Conceptual Workflow for Complex Data Integration using AXML

Rashed Salem, Omar Boussaïd and Jérôme Darmont
Université de Lyon (ERIC Lyon 2)
5 av. P. Mendès-France, 69676 Bron Cedex, France
Email: firstname.lastname@univ-lyon2.fr

***Abstract*—Relevant data for decision support systems are available everywhere and in various formats. Such data must be integrated into a unified format. Traditional data integration approaches are not adapted to handle complex data. Thus, we exploit the Active XML language for integrating complex data. Its XML part allows to unify, model and store complex data. Moreover, its services part tackles the distributed issue of data sources. Accordingly, different integration tasks are proposed as services. These services are managed via a set of active rules that are built upon metadata and events of the integration system. In this paper, we design an architecture for integrating complex data autonomously. We have also designed the workflow for data integration tasks.**

I. Introduction

Data warehousing is a very successful approach for decision-support. Data warehousing processes involve integrating, storing and analyzing large amounts of business data. Classical data warehousing is very well adapted to structured data, but structured data only represent a small part of relevant data that need to be warehoused for several enterprises. Indeed, there are huge volumes of heterogeneous data (e.g., structured, relational, multidimensional, semi-structured/XML, emails, Web, text, and multimedia data) that are available over networks. We term these data *complex data* [1]. 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 [2] is not very adequate when dealing with complex data, particularly in the data integration phase.

Moreover, we are motivated by handling complex data and performing integration tasks ”intelligently”, i.e., in an autonomous and active way. Thus, we propose a system that integrates complex data from heterogeneous and distributed sources into a unified repository autonomously. Our integration system is mainly based on the concept of *data integration services*. Indeed, a service provides transparent access to a variety of relevant data sources as if such sources represented as a single source. Data integration services can be effectively applied utilizing the Active XML (AXML) language [3]. An AXML document is an XML document with embedded calls to Web services. We use XML to unify complex data into a unified format, while Web services provide a uniform access to information, independent from platform, system, language, communication protocol and data format. In order to automate, reactivate data integration tasks and enrich the system with intelligent features, our architecture is managed and controlled using active rules (called ECA, Event-Condition-Action).

Building an integration system involves designing, modeling and using that system. In this paper, we focus only on designing the system. Thus, the main contributions of this paper are:

- propose and design a general AXML-based architecture enriched with active and intelligent features (active rules and services), in order to integrate complex data from heterogeneous and distributed data sources;
- design the workflow that depicts the sequence of data integration tasks and steps.

The rest of this paper is organized as follows. Section II presents our architecture for integrating complex data. The detailed data integration workflows are demonstrated in section III. Related works are discussed in section IV. Finally, we conclude and highlight future trends in section V.

II. AXML-based Architecture for Integrating Complex Data

Our architecture handles complex data integration, which integrates data residing at distributed and heterogeneous data sources into a unified repository (Fig. 1). Data sources are integrated without making any changes on their design or structure. Data integration services are utilized to perform Extraction-Transformation-Loading (ETL) tasks. Integration tasks extract data from a variety of data sources, transform them into XML format, and load them into a repository of AXML documents. The workflow of integration tasks is organized via a set of rules, managed and controlled by ECA and AXML engines.

Indeed, metadata of architecture components can play an important role with events logged from these components, in order to automate data integration tasks. Metadata include data not only about different data sources (e.g., type, location, access information, format, structure, etc.), but also about several integration tasks (e.g., service name, methods, parameters, sources-to-services mapping, services-to-target mapping, etc.), AXML repository (e.g., URL, port, user identification, destination documents, etc.), events (e.g., event type, target log, event origin, event type, etc.), and rules (e.g., events-to-actions mapping, conditions, trigger timing, etc.). Active rules are built upon logged events (section III-D1) and follow the ECA form.

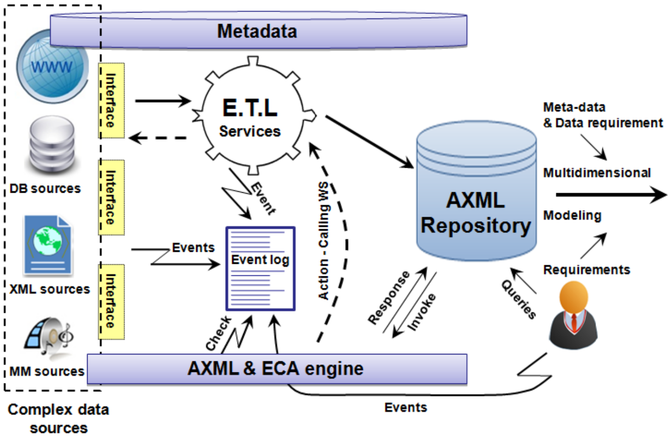

Fig. 1. Complex data integration architecture

III. Conceptual Workflow for Integrating Complex Data

Integrating complex data into a unified repository in an autonomous way consists of several steps and tasks, which requires a complex integration system. However, a workflow can describe the sequence of these steps and tasks simply. We design the workflow that describes the sequence of integration steps (Fig. 2). To simplify the system workflow, we subdivide it into four sub-workflows. The first workflow handles preparation tasks before integrating relevant data; it depicts how to construct the input schema based on metadata of data sources. The second workflow deals with monitoring changes of data sources and filtering relevant changes, then applying these changes into the input schema. The third workflow describes how to integrate relevant data from the input schema into the AXML repository, using open-source tools for carrying out the ETL tasks. The fourth workflow addresses the mining and analyzing tasks that can be performed upon event logs.

A. Constructing the Input Schema

Data sources have enough metadata that can help automate the data integration process. However, all metadata of a specific data source may not be relevant to a particular application. Relevant metadata are extracted and stored into a so-called *input schema*. An input schema identifies all relevant data sources, their characteristics and structures.

Let us address the workflow scenario of constructing the input schema (Fig. 3). A particular user can login to the integration framework identifying his role. The user selects data sources identifying their location path or URL and their access information. For each selected data source, a dispatcher service is directed to the appropriate service for retrieving metadata and schema. Then, the user selects only relevant metadata and schema from the service's results via a graphical interface. The selected metadata are written into the input schema. It is the first step in building this important document.

1) Types of users: There are three types of user roles: administrator, steward and normal user. In general, an administrator is someone who can manage and control different modules of the framework. A steward is someone who has enough expertise of data integration and its processes. A normal user

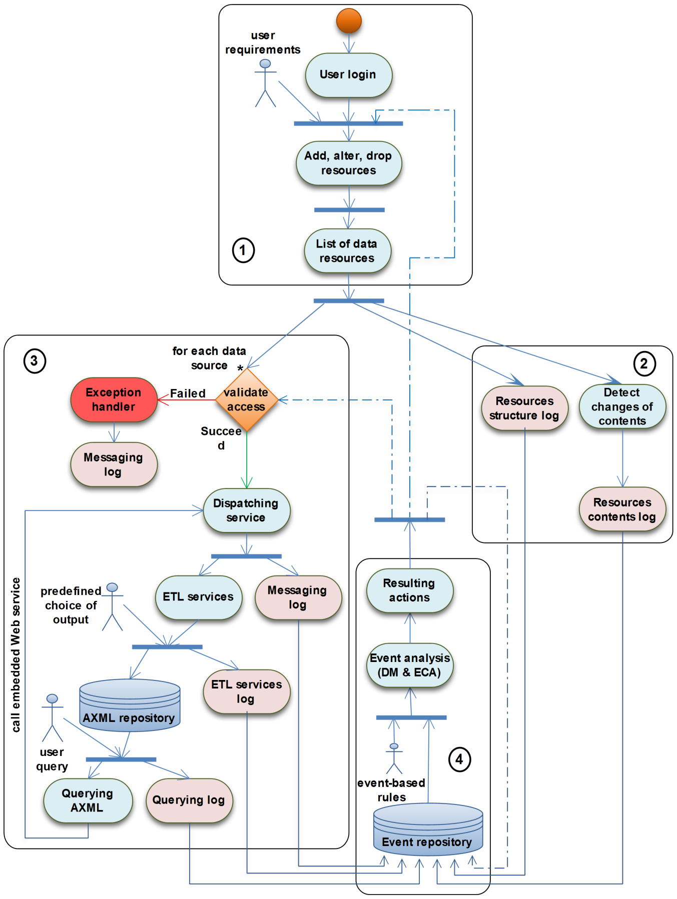

Fig. 2. Complex data integration workflow

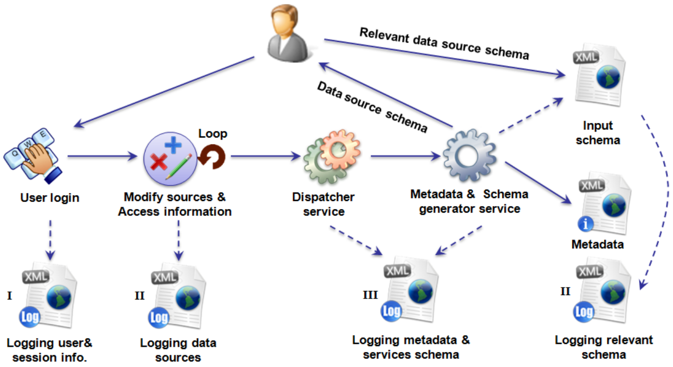

Fig. 3. Constructing input schema based on metadata workflow

is an end-user who can query the AXML repository. It is not necessary for normal users to have any knowledge about data integration or its processes. From here onwards in this paper, user refers to steward.

2) Explicit input schema evolution: Different data sources are subject to continuous changes of their structures and/or their contents. User requirements are also primordial for successful data integration. These requirements might change over time and should be taken into consideration. For this reason, the framework allows users to add new data sources, alter and/or drop existing data sources. Altering data sources involve adding new relevant objects, altering and/or dropping existing objects. Such structure changes in data sources are written into the input schema. As a result, the input schema is always up-to-date. If such structure changes are carried out by the user, it is called explicit input schema evolution.

3) Resource metadata versus input schema: Resource metadata are data about data sources that describe, locate, ease to retrieve, utilize, or manage data sources (Fig. 4). In our framework, resource metadata can be obtained automatically using metadata processing or manually using a graphical interface. Resource metadata are a superset of (or include) the input schema. The input schema represents relevant metadata to a particular application.

B. Monitoring Change Detection

Data sources should always be monitored for detecting changes. We distinguish two types of event changes, namely structure changes and content changes. Structure changes of data sources are changes related to the structure of data sources, such as adding new object (e.g., table, view, etc.) to a specified database or deleting a field from a specified spreadsheet. Content changes are changes related to the contents of data sources, such as adding record(s) to a specified table or changing the value of XML document element(s). As depicted in Fig. 5, Several alternatives are proposed for detecting changes from heterogeneous data sources. New detected changes are compared against previous versions of input schema or metadata in order to determine if these changes are relevant or not. Only relevant changes are taken into consideration in order to update the input schema.

1) Data source change detection alternatives : Due to the heterogeneity of data sources, there are several alternatives for detecting changes.

- ***Triggers***. The first alternative is to apply triggers at data sources that support them (i.e., databases, XML, etc.). Triggers are responsible for monitoring content changes, rather than structure changes. However, applying triggers at data sources is not always allowable due to access limitations. Indeed, most data sources do not ”know” that they are part of an integration system.
- ***Metadata comparison***. The second alternative is to generate metadata for some types of data sources periodically and compare them against previous extracted versions. Metadata comparison can be effectively utilized for monitoring structure changes.
- ***Third-party applications***. The third alternative is to employ third-party applications designed for change detection purposes in order to detect content changes for some types of data sources. For instance, there are several third-party applications to monitor web pages for changes (e.g., changedetection.com, WebSite-watcher, etc.).

2) Change detection module: The change detection module monitors changes that may occurr in different data sources. Notice that not all encountered changes at data sources are relevant. Such changes are called irrelevant changes and should not affect the input schema. The change detection module (Fig. 5) deals with different types of changes monitor alternatives (section III-B1) with some minor differences. Firstly, changes coming from triggers (either structure changes or content changes) are compared and filtered in order to get only the delta of relevant changes. The corresponding module affects the input schema and metadata with these relevant structure changes. However, it notifies the integration server with content changes in order to invoke the appropriate integration service to engage these changes. Secondly, the corresponding module gets a new version of metadata and compares it against the previous extracted version. If it finds encountered changes, it filters them against the input schema in order to get only a delta of relevant changes. Such relevant changes affect the input schema and metadata with structure changes. Thirdly, there are rich third-party applications that can notify a specific server with encountered changes. Such changes need to be filtered by the change detection module for getting only relevant data. However, the user checks relevant changes manually when using limited third-party applications,

3) Automatic filtering of irrelevant changes: Irrelevant changes can be filtered automatically by comparing them to the input schema. If any change entry does not match with input schema entries, the change detection module marks this entry as irrelevant. On the other hand, not all ”irrelevant” changes should be neglected. Some irrelevant changes may be important. For example, if the change detection module detects new object ”X”, thant not found extracted versions of input schemas and metadata, the module considers this object as irrelevant. But in fact, this object is important for the user and was not available when constructing the input schema. So, the change detection module should notify the user with any new irrelevant changes. Then, the user decides if a new object is actually relevant or not.

4) Implicit input schema evolution & change detection: The input schema needs to be frequently updated and refreshed in order to meet new requirements of the application. Updating the input schema is done either explicitly (by user), or implicitly (by detecting changes automatically using the change detection module). The change detection module can be configured to detect changes periodically.

Moreover, the input schema needs to be monitored for changes. Because the input schema is represented in XML, active XQuery triggers can be applied to the input schema for detecting changes. When any updates detected in the input schema, some actions may be taken such as invoking the appropriate integration services in order to integrate the new relevant data.

C. Metadata & AXML-based Integration of Complex Data

Constructing the input schema rather than defining Sources-to-Services mappings are important components before executing the integration tasks. Each relevant data source is dispatched to the appropriate integration service. The integration service is managed using Pentaho Data Integration (Kettle)[1]. The output of integration services is a set of AXML documents that can be later queried by calling embedded Web services (Fig. 6).

[1] http://kettle.pentaho.org/

Fig. 4. Resource metadata

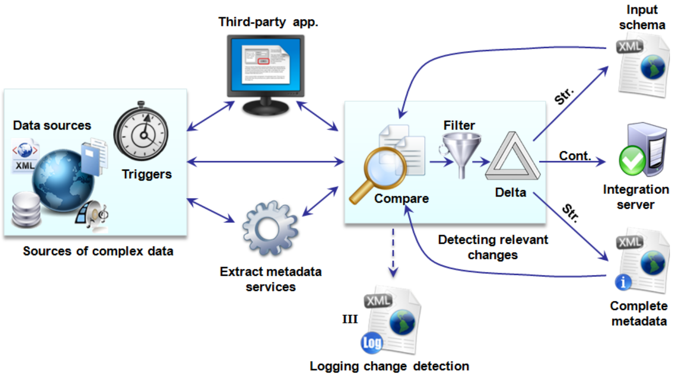

Fig. 5. Monitoring change detection workflow

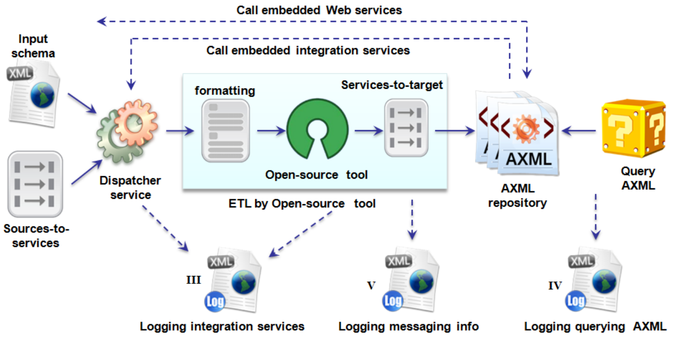

Fig. 6. Integrating complex data using open-source tool workflow

1) Integrating data: From an implementation viewpoint, using an open source tool for performing ETL tasks saves implementation time. Our framework utilizes Pentaho Data Integration (PDI). PDI is a powerful, metadata-driven ETL tool. It has a graphical user interface (called Spoon) that allows users to design and execute transformations and jobs. Transformations and jobs can be saved in binary format or can be described in XML format. Our framework utilizes PDI's Java library to manage the integration tasks. It adapts the structure of relevant data sources from the input schema into XML files handled by PDI's tools. Then, PDI's tools (e.g., Pan or Kitchen) can be invoked to execute transformations or jobs, which are described also in XML format. The XML file of a specific transformation involves metadata about the input data source and its detailed characteristics, metadata about transformation rules, and metadata about output targets. Such metadata describe the ETL operation. The target of the integration tasks is always a set of AXML documents that are stored in a native XML database, namely eXist [2].

2) Explicit versus implicit elements of AXML documents: To specify automatically explicit AXML elements versus implicit AXML elements, our framework should add a flag attribute for each target element as static or dynamic (explicit or implicit) in the mapping step. A static element (default) is written as a traditional XML element and a dynamic element is written as a call to a Web service. We assume that we can apply some statistical operation on event changes to know which object or sub-objects of data sources are frequently changed. If an object/sub-object exceeds a specific threshold of number of changes in a particular period, it is expressed automatically as invocation of Web services and its type flag should be changed into dynamic.

3) Querying AXML documents: AXML documents can be queried by the user or by analysis modules built upon the AXML repository. When querying AXML document, embedded Web services should be invoked. The services' results replace the node calling the service, or append to it. Appending results after call's node permits the service to be reused later.

D. Mining and analyzing events

Different events are logged when constructing the input schema, detecting changes in data sources, and integrating data using an open-source tool. Such events need to be mined and analyzed in order to maintain, and automate different integration tasks (Fig. 7) . The user defines several events-to-action rules using a graphical interface. When mining and analyzing events, if a specific event is encountered and its associated condition is met, then the associated action is taken.

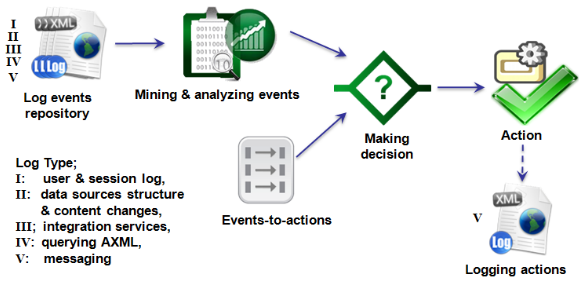

Fig. 7. Mining and analyzing different events log workflow

1) Event log types: Event logs are categorized according to their origin modules.

- ***User & session log***. This log contains events logged by accessing users. It includes their identifications, their roles, and starting and ending times of sessions.
- ***Data source structure & contents change log***. This log contains events logged when changing the structure and contents of data sources.
- ***Integration services log***. This log contains events logged by executing extraction, transformation, and loading services.
- ***AXML query log***. This log contains events logged when querying AXML documents, and invoking embedded Web services.
- ***Messaging log***. This log contains information, warning, and error messages.

2) Event repository: Different events are modeled and stored into a specific repository, in order to be easily mined and analyzed. The event repository can be modeled by applying a *star schema* model [2], where a single fact document represents all events. However, there are multiple dimension

[2] http://exist-db.org/

documents (e.g., user dimension, machine dimension, session dimension, data source dimension, AXML document dimension, date and time dimension). Another solution is to apply a *constellation schema* model [2], where there are several fact documents; one fact document for each type of the following event types: data source contents and structure changes, ETL services, querying AXML repository, and messaging. However, there are still multiple dimension documents.

3) Event mining and analysis: Each event type is mined and analyzed for a specific purpose. Events are described in XML, which results in mining XML documents. Both structure and contents of XML documents are interesting to be mined and analyzed. Examples of mining and analyzing tasks are given for event types.

a) Data source structure events mining and analysis.: Finding similarity of events (e.g., adding new data sources) can result in applying the same integration services automatically to integrate relevant data, taking the new parameters into account. Moreover, association rules can be mined in order to find the relationship between the most frequently accessed data sources and the nature of the application.

b) Data source contents event mining and analysis.: Content change events affect AXML explicit parts by replacing old elements with newest ones. They also affect the AXML implicit part by notifying the integration server to execute the appropriate services for getting changes. For this reason, content change events should be mined and analyzed in order to execute appropriate actions. Finding the most frequently updatable pattern of a specific data source can also lead to change this pattern to be expressed as calls to services instead of defining it explicitly.

c) Integration (ETL) service event mining and analysis.: PDI already generates logs containing the status of integration tasks such as the total number of lines read, written, input, output, updated, and rejected. Such logs are mined in order to maintain the integration services themselves. For instance, let us assume that the total number of lines read, written, input, or output are equal to zero. In this case, the framework indicates that run-time errors or logic errors have happened.

d) AXML repository query event mining and analysis.: Finding the most frequently queried AXML documents and their embedded services may be interesting. Finding the most frequently queried explicit pattern and calculating its changing rate is also important. If changing rate exceeds a specific threshold, then we can recommend the user to express this pattern as service calls.

e) Messaging event analysis and mining.: Event logs are an excellent source of information for monitoring the system. Information that is stored in event logs can be useful for analysis at a later time (e.g., for audit and maintain procedures, retrospective incident analysis, etc.).

IV. Discussion and Related Work

There are several approaches to integrate data: virtual views, materialized views (warehousing), and hybrid approaches. In a virtual view (or *lazy*) approach, data are accessed from sources on demand when a user submits a query. In a materialized view (or *eager*) 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. A hybrid approach is applied when integrated data are selectively materialized. Data are extracted from data sources on-demand, but some query results are pre-computed and stored.

Some of the integration architectures 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 in order to share data [4]. Mediated systems integrate data from heterogeneous and autonomous data sources by providing a virtual view of all data sources [5]. They handle not only operational databases, but also legacy systems, Web sources, etc. Mediated systems provide one schema to the user (called the Mediated Schema) and users pose their queries w.r.t. this schema. Although the materialized view approach provides a unified view of relevant data similar to a virtual view, it stores these data in a data warehouse. This approach is mainly designed for decision-making purposes and supports complex query operations.

Our architecture follows the hybrid approach, taking the advantages of both materialized and virtual view approaches. It gains advantage from the virtual view approach by integrating data from a large number of data sources that are likely to be updated frequently, and there are no predefined user queries. It also gains advantage from the data warehousing approach by storing relevant data into a unified repository. As a consequence, better performance can be achieved from saving response time when querying data, especially if their data sources are physically located far away from the integration system.

Furthermore, Our architecture is designed around XML data warehousing approaches. 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, focusing either on Web data warehousing [6], [7], [8], XML data warehousing [9], [10], [11], [12], or XML document warehousing [13], [14], [15].

Technically, our architecture is based on the AXML language [3]. AXML is a useful paradigm for distributed data management on the Web. Several AXML issues have been studied in Peer-to-Peer (P2P) architectures, such as distribution, replication of dynamic XML data, semantics of documents and queries, terminations and lazy query evaluation. Moreover, our architecture is managed and controlled using different active rules (or ECA) [16], [17]. Active rules are also proposed for analysis purposes in data warehousing environments, as analysis rules [18].

V. Conclusion and Future Trends

In this paper, we designed an architecture for integrating complex data into an AXML repository. In this architecture, the integration tasks are expressed as Web services. The work of such services conforms to the work of ECA rules, achieving some form of intelligence when integrating complex data. The choice of AXML results in unifying various data formats, accessing them independently from their platform, system, physical location, and communication protocol. Moreover, AXML documents are always up-to-date due to invoking embedded Web services. The workflow of integration tasks is designed for describing the different steps of the integration architecture. It is subdivided into four smaller ones (section III), each of which describes a specific task in more details.

In the near future, we intend to give more attention to mining and analyzing different events logged from the proposed architecture, in order to achieve intelligent ETL. Moreover, we will implement and validate the data integration system. We aim at developing the complex data integration system as a Web-based application, to help users run and manage it entirely through a lightweight client (i.e., a web browser).

References

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

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

[3] S. Abiteboul, O. Benjelloun, and T. Milo, "The active XML: an overview," in *VLDB Journal*, 2008, pp. 1019–1040.

[4] A. Sheth and J. Larson, "Federated database systems for managing distributed and autonomous databases," *ACM Computing Surveys*, pp. 183–236, 1990.

[5] H. Garcia-Molina, J. Ulman, and J. Widom, *Database System Implementation, chapter 11: Information Integration*. Prentice Hall, 2000.

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

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

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

[9] O. Boussaid, R. Ben Messaoud, R. Choquet, and S. Anthoard, "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, 2006.

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

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

[12] L. Rusu, J. Rahayu, and D. Taniar, "A methodology for building XML data warehouses," *International Journal of Data Warehousing & Mining*, vol. 1, no. 2, pp. 67–92, 2005.

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

[14] V. Nassis, R. Rajugan, T. Dillon, and J. Rahayu, "Conceptual and systematic design approach for XML document warehouses," *International Journal of Data Warehousing & Mining*, vol. 1, no. 3, pp. 63–86, 2005.

[15] R. Rajugan, E. Chang, and T. Dillon, "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, 2005.

[16] J. Bailey, A. Poulovassilis, and P. T. Wood, "Analysis and optimisation of event-condition-action rules on XML," *Computer Networks*, vol. 39, pp. 239–259, 2002.

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

[18] T. Thalhammer, M. Schrefl, and M. Mohania, "Data warehouses: Complementing OLAP with active rules," *Data and Knowledge Engineering*, vol. 39, no. 3, pp. 241–269, 2001.