Architecture for Near Zero Latency in Datawarehouse

Ahmed Kabiri and Dalila Chiadmi
Ecole Mohammadia d'ingénieurs, Université Mohammed V - Agdal, Rabat, Morocco
Email: {akabiri, chiadmi}@emi.ac.ma

***Abstract*—Data warehouse (DW) has been widely recognized as an effective solution for integrating diverse information systems. In addition, DW supplies a platform for business intelligence applications which aims to improve the process of decision making. At execution level, after the first load, DW is regularly populated with fresh data by ETL (Extraction, Transformation and Loading) processes. In real setting, DW refreshment is an event managed by DW administrators. Still, non freshness of data is a drawback of classical DW system. In this paper, we suggest a solution for this issue by presenting architecture of near real time DW. We visit DW system layers and we show how they should cooperate. Finally, we define the scope and the challenge of each layer simultaneously with our suggestion to fill its mission.**

***Index Terms*—KANTARA, Data warehouse, real time data warehouse, ETL, ALWASSIT, MDA.**

I. Introduction

In this section we review data integration solutions which have been investigated by research community. Particularly we focus on data warehouse solutions. In section *B*, we expose the motivation of this work while in section *C* we visit data warehouse architecture. Finally section *D* highlights the outline of this work.

A. Data Integration Background

Data integration is an active area of research. Several solutions have been suggested to same issues. In one hand, there is virtual data integration solutions represented by mediators like WASSIT [3, 4]...etc. A mediation system usually has a global schema, which provides the user with a uniform interface in order to access information stored in the data sources. This category of approaches has the advantage to access to data source on fly. However, rewriting step involved in mediation process is complex and requires non negligible time to dispatch the query before to collect results and combine it's again. Depending on the query complexity and data volume, mediator response will last.

In other hand, data integration with data warehouse approaches, supplies flexible solutions regarding response time. In this category, data transformation (including data cleaning and data conforming) are done at early stage. Data is pre calculated and saved in DW stores. Then, user queries are executed over aggregated data. During data-warehousing process, ETL (Extraction, Transformation and Loading) layer is responsible of gathering data from disparate sources. Finally, ETL processes are executed at periodical times for refreshing DW with new events recorded at sources.

B. Motivation

In order to develop a successful strategy, managers need constantly to know the situation of their business. For this reason, they have to analyze their own data (data collected from operational applications). To meet this need, enterprises as organizations invest in DW projects. Thus, DW defined by Inmon [1] as "collection of integrated, subject-oriented databases designated to support the decision making process" aims to improve decision process by supplying unique access to several sources.

The data latency requirement describes how quickly the data must be delivered to end users [2]. In first projects implementing DW solutions, this requirement has not a great weight. End users are satisfied to analyze on delayed data. But step by step the context is changing and the need to access fresh data is increasing. Indeed, business is growing and competiveness is becoming hug. So in order to survive and to face this international mutation, enterprises as organizations should sophisticate and speed up their responsiveness. Thus, the need to access fresh data (slow data latency) in DW system is a serious requirement.

In next section and in order to meet the requirement expressed above, we start by introducing the architecture of DW system.

C. DW Architecture

The architecture of DW system is depicted in Fig. 1 where disparate sources are integrated into a single repository called data warehouse (DW).

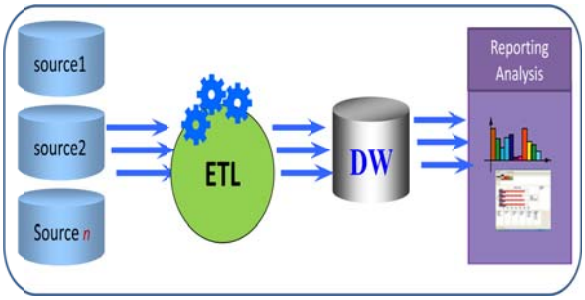

Figure 1. Classical Data warehouse Architecture.

doi:10.4304/jsw.9.5.1055-1061

In this setting centralized around DW, ETL tools (ETL layer) perform three tasks. Firstly they pull data from several sources (databases, flat files, ERP …etc), secondly they apply complex transformations to data extracted in previous step. Finally they load data to the target which is DW. The last layer of Fig 1 is REPORTING and Analysis layer which has the mission to catch end-user query and translate it to DW layer. Once queries are executed, collected data are served and formatted into reports like histograms and dashboard for end users.

In classical DW configuration, ETL was constraint to run at special time, often at midnight, where sources (data producers) are not actives. Consequently, DW and sources are asynchronous. In other words, events happing at sources take ***T*** time to appear in the DW. Thus, ***T*** is time duration between two consecutive DW refreshments performed by ETL processes. Consequently, ***T*** measures data latency, the delay to populate DW with fresh data.

D. Paper Organization

The remainder of this paper is organized as follow. Section 2 formulates the problem which is data freshness requirements in DW environment. Section 3 is reserved to our proposal while in section 4; we discuss related works to the issue involved in this work. We conclude and present our future works in section 5.

II. Problem and Solution Statements

A. Non Freshness of Data is the Problem

In first generation of data warehouses, DW refreshment events are planned at specific times, exactly at times where organizational applications (producing data) are inactive. The priority is given to data producers. But by the time, the context is changing and end users needs are changing too. Simply, end users need accessing fresh data.

An intuitive solution to ensure data freshness is to shortcut DW refreshment event by running ETL processes many times per day. But, since data latency requirement is urgent, the architecture of the DW system must evolves to new state in which almost every step of the data-delivery pipeline must be re-implemented [2].

Since DW serves as the foundation for improved decision making, DW should supply right information in right time. Thus, the objective is to reduce delay between the following events:

- When business events occur (at sources),
- Data are available on the DW,
- Decisions are made using new data.

This third factor depends on end-user. We ignore it in the following sections. Thus, we focus on the first and the second factors expressed above.

In conclusion, non freshness of data is a main drawback of classical DW solutions. To overcome this situation, it is helpful and desirable to reduce latency time in DW system. Therefore all components of DW should cooperate effectively to achieve performance and speed.

B. Solution Overview

In this paper, we intend to overcome data freshness drawback in DW system. We aim to have a near real time DW system where events recorded at source level become available on time at DW level. To meet this need, we suggest end-to-end DW solution focusing on data freshness issue with optimized coordination between its layers. Namely we suggest:

- Near real time oriented architecture of DW based on partitioning data inside DW into local and independent sources.
- Managing ETL processes in near real time context. We use KANTARA, our framework for managing ETL.
- Using Al-WASSIT, a light version of WASSIT mediator developed in our laboratory, as REPORTING layer

In next section we illustrate this new DW architecture by presenting our solution for each layer.

III. Our Proposal for near Real Time DW

In this section, we expose our proposal for handling data freshness in DW system. Particularly, section *A* gives an overview of this architecture while next sections give details of each DW layers. Thus, section *B* deals with ETL while section *C* deals with DW store. Finally, section *D* deals with REPORTING layer. At the end, in section *E* we sketch DW schema management in near real time setting.

A. Global Architecture

At high level, the architecture of DW in near real time environment that we suggest is similar to the classical one illustrated in Fig. 1. This architecture is based on same layers which cooperate to fill the same mission that is the warehousing of data. However, the main difference is the internal architecture.

Fig. 2 highlights our solution. It has three levels. From top to down, the first level retrace classical data warehouse layers depicted in Fig 1, namely Sources, ETL, DW and REPORTING. The second level recalls the basic mission of each layer present in first level. Finally, the third level shows the candidate or the technique we will use to perform the mission of associated layer.

KANTARA and AL-WASSIT used respectively for ETL and REPORTING layers are two frameworks

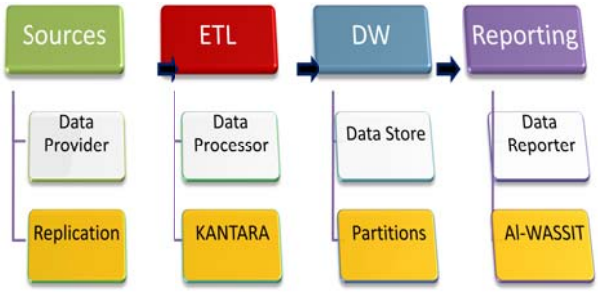

Figure 2. Solution Overview.

developed in our laboratory SIR

DW layer deals with data storage. In our vision, we split DW into many parts or partitions.

Finally, for sources it is interesting to note that they are autonomous. Furthermore they have the priority to run their processes (which produce data) than to serve any business intelligence application. To overcome this situation we consider cloning sources. In other words, we replicate sources. At this level we consider that the owner of data (data provider) collaborate with ETL and DW teams. Making near real time DW is a vital decision which requires cooperation and high integration.

In next sections we give details of each layer.

B. ETL

The ETL mission is to feed DW with data coming from sources. In first DW projects, DW refreshment (performed by ETL processes) takes place regularly at specific times. But currently, DW users' needs have been changed. Simply, end users want accessing fresh data via DW solution (to keep DW advantages). Thus the ETL mission, in near real time DW configuration, becomes critical and very complex. Stated differently, after the first load of DW, new events occurring at sources should take little time to figure in the DW. Consequently the challenges are:

1. To detect changes at sources level,
2. To propagate changes toward the targets.

Regarding the first challenge, many solutions have been proposed. They are associated to incremental loading techniques of DW. Thus as mentioned in [2], one can use audit columns technique or log sniffing method. Another option is to use triggers on databases. We refer to [2] for definitions and more details. All these approaches are effective [2], but they require cooperation from sources owners. They have to supply this service and implement these techniques in their operational applications. Making near real time DW is a strategic investment which comes with cost.

Finlay in table 1, we summarize techniques of extracting changed data. It is clear that all techniques don't supply same service. Therefore and depending on the technique used; only a part of changed data can be detected.

TABLE I.
TECHNIQUES CAPABILITIES FOR DETECTING CHANGES

	Insert	Delete	First update	Other updates
Audit Columns	Yes	No	Yes	No
Snapshot Differential (Process of Elimination)	Yes	Yes	Yes	No
Log scraping or sniffing	Yes	Yes	Yes	Yes
Initial and Incremental Loads	Yes	Yes	Yes	No

The second challenge deals with integrating modified data. The idea is to deliver changed data into DW (dimensions or facts) with less delay. Therefore the goal is to adapt ETL processes which work on bulk mode (extraction of full data from sources) to ETL processes which work on delta mode where only changed data are extracted. It is interesting to note that switching from bulk mode to delta mode is done at logical level.

As shown in Fig. 3, we represent the problem according to KANTARA (our framework) which is aligned to MDA (Model Driven Architecture) paradigm. Let recall that "MDA separates certain key models of a system, and brings a consistent structure to these models… explicitly and mainly into Platform Independent Models (PIMs), and Platform Specific Models (PSMs) which is derived from the PIM via some transformation" [17]. Using MDA terminologies:

- **Conceptual Model of ETL** process constitutes the PIM of the model.
- **Logical design of ETL on Bulk Mode** and **Logical design of ETL on Delta Mode** respectively PSM_1 and PSM_2 constitute two instances of PSM model.
- ***T1 , T2*** and ***Adaptor*** constitute transformations between models (see Fig. 3)

The Fig. 3 states that there are two direct paths to get logical design of ETL processes on delta mode (DM). Indeed, this is possible from Conceptual Model of ETL process (CM) through **T2** transformations or from Logical design of ETL on Bulk Mode (BM) through **Adaptor** transformations. Obviously indirect paths are possible by combining {***T2*** and ***Adaptor***} with ***T1***. However let note that direct path means less transformations.

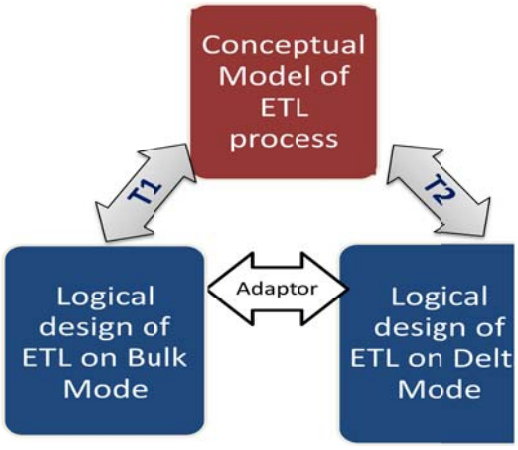

Figure 3. Relationship between ETL modes.

Actually in the literature we find some works proposing conversion from BM to DM. Unlike theses works, we suggest to build ***Adaptor*** transformations for existents projects and to build ***T2*** transformations for new projects.

The specification and the implementation of ***T2*** as ***Adaptor*** are not in the scope of this paper. In addition,

making an ETL layer, should take in account DW organization. Next section presents our vision toward DW layer.

C. Data Warehouse or Data Store

In his evaluation of DW, Inmon [4] et al recognizes a life cycle of data within the DW. The basic idea is, as data ages (once it is inside DW) its access probability diminishes. Therefore data inside DW can be divided into partitions. The benefit is achieving performance in query processing. Still, how many partitions to make? How to identify the content of a partition?

In classical DW environment, DW is a central element that acts as a single store. In near real time environment, DW stills a central element but acts as multi store. Clearly we split DW into several partitions. The number of partitions is governed by a set of parameters managed by DW administrator. These parameters and the process of making partitions are described below.

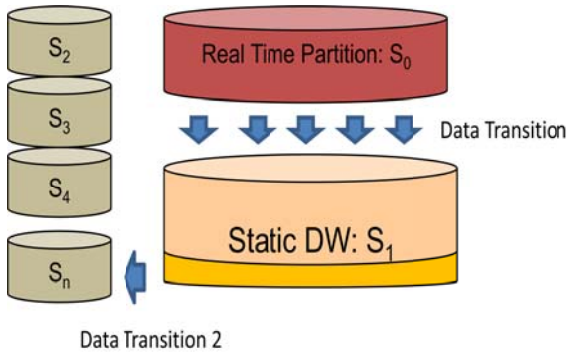

Figure 4. Data Repartition within Data Warehouse

Fig 3 illustrates how we distribute data within DW. We consider each partition as a source. Let note that at early age, only S_0 and S_1 exist.

- **S_1**: Static DW store. It represents the classical data store, except that it contains recent data produced in a given time interval. For example, if such time interval is 5 years then S_1 contains data of current 5 years.
- **S_0**: It plays the role of real time partition which has the mission to collect fresh data from data sources, on near real time. Thus, events recorded at source level are immediately processed and forwarded to this store area. Consequently S_0 contains today data. Finally S_0 can be seen as a gate that data transit to arrive into DW.
- **Data Transition (DT):** it is a data flow which has the mission to feed S_1 by S_0. More precisely, all S_0 data are purged into S_1. Data transition event takes place every day at specific time. For example, this event can be planned every midnight.
- **S_2:** At the beginning, DW contains only S_0 and S_1. After T^1 time (for example 6 years), data inserted at early phases becomes old. Thus S_2 is derived from S_1 by purging old data from S_1 into S_2. Old data, which are less active than other data, are relatively defined by DW administrator. For example, old data may be data belonging to [0, 2 years] interval.
- **Data Transition2 (DT2):** Conversely to *Data Transition* flow, this data flow is not frequent. It performs data migration from S_1 to next candidate partition S_{in}. Depending on partition time interval, it may happen few times. For example, if partition time interval is 2 years, DT2 is executed twice (one by year). Finally, this event can be planned at specific date every year.
- **S_i:** for i >2; S_i is produced similarly to S_2. One can note that $S_i \neq S_j$ for $i \neq j$.

The content of DW at certain time ***t*** is the content of all partitions. With abuse of notation, we note in (1)

$$DW = \sum_{i=0}^{i=n} Si \quad (1)$$

The relation between ***t*** and ***n*** is defined in (2).

$$\boldsymbol{n} = floor(\frac{t-T}{L}) + 2 \quad (2)$$

floor refers to greatest integer function. ***T*** is interval time of source S_1 while ***L*** is time interval of each partition S_i. Obviously, we suppose that these parameters are constants.

During this section, we have presented how we store data inside DW (by splitting DW into partitions, which we consider as local sources). This has an impact on REPORTING layer. Indeed, in classical configuration, REPORTING layer consumes data from a single store that is DW. In next section we will see how to handle reporting task in this new configuration.

D. Reporting and Analysis

The closing layer in DW environment is the REPORTING (RA) layer. To avoid confusion, we use this term to refer to different ways to access data in DW. By construction RA is the representation layer which has the mission to catch end-user queries and translate it to DW. Resulting data are formatted and served to end-users in several formats like reports or dashboards…etc.

In classical DW configuration, RA layer pulls data from a single source that is the data warehouse. In our setting we had split the DW store into several partitions which behave as sources. Therefore the RA layer should evolve to work in compatibility with this new setting. The challenge is:

- To query several sources instead of a central repository whether the RA layer is heavy or thin.

RA layer is heavy when DW layer is thin and vice versa. DW is heavy when the schemas of all partitions are synchronized (more description in section *E*).

For RA layer we suggest to re-use WASSIT mediator developed in our laboratory. The main reason for that is we have to deal with several sources. Each source

[1] It should be estimated by DW administrator

contains data belonging to DW in projection to a specific time interval (snap shot of DW in certain time interval).

WASSIT mediator is based on the wrapper-mediator architecture. The description of WASSIT architecture and more details about it can be found in [3], [4]. WASSIT intends to integrate disparate sources. But currently, we want to adapt WASSIT mediator to fill the mission of REPORTING layer in near real time DW architecture. For this end, we call Al-WASSIT (depicted in figure 3) the derived version from WASSIT mediator.

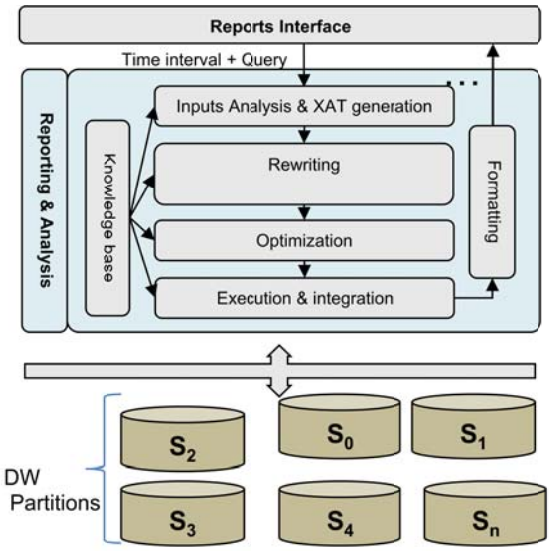

Figure 5. A High level Architecture of Al-WASSIT.

Description of Al-WASSIT components:

- **Time interval & Query:** constitute the input of Al-WASSIT. *Query* is the end-user request while *time interval* indicates how long the query is. It dictates the age of data to be taken in account during data analysis. Stated differently, this last input governs the number of sources which will be solicited while executing the *query*.

- **Inputs Analysis & XAT generation**: This module has to analyze the input query. It rejects bad inputs which are syntactically incorrect. Too, it eliminates the queries that refer to unavailable concepts. This is done using the knowledge base. User's queries are then transformed into XML algebra trees where each node of a XAT tree is an algebraic operator.

- **Rewriting:** Let's remind that we aim to access a set of sources while every source has (by default) its local schema that describes its structure in a data model. The query submitted by end user is formulated in terms of DW schema (global schema that is the schema of S_1). In order to execute a user's query, Al-WASSIT must rewrite this query, expressed in terms of DW schema, as several sub-queries where each sub-query is dedicated to a particular source and formulated in terms of its local schema. The rewriting module takes as input the XAT tree corresponding to the user's query. Also, a mapping between the DW schema and the local schema may be included in this process of rewriting which produces at the end; Query Execution Plan or QEP to be sent to each involved source.

- **Optimization:** This module is useful only when each data source has its own schema. Otherwise sub queries are identical then they are directly sent to local sources. Next section explains when local sources have the same schema. In first case where local sources are different, this module takes three inputs: a) the QEP obtained after query rewriting, b) the description of sources capabilities and finally c) the cost model. Then it produces the optimal query plan for each sub-query.

- **Execution & integration:** This module sends sub-queries to appropriate sources using localization information supplied by the knowledge base. Also it uses a cache in order to store the intermediate results. After sub queries execution, it combines the results in order to forward them to coming module.

- **Formatting:** This module has the mission to format results returned by the *Execution and Integration* module, according to user specification. For instance, this module is responsible of data graphical representation.

- **The knowledge base:** The knowledge base stores general information used for queries processing. It is a repository for metadata involved in Al-WASSIT. Particularly, it contains schemas of DW partitions, formatting options, mapping…etc.

In conclusion, Al-WASSIT performs RA layer mission. But as we said previously, there are two modes of making this layer: RA heavy or RA thin. Next section deals with this aspect.

E. Schema Management: DW or REPORTING Heavy

The life cycle of data within S_0 (real time partition) is very short (one day in maximum). Furthermore, the schema of S_0 is identical to S_1 schema. Therefore data transition from S_0 to S_1 is direct. It does not require any transformation. However data transition, from S_1 to S_n, is more complex and requires dealing with schema differences between the source S_1 and the target S_n.

For i >= 1; we note that $SHC(S_i) \neq SHC(S_{i+1})$ where $SHC(S_i)$ refers the schema of the source S_i. Therefore each source has its own schema. This is the default approach for managing schemas. Another approach behaves differently. Indeed we can standardize schemas

of all sources. Then for i >= 1; we note $SHC(S_i) = SHC(S_{i+1})$.

The default approach for managing schema is simple and direct. The second one has the cost of standardizing schemas. In other words, any evolution or change in $SHC(S_1)$ should be propagated to other S_i. This includes the application of change subject to data stored in each S_i.

It is true that the task of standardizing schemas and data takes extra times, but it has the advantage to simplify the task of REPORTING layer. Indeed, when all S_i share the same schema, calculations and data operations performed in REPORTING layer are executed in short laps of time in comparison of the case where each S_i has its own schema. More precisely, in the second approach the step of rewriting in REPORTING layer is simplified and saves time contrarily to the default one.

Finally we note that the default approach of managing schemas makes heavy the REPORTING layer, while the second one makes the DW layer heavy.

IV. Related Works

Single tools as well as end to end DW solutions exist. Indeed, a plethora of commercial ETL tools [11], [12] as well as a set of open sources [9], [10] exist. The same remark is valid for REPORTING layer [11, 12].

SAP and PENTAHO propose end to end solution. Respectively SAP-BW [16](commercial) and PENTAHO [15] (open source). All these solutions are interesting. However, they solve only a part of DW problem in near real time context. In addition they consider DW as a single store (classical configuration).

For ETL in near real time context discussed in this work, authors in [7] formalize the problem of ETL running on delta mode. They present an approach to the automated derivation of incremental ETL based on equational reasoning. Transformation rules for this purpose are defined too. While in [8] authors use the existing method of incremental maintenance of materialized views to implement the automatic creation of incremental ETL processes. In comparison to our approach, [7] like [8], specify and implement *Adaptor* transformation (transformation from PSM1 to PSM2 see Fig. 3). Therefore these approaches require the existence of the first ETL processes of initial load. Furthermore, the management of change and evolution, with these approaches, is not easy.

In reference [2], Kimball et al, examines the historical and business contexts of the real-time data warehouse. Particularly, they justify why real time ETL? The authors focus on evaluating several mechanisms for delivering real-time reporting and integration services. The strengths and weaknesses of each approach are presented. In both cases the architecture of underlying system is based on two partitions of data: static and real time partition. The same idea, of DW architecture, is expressed in [5] where authors use the concept of cashes to receive actual data. An algorithm for transiting data from certain cash to its successor is detailed too. However these approaches do not deal with static partition neither with REPORTING layer. Finally we note that authors of [6], adopt the same concepts (static and dynamic data division) to partition data.

In reference [1], Inmon et al, suggest architecture for new generation of data warehouses. First, the life cycle of data within DW is repartitioned in four sectors. Interactive sector is the entry point of data into DW. Data arrives rapidly to interactive sector either by ETL processes or either by a direct application housed inside. Secondly, the transition of data from one sector to another is based on access probability. The old is data, the low is its access probability. This interesting reference defines many important architectural features of DW. However it does not deal deeply with ETL layer. It suggests to use changed data capture technique (log file) when possible without showing how to create ETL in this new mode.

V. Conclusion

Data warehouse (DW) is recognized as an effective solution for integrating diverse information systems. But DW has the drawback of data freshness. There is an important delay between events recorded at source level and availability of such events (data) on DW. Consequently, it is mandatory to have a new DW architecture and an effective approach for ensuring data freshness and performance. Stated differently, there is a need of DW solution in near real time setting which comes with the cost of more cooperation between different stakeholders and a strong integration between DW layers.

In this paper, we have presented architecture of DW oriented near real time. Our proposal aims to overcome high latency in DW environment. Thus, we have shown DW layers and how they cooperate. Also, we defined the scope and the challenge of each layer simultaneously with our suggestion to fill its mission.

For DW layer we suggest to split DW store into local sources. We have shown the mechanism to create sources or partitions. Also we discuss schemas management in DW layer that leads to double status of DW layer. Namely DW heavy or DW thin depending on standardization or not of partitions schemas.

For REPORTING layer, we have exposed how to use Al-WASSIT, a light version of WASSIT mediator, as a REPORTING layer in near real time DW setting.

In this setting and for ETL layer, we have discussed the techniques of detecting changed data and how to integrate them. Namely, via our framework KANTARA oriented MDA; we have presented how to get ETL processes running on delta mode.

Finally, in future works, we plan to work on metadata management in near real time DW setting. Particularly, managing data and DW schemas being subject of change is an interesting topic. *What* and *how* to manage metadata will be the main challenges. Besides this, another question will be whether to centralize all metadata involved in DW environment in a single repository or to manage each layer metadata separately.

REFERENCES

[1] W. Inmon, D. Strauss and G. Neushloss. "*DW 2.0 The Architecture for the next generation of data warehousing*". Morgan Kaufman, 2007.

[2] R. Kimball and J. Caserta. "*The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*". Wiley Publishing, 2004.

[3] F. Wadjinny, L. Gounbark, D. Chiadmi, L. Benhlima and A. Moujane "Query processing in the WASSIT mediation framework". in the seventh ACS/IEEE International Conference on Computer Systems and applications (AICCSA), May 10-13Rabat, Morocco, 2009.

[4] L. Benhlima and D. Chiadmi "Vers l'interopérabilité des systèmes d'information hétérogènes". e-TI : la revue électronique des technologies d'information, Number 3, available: http://www.revue-eti.netdocument.php?id=1166. (2006,december),.

[5] Cuiru Wang; Shuangxi Liu "SOA Based Electric Power Real-Time Data Warehouse". Power Electronics and Intelligent Transportation System, 2008. PEITS '08. Workshop on, vol., no., pp.355-359, 2-3 Aug. 2008.

[6] Delin Qin. "Design of Medical Insurance Supervision System Based on Active Data Warehouse and SOA". Computer Science and Information Engineering, 2009 WRI World Congress on , vol.3, no., pp.45-49, March 31 2009-April 2 2009.

[7] T. Jorg and S. Debloch. "Formalizing ETL Jobs for Incremental Loading of Data Warehouses". BTW, 2009, 327-346.

[8] X. Zhang, W. Sun, W. Wang, Y. Feng, and B. Shi. "Generating Incremental ETL Processes Automatically". Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06).

[9] Talend Open Studio, www.talend.com

[10] Vanilla Open source, www.bpm-conseil.com

[11] IBM InfoSphere DataStage, http://www-01.ibm.com/software/data/infosphere/datastage/

[12] Informatica, http://www.informatica.com/FR/Pages/index.aspx

[13] SAP Business Objects, http://www.sap.com/france/solutions/sapbusinessobjects/index.epx.

[14] IBM Cognos, http://www-01.ibm.com/software/fr/data/cognos/

[15] Pentaho, http://www.pentaho.com/

[16] SAP Business Information Warehouse, www.sap.com.

[17] OMG, http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01
.