements need to be handled. We present in this section our framework to accomplish the integration tasks on complex data in an original way. Metadata-based integration services are employed for performing data integration tasks. The target of these tasks is a set of AXML documents, warehoused in a native XML-based repository. Employing XML and WSs in integrating data ensures tackling data heterogeneity, interoperability, distribution and freshness. In addition to "data sources" as input and the unified "AXML repository" as target, the data integration framework consists of three main modules: integration services, logging events, and management and reactivity, as shown in figure 1. Framework processes are represented as rectangles, metadata as rounded rectangles and flow of data among framework components are shown using arrows. Framework modules are briefly discussed in following sections. A prototype framework including different components and services is implemented and deployed as a Web application.

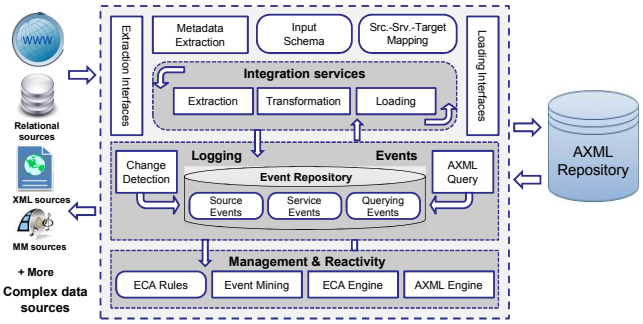

Figure 1: Complex data integration framework

3.1 Integration Services

As in traditional data integration frameworks, ETL tasks are the main processes of our framework, for integrating data from their sources to a target. Nevertheless, such ETL tasks are performed via a set of metadata-based services. Depending on metadata ensures minimal user intervention for maintaining services. Moreover, integration services are invoked autonomously either due to triggering ECA rules (where the action part invokes a specific integration service), or due to querying the embedded calling services in AXML documents. Beside ETL services, some aided interfaces (i.e., extraction and loading interfaces), processes (i.e., metadata extraction), and metadata (i.e., input schema, sources-to-services, and services-to-target mapping schema) are required to accomplish different data integration tasks.

ETL Services. ETL services involve three main tasks. *i*) extracting relevant data from a variety of data sources using *extraction services*. *ii*) transforming extracted data into XML format using *transformation services*. *iii*) loading transformed data into the AXML repository using *loading services*. ETL services are based on a metadata-driven approach for integrating relevant data. The metadata-driven approach allows users to specify "what" data need to be integrated, without regarding "how" to integrate them. Due to the heterogeneity of data sources, extraction services can utilize several types of interfaces. Transformation services involve several functions or services to manipulate extracted data. Loading services help load transformed data into the

AXML repository. The type of data to be loaded is defined either as "explicit data" or "implicit data" in the services-to-target mapping schema. An explicit data type is mapped to a static element in the corresponding target of AXML documents, while an implicit data type is mapped to a dynamic element that represented as call to service.

Extraction Interfaces. Extraction interfaces are sets of specifications that extraction services follow in order to access complex data sources. They facilitate interaction between data sources and integration services.

Metadata Extraction. Metadata extraction is the process of extracting metadata from several complex data sources via extraction interfaces. Beside extracting metadata of non-hierarchical data sources, such processes navigate metadata of hierarchical structured data sources from high to low level. Users then select a relevant metadata schema via a GUI.

Input Schema. The input schema specifies the structure of relevant complex data sources. The more metadata we have, the better and easier the administration and automation of the data integration services. Extracted relevant metadata are stored into the input schema that is considered as catalog of input data sources' structure. The input schema could be queried continuously to maintain data integration tasks.

Sources-Services-Target Mapping. Sources-to-services and services-to-target mapping schemas bind data sources and their associated services, and services and their associated targets, respectively. Mapping schemas have a significant role in automating and managing data integration services.

Loading Interfaces. Loading interfaces are sets of specifications that loading services follow in order to access the target repository. Hence, they communicate between integration services and the AXML repository.

3.2 Logging Events

Our framework logs events about activities of different modules. Then, data mining techniques are applied on logged events to discover some interesting rules. Discovered rules can be utilized to maintain, automate, and reactivate the working of data integration services.

Change Detection. Change detection processes monitor changes that may occur in data sources. We distinguish between two types of event changes, namely "structure changes" and "content changes". Due to the data sources heterogeneity, there are several alternatives for detecting changes (e.g., relational and XML databases triggers, metadata-snapshots comparison, timestamps of last modification, version numbers, and third-party change detection applications).

AXML Querying. AXML documents can be queried either by the user or analysis modules that built upon the AXML repository. When querying AXML documents, embedded services could be invoked implicitly. Service results replace the calling service node, or append to it. Appending results after the calling node permits the service to be reused later.

Event Repository. The event repository involves a variety of events that are logged from different framework modules. Events can arise from changing structures or contents of data sources. Examples of structure changes events include events of adding, altering, and/or dropping data sources. Content changes events include data manipulation events such as inserting, updating, and/or deleting values at specific data source. Integration services log events about data sources that they extract, transform, and load. Event information includes date processed, number of records read, written, input, output, updated, errors, and services carried out. Another category of events is logged when querying AXML documents, including information about AXML documents, XPaths and XQuery expressions.

3.3 Management and Reactivity

One of the most important framework features is carrying out data integration tasks autonomously and reactively. To achieve reactivity, our framework is supplied with a set of active rules that follow the ECA paradigm. Events of active rules are defined by users or discovered from the event repository by the event mining module. Moreover, invoking the embedded services of AXML documents refreshes the repository with active and up-to-date information. Such embedded services are managed via the AXML engine.

ECA Rules. ECA or active rules are associated to framework objects such as data sources, integrations services, and the AXML repository. These rules make objects responsive to a variety of events, where the actions are taken when encountering an event and satisfying the condition. A generic example of ECA rule follows.

ON *data-source-update* **WHEN** *condition* **DO** *invoke-integration-service*

Events are either simple or composite. Simple events (also called, primitive or atomic events) correspond to a data modification operation, execution of integration service, or querying an AXML document. A composite event is a logical combination of simple events or other composite events, defined by logical operators such as disjunction (*or*), conjunction (*and*), sequence (*and then*), and negation (*not*). Such composite events are semantically and formally defined in [3], where an event E is a function on the time domain onto the Boolean values: $E(T) \rightarrow True$; if an event E occurs at time point t, and $E(T) \rightarrow False$; otherwise.

Disjunction of two events $E1$ and $E2$, denoted $(E1 \bigtriangledown E2)$, occurs when either $E1$ or $E2$ occurs.

Example 1 *(Disjunctive Event)*: If inserting or updating contents of a specific object (e.g., table "orders" of database "sales"), the framework invokes the corresponding integration service to integrate new data. The semantic structure being detected and manipulated by the ECA engine.

```
<eca:event xmlns:evtCnt="http://www.eca-rules.com/events/content">
 <evt:disconjunctive>
  <evt:simple><evtCnt:insert-tuples obj-type="table"
        obj-name="sales.orders"/></evt:simple>
  <evt:simple><evtCnt:update-tuples obj-type="table"
        obj-name="sales.orders"/></evt:simple>
 </evt:disconjunctive>
</eca:event>
```

Conjunction of two events $E1$ and $E2$, denoted $(E1 \bigtriangleup E2)$, occurs when both $E1$ and $E2$ occurs in any arbitrary order.

Example 2 *(Conjunctive Event)*: If content changes are detected in multiple data sources, so that a specific item is deleted, action to send a message indicating that the item is no longer available can be taken.

```
<eca:event xmlns:evtCnt="http://www.eca-rules.com/events/content">
 <evt:conjunctive>
  <eca:variable name="itemID" lang="xpath" expr="./@ID"/>
  <evt:simple>
   <evtCnt:delete-tuples objSrc="src1.items"
         attribute="item_key" value="$itemID"/>
  </evt:simple>
  <evt:simple>
   <evtCnt:delete-tuples objSrc="src2.products"
         attribute="product_key" value="$itemID"/>
  </evt:simple>
 </evt:conjunctive>
</eca:event>
```

The sequence of two events $E1$ and $E2$, denoted $(E1; E2)$, occurs when $E1$ occurs before $E2$.

Example 3 *(Sequence Event)*: When a data steward adds a specific database as data source, selects one object of this database, and then selects the relevant fields from this object, action can be taken to invoke integration services to integrate data considering all parameters selected by the user.

```
<eca:event xmlns:evtStr="http://www.eca-rules.com/events/structure">
 <evt:sequence>
  <evt:simple><evtStr:add-data-source type="database"
        name="sales"/></evt:simple>
  <evt:simple><evtStr:add-object type="table"
        name="orders"/></evt:simple>
  <evt:simple><evtStr:add-field><eca:variable name="newField"
  lang="xpath" expr="./@Field"/></evtStr:add-field></evt:simple>
 </evt:sequence>
</eca:event>
```

The negation of event E, denoted $(\neg E)$, occurs when there is no occurrence of E.

Example 4 *(Negation Event)*: When adding a specific object of data source to integration application is important, the data steward can be notified if this object is not added.

```
<eca:event xmlns:evtStr="http://www.eca-rules.com/events/structure">
 <evt:not>
  <evt:simple><evtStr:add-object type="table"
   name="promotions"/></evt:simple>
 </evt:not>
</eca:event>
```

Complex events can be noted as combination of composite events (e.g., when either $E1$ or $E2$ occurred, and then $E3$, but not $E4$). Conditions denote queries to one or several events and are expressed in XPath or XQuery.

Example 5 *(Condition)*: If the framework detects content changes in an object, it further requires querying the input schema to test whether this object is relevant or not.

```
<eca:condition lang="xpath"
 check="./*/object-name/text()=$updatedObject"/>
```

Actions can be notifying the integration server when a data source changes, sending message, executing an integration service, or invoking embedded AXML services.

Example 6 *(Action)*: If condition in Example 5 evaluates to true, then an action can be taken to update the corresponding target object.

```
<eca:action>
 <act:invoke-service name="updateTarget" opertion="updateObject"
  input="$updatedObject"/>
</eca:action>
```

Event Mining. Event mining processes apply data mining techniques on the framework's XML logged events. Each event type is mined for a specific purpose. For instance, event mining can be applied to judge classification and determine whether the rule should be fired and/or activated. In addition, event mining can help to discover association rules and frequent logged events that may change the policy of AXML activation timing [10].

ECA Engine. The ECA engine plays an important role to control the evaluation of the ECA rules, which parsed as XQuery triggers. It manages event triggering, evaluates the conditions, and performs the associated rule actions.

AXML Engine. The AXML engine manages the evaluation of embedded AXML services. It monitors querying the AXML documents, and then invokes the embedded services to refresh them with up-to-date information. Beside evaluating services implicitly when data are requested, services can also be evaluated explicitly (e.g., hourly, daily or after occurring event). The engine manages where services' results are written, either replacing the calling service or appending to it. Moreover, AXML engine manages different versions of AXML documents in AXML repository.

4 Implementation

We implement our framework prototype mostly using standard open-source software. It is implemented as a Web-based application using Oracle, XML and Java technologies. Figure 2 shows the deployment diagram, which visualizes the hardware, middleware and software. The diagram is composed of multiple tiers: data source, integration application, web, client and target tiers. Different tiers communicate via several protocols and interfaces.

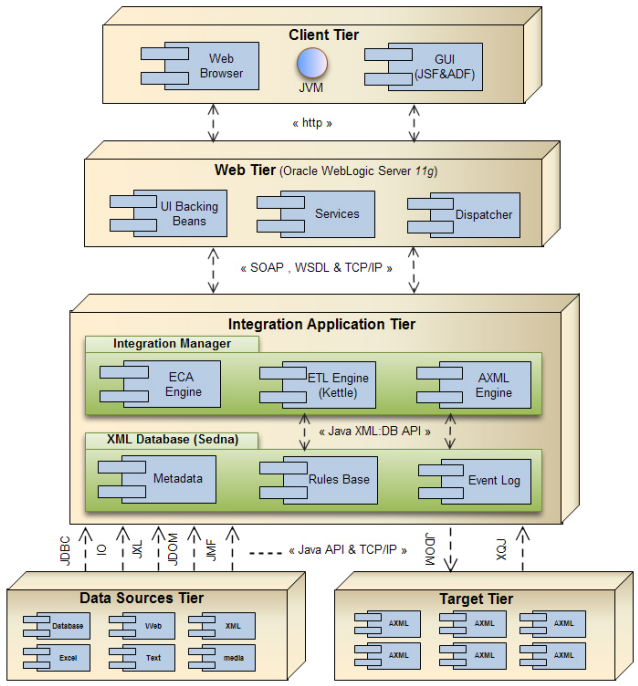

Figure 2: Deployment diagram of data integration web application

The integration application tier is composed of two main sets of cooperative components: application manager and native XML database. Our framework utilizes the Java library of *Pentaho Data Integration* (PDI)[1] for performing ETL tasks. The XML database is implemented using *Sedna*[2]. It involves several collections of databases, such as metadata, defined and mined active rules, and logged events. Active rules are implemented, triggered and fired using the XQuery trigger facility supported by Sedna.

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 ADF and Sun JSF. The user specifies the interesting data source, and then extraction services bring out the metadata from which to select a relevant schema. Users have access to several functionalities such as mapping data sources with integration services and targets, scheduling the execution of integration services, browsing and querying AXML repository, browsing changes of data sources, and defining ECA rules. 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.

[1]http://kettle.pentaho.com/
[2]http://www.modis.ispras.ru/sedna/

5 Case Study

The main power of our framework is to integrate complex data into an unified repository autonomously and reactively. This can be accomplished by capturing the metadata schema of data sources, integrating data using metadata-based integration services, monitoring sources to detect any occurred changes, and apply event-driven rules to reactivate integration services. There are several case studies that have heterogeneous and distributed data with frequently changing nature such as: health-care, traveling, retails, comparing prices, and e-commerce.

Let us address the *"compare prices"* case study where a data integration framework is needed to integrate product (e.g., digital cameras and camcorders) data from several vendors. The framework helps define data sources, one by one, via a GUI. Metadata extraction services are invoked to extract data sources' structure, and then the user selects relevant attributes from the extracted structure. Defining data sources events are logged into the event repository. Data sources may include data about vendors, products, pricing, seasons, promotions, date, ads, product images, product videos, as well as images and videos captured by product for reviewing. Such heterogeneous types of data are unified into XML. Integration services communicate with suppliers by accessing their data sources (e.g., Web servers, relational databases, spreadsheets, multimedia, delimited text, and XML), which are distributed over networks. Relevant metadata are written into the input schema. An XML file is generated from the input schema, and is adapted into a format that can be manipulated by the PDI engine, for performing ETL. Beside integration services, external WSs [3] can be imported, such as WSs to return latest currency exchange rates for currency conversion into a unified currency format. Moreover, other services can be developed, such as comparing prices of the same camera from different vendors returning the cheapest one that is written as call to service.
Example 7: This example shows a fragment of AXML document before and after invoking the embedded services.

AXML fragment before invoking embedded services:

```
<lowestPrice>
 <axml:cs mode="replace" serviceName="getPricesService"
  serviceURL="http://localhost:7101/services/getPricesPort?WSDL"
  methodName="getLowestPrice">
  <axml:parameters>
   <axml:param name="product_key" value="5000001"/>
  </axml:parameters>
 </axml:cs>
</lowestPrice>
```

AXML after invoking embedded services:

```
<lowestPrice>
 <axml:result>$315</axml:result>
</lowestPrice>
```

To get rid of slow-selling and high in-stock items, most vendors offer promotions to their stock throughout the year. An event-driven mechanism can be applied effectively to monitor such discount amounts, and to feed the discounted prices into the target repository by invoking the appropriate integration services.Content change events are logged when adding a new camera item, updating or deleting one. But structure change events are when adding a new vendor with its family of products or dropping existing vendor with its family of products. Purchasing items, their timing, promotions, products, and their vendors are queried. Querying events are also logged into the event repository. All event types need to be mined in order to discover interesting knowledge, for reactivating the integration services via a set of defined rules.

[3]http://www.xmethods.net, http://webservices.lb.lt, etc.

6 Conclusions and Perspectives

In this paper, we propose a framework for integrating complex data into an AXML repository autonomously. Our framework tackles limitations of traditional data integration systems. It combines both advantages of virtual integration and warehousing approaches, either by handling a large number of heterogeneous data sources using XML and WSs, or storing integrated data into a unified AXML repository to support complex analysis queries later. The target repository stores not only data, but also service calls. Thus, such services can be invoked implicitly or explicitly to refresh repository contents. Moreover, our framework has reactive and autonomic capabilities by warehousing events about framework activities, mining such events, and then following ECA rules mechanism to reactivate integration tasks. Discovering interesting knowledge from logged events and triggering active rules help minimize user intervention to execute frequent integration tasks. Our software prototype and detailed technical report are available on-line [4].

Recall that different framework activities are logged in a specific repository for events. Since the event repository can be considered as miniature copy of a data warehouse, OLAP tools can thus be coupled with data mining to explore and analyze logged events information. We also aim to carry out machine learning techniques to automate schema matching and mapping between data sources and integration services, using descriptions of sources and services.

[4]http://eric.univ-lyon2.fr/∼rsalem/axdi/

7 References

[1] S. Abiteboul, O. Benjelloun, and T. Milo. Web services and data integration. In *WISE '02*, pages 3–6, Washington, DC, USA, 2002.

[2] S. Abiteboul, O. Benjelloun, and T. Milo. The active XML: an overview. In *VLDB Journal*, pages 1019–1040, 2008.

[3] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *Proceedings of VLDB*, pages 606–617, Santiago, Chile, 1994.

[4] S. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *IPSJ*, pages 7–18, 1994.

[5] J. Darmont, O. Boussaid, J. Ralaivao, and K. Aouiche. An architecture framework for complex data warehouses. *ICEIS'05, Miami, USA*, pages 370–373, 2005.

[6] W. Inmon. *Building the Data Warehouse.* John Wiley & Sons, 1996.

[7] R. Kimball. *The data warehouse toolkit.* John Wiley & Sons, 1996.

[8] C. A. Knoblock, S. Minton, J. L. Ambite, N. Ashish, I. Muslea, A. G. Philpot, and S. Tejada. The ariadne approach to web-based information integration. *IJCIS*, 10(1 & 2):145–169, 2001.

[9] T. B. Pedersen. Warehousing the world: a few remaining challenges. *Proceedings 10th DOLAP*, pages 101–102, 2007.

[10] R. Salem, J. Darmont, and O. Boussaid. Efficient incremental breadth-depth xml event mining. In *IDEAS'11, Lisbon, Portugal*, 2011.

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

[12] M. N. Tho and A. Tjoa. Zero-latency data warehousing for heterogeneous data sources and continues data streams. In *iiWAS'03, Jakarta, Indonesia*, pages 55–64, 2003.

[13] F. Zhu, M. Turner, I. A. Kotsiopoulos, K. H. Bennett, M. Russell, D. Budgen, P. Brereton, J. A. Keane, P. J. Layzell, M. Rigby, and J. Xu. Dynamic data integration using web services. In *ICWS*, pages 262–269, 2004.