Patterns-based Real-Time Data Warehouse Architecture

Djillali Nora
LRDSI Laboratory-
dept. Science Computer
Blida University (Algeria)
djillalinora@gmail.com

Mohammed Mahieddine
LRDSI Laboratory-
dept. Science Computer
Blida University (Algeria)
mo_mahieddine@mail.univ-blida.dz

Oukid Salyha
LRDSI Laboratory-
dept. Science Computer
Blida University (Algeria)
osalyha@yahoo.com

Abstract—**In this paper, we proposed a pattern-based architecture for real-time data warehousing systems. We detailed functions of components and their realizability in the architecture by means of design patterns and we illustrated the solution proposed for prescription medical data warehouse. We also highlight the advantages induced by of the used patterns in the architecture.**

Keywords—real time data warehouse, design patterns.

I. INTRODUCTION

To influence what should happen next, the enterprise data warehouse needs to know what is happening right now.In the real time enterprise (or "zero latency" organization), data coming from data bases is processed and transmitted quickly enough so that decision makers can successfully evaluate and act on it [1].This has led to the development of specialized techniques named real-time data warehouses. Real-time data warehouses are characterized by continuous, asynchronous, multi-point delivery of data providing the user with the most up-to-date information possible. Almost, immediately after the original data is written, that data moves straight from the originating source to the data warehouse [10].

The main advantages of real-time warehouses are updated decision-making, faster decisions, and feeding real-time dashboard of the company.

The ability to access meaningful data in a timely, efficient manner through the use of familiar query and analysis tools is critical to realizing competitive advantages.

The work presented in this paper concerns the pattern-based modeling of real-time data warehouse architecture.

The main advantages of design patterns for the designs of architectures are: a clear concept, effortless understood and relatively easy, simple integration of anew existing components and lower maintenance costs.

We describe this article; the important sides related to the construction of real-time data warehouse, the architecture of data model modeled based patterns and process food.

Some architecture for data warehousing systems has published in the literature [7] [5]. Sharma has proposed 3-tier architecture [11]. The types of known architectures for the real time data warehouse are:

- **Flow regulators between the source and the Data Warehouse** A Warehouse Flow Regulator orchestrates the propagation of data from the source to the warehouse.
- **Real-time partitions (RTP):** by maintaining two versions of the star schema representing a data warehouse. One version should be static and the other at real time, according to their population. And periodically fills (in short periods of time) the static fact table before being emptied.
- **Fast transformations via Capture-Transform-Flow (CTF) processes.** CTF solutions move operational data from the sources apply light-weight transformations and then, organize the data in a staging area. After that, more complex transformations are applied (triggered by the insertions of data in the staging area) and the data are moved to a real time partition and from there, to static data stores in the data warehouse.

Compared with them, the real-time architecture we propose is completely modeled by means of design patterns. In our case, we have adopted a highly decoupled layered architecture but we have integrated a pattern-based modeling.

We'll concern mainly to build the data model, then the operational data bases to the model that represents the data-warehouse data and that conceptually and logically, we describe our technique that is based on design patterns and our choice of methods where by data from the databases will be extracted, transferred over the network to be adapted and eventually integrated into the warehouse to be used by decision makers.

In the paper, we propose an efficient and flexible architecture for real-time data warehouses. This architecture is presented conceptually with detail. Then, we describe the flow throughout the architecture. We give detailed description of the functions of the components of the architecture. We examine the advantages of the architecture based on functions of each component and the design patterns used for its modeling.

In the designing of the architectural components, many reusable design patterns are used. These patterns are Mediator, Adapter, Wrapper, Singleton, and the Observer [4]. These reusable and well-defined patterns help the producing of a robust architecture.

The proposed architecture (see "Fig. 1"), is based on a layered approach to clearly define the responsibilities and services of each distinct class of components. The clear advantages of this layering are modularity to allow for maximum development and maintenance flexibility, interoperability, the ability to communicate easily with other systems and integrability, the ability to upgrade the system with new functionalities without major changes to the overall architecture. The following sub-sections will discuss each layer in greater detail.

One of the aims of this project is its using for the implementation in *Java* [6] of a data-warehouse for representing the relevant information associated with the clinical handling of patient affections of a concrete pathology: disease, diagnosis, treatment, drugs prescription and planning, and the history of patient assistance [9].

The remainder of the paper is organized as follows: In Section 2, we provide an overall view of the components-layers of the proposed architecture. Section 3 gives a detailed description of the components of this architecture with reference to our implementation of this model, the patterns used for their modeling, and their advantages based on design patterns. We conclude in Section 4 with a summary and plans for future work.

II. ARCHITECTURE

In this section, we propose modular layered pattern-based architecture for real-time warehousing systems.

Data warehousing is all about capturing information about the organization. Real-time implied that it is captured as it happens. Real-time decision support is a framework for deriving information from data as soon as it becomes available.

In addition to detecting events, an event system should provide facilities for notification and storage of the event.

The conceptual level of the architecture is shown in "Fig. 1" which includes components of data sources and extraction components, the data integrator, the warehouse storage, the queering analyzing and reporting tools. The data sources supply data for the data warehouse. Data stored in the warehouse is then accessed by OLAP operators, processed in a multidimensional way, and supplied to the users to be analyzed.

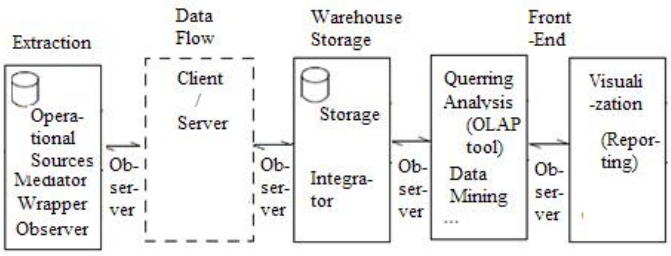

Fig. 1. Architectural Layered Components.

The architecture, we propose, is compound of four layered sections: extraction section, data flow section, storage section, and front-end section.

Using the Observer pattern we have positioned change tracking inside methods that are executed inside the source database when a table change occurs, and defined updating observers that process the event. This provides a real-time copy of the transactional data to the data warehouse without introducing staging areas.

Changes frequently involve several data stores at one time. For example, when a data is created, updated, it involves the updating of the corresponding table in the data warehouse side.

The design of the architecture components uses a set of good object-oriented design patterns that allow developers to extend functionalities and features easily.

The proposed architecture is layered and is modeled by a transformed highly decoupled version of the Layered pattern [2], in with relations between layers are modeled by the Observer/Observable design pattern (See "Fig. 2").

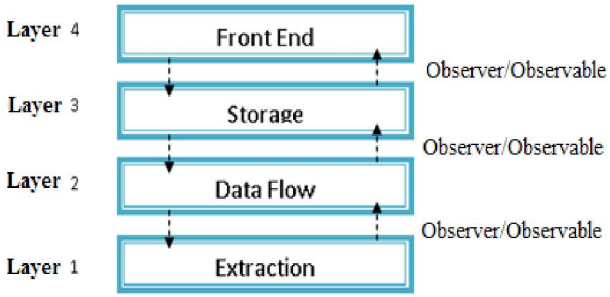

Fig. 2. A highly decoupled Layered Design Pattern.

The extraction layer contains the data of data bases and the extraction system. In our solution we propose an extractor based on a wrapper for each data source, and event listeners of updated data. The data flow section is a client/Server component responsible for transferring of updated or inserted data by the extraction section in real time for the storage section of the data warehouse. The storage section contains the integrator and the warehouse storage component. The front-end section includes the client requests, analyze, and visualization components.

Queries in the front end are started by decision maker users. They are transformed and divided into data requests and sent to the analyze component, that automatically update the views of the visualizing component.

Systems integration of information-based mediators[3] stand out: first, how is established correspondence between the global schema and the schemas of data sources to integrate, the other by the languages used to model the overall pattern, patterns of data sources to integrate and user requests.

On our side, for the mediation layer, we modeled the basic component relational schemas, in this case the table by a general abstract classDBTable (See "Fig. 7") to meet with the classDB (See Fig.8) to a general interface producing a very simple query language that provides the user (See Tab. 2).

Global query in this general queering language, formulated by the data base users, is sent to the mediator, which in turn, converts it to a set of sub queries for the wrapper of individual schemas of the local sources.

Data at the back-end flows from the data sources, as a result of insertions or updates, to the extraction component. The data is then transmitted across the data flow component to the integrator of the storage component. The data is filtered and transformed in the integrator in uniformed formats of the data warehouse.

The integrator, situated in the warehouse side, cleans the data, conciliates the conflicting data from different sources, integrates and aggregates data from related sources and sends the results to the storage component.

The data residing in the warehouse storage component can be used by the front-end component to initiate view or to refresh views of the visualizing component based on the query types of the queering component or the analyze component (OLAP server, etc. …).

The proposed solution reduces development costs and above-tenance of the warehouse.

Ultimately, this remedial work to add bricks to the system more flexible and give without breaking the flexibility, modularity, and scalability that requires a good design.

In the next section, we describe the function of each component by highlighting the advantages of the design patterns used to that end.

III. DESCRIPTION OF THE ARCHITECTURE

In this section, we detail the functions of warehouse components and analyze their advantages in the architecture being modeled by means of patterns.

A. Extraction components

The functions of extraction components are to extract up to date data from data sources and to maintain data in warehouses so that it is always consistent with data sources.

The solution adopted for the uniform treatment of requests for tables is the Mediator.
Real time notification can take a variety of forms.

Our solution is based on the earphones of requests for updates and integration, achieved through the Observer design pattern.

For modeling the extraction component, we propose four elements: a mediator, wrappers, listeners, and data sources.

- A **Mediator** receives local language queries from the data base user interfaces, translates them into a queering local language (See Tab. 2) and sends them for the wrapper of each data base.
- A **Wrapper** receives local language queries from the Mediator, translates them into SQL language, and sends the local queries to the data source to be evaluated. Then the wrapper receives queried data from the data source, filters out the unrelated data items, and transforms the data into uniformed formats of data warehouses. The wrapper is made around the class DB and the class DBTable (See "Fig. 7" and "Fig. 8") that will allow the adoption of a general interface for operations to create, insert, delete and update tables relational. So these classes will allow adapting the basic operations for processing relational DBMS tables. DBTable class is an abstract class. We can say that it is a virtual table that will allow all operations to adapt the tables to a physical relational data base through a common interface making operations.
- A **Listener** detects update of a base relation in a data source and notifies the data flow component of this update. The update is then propagated, by the listener, through the wrapper to the data flow component who will transmit it in return to the integrator (See "Fig. 4").

To detect the update, the Listener has been modeled with the *Observer* design pattern [4].

- A **data source** answers queries from the wrapper, returns the required data.

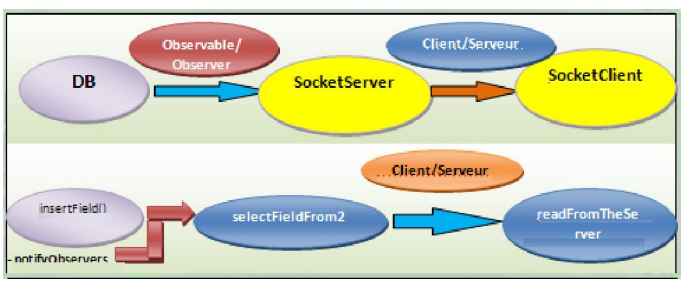

Fig. 3. Diagram of the data flow with associated patterns.

B. Data Flow Component

The Data Flow Components models the transfer of data and controls the access of the data in the warehouse storage, and the simultaneity of the transfer.
We have modeled by a Client-Server architecture adapted, in which the client receives all that passed through the server, and the Observer Pattern.
The DBclass is observed [6] Class SocketServer order to inform them of any changes made in the source tables. It is designed as a singleton at the same time (See "Fig. 3", "Fig. 7" and "Fig. 8").

C. Storage components

Data warehouses store the integrated, summarized, historical, and non-volatile data used for decision support. The structure of DWs is based on a multidimensional view of data usually represented as a star or a snowflake schema, consisting of fact and dimension tables.

A fact table contains numeric data called measures. Dimensions contain attributes that allow exploring measures from different perspectives.

This layer consists of two main components this component is modeled by two main components: the integration component [8], and the storage component.

At the logical level the basic unit of storage model is the table as in the relational model.

The **integrator** in a data warehouse performs data cleaning and transformation before loading data in the storage schema tables.

The integrator receives inserted or updated data in data bases from the data flow component, filters out the unrelated data items, and transforms the data into uniformed formats of data warehouses. The integrator has been achieved through Observer and Mediator design patterns [4] (See "Fig. 9").

The component storage is achieved by the tables that are the basis of known patterns of data warehouses. For purposes of simplicity, our architecture, we have favored the establishment classical model is to consider a star.

D. Front-end components

Front-end components supply user-friend queering interface, abundant tools and applications for functional analysis and data visualizing. Querying is the process of getting data from a data store, which satisfies certain criteria. Here is an example of a simple query: "Patients with blood group O-?" (See "Fig. 5"). Reporting is retrieving data from the data warehouse and presents it to the users. The reports display the data required by the business user to analyze and understand business situations. The most common form of report is a tabular form containing simple columns.
The front-end component comprises the following components:

- **Queering component** contains applications for query specification, accepts and processes queries from client applications and puts data request to the warehouse analyze component. This component is actually modeled by pre-existing general queries updating related views of the visualizing component.
- **Analyze component** contains applications for data analysis, data mining, trend forecast. Various front-end tools such as spread sheets, pivot tables, reporting tools, and SQL query tools to retrieve and analyze the data. An analysis server is a way for analysts and policy makers to navigate, drill, explore the data warehouse. For navigation and data mining tools are certainly the most suitable tools OLAP with all their

potential in the inter section of data, however, this component has not yet been achieved, we have currently opted for pre-registration search queries made by the maker, and we have achieved through SQL queries (See "Fig. 5" and Tab. 1). On visualization tools, which are important for the presentation of results or the discovery of new data, but never the less, remain in the field of research, once their achievability is identified, they can easily be integrated with the boss Observer this layer is for users of the warehouse. It allows you to query and retrieve the query results. In our work, the request of a user by the Wrapper will be converted in to an equivalent SQL query defined in the schema, and

- **Visualizing component** contains applications for report formatting. On visualization tools, which are important for the presentation of results or the discovery of new data, but never the less remain in the field of research, and that once their achievability is identified, they can easily be integrated with Observer pattern.

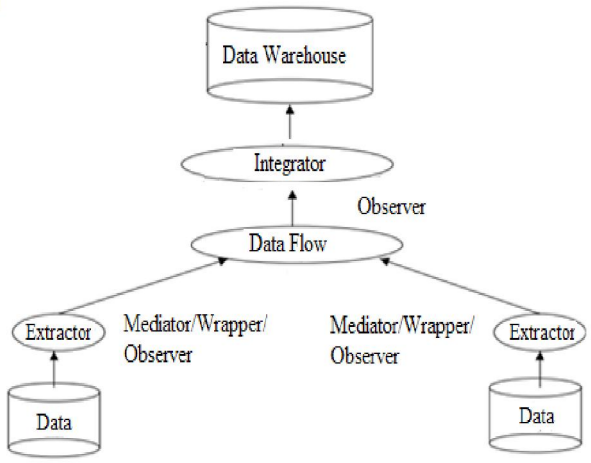

Fig. 4. Patterns on the basis of components of the architecture.

E. Advantages of the architecture

An apparent advantage of real-time data warehouse that there is no need to reconcile differences between source applications and the warehouse. A data warehouse that doesn't appear to be always up to date tends to lose the confidence of the users.

Since data warehouses and related tools are continuously evolving, it is important to have an adaptable architecture which can scale well with changes. A pattern-based layered architecture is best adapted for reusability and extendibility ends.

The combination of three patterns Mediator-Adapter-Observer lets us to realize the extraction component. The real-time integration was realized by means of the Observer pattern. We have modeled the data flow component by using client/Server architecture and the Singleton and Observer patterns. The visualizing tools, as views, are modeled by means of the Observer pattern. The Wrapper supports the extendibility of back-end, and the Observer highly decouples the layers both offering the ability for easy modifiability and expanding of the system

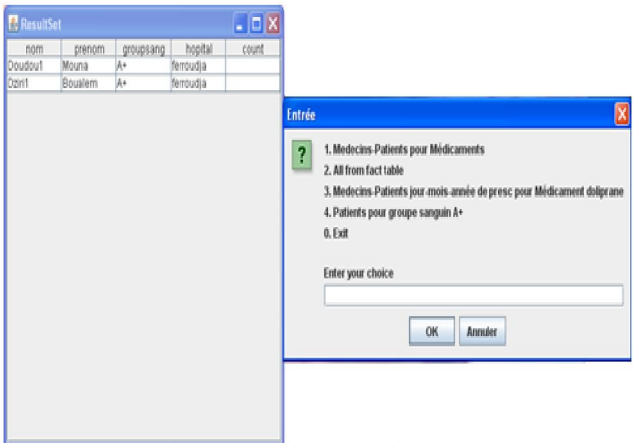

Fig. 5. Queering and Report for a decision-making.

TABLE. 1 –*EXAMPLE SQL QUERY.*

```
SELECT c.nom, c.prenom, b.nom, b.prenom, g.jour, g.mois, g.année
FROM MédecinDim b,PatientDim c,MédicamentPrescriDim d,
  OrdonnanceDim e, MédicamentDim f, PrescriptionMédicamentDateDim g
WHERE c.CodePt = e.CodePatient AND  e.NumOrd = d.NumOrd AND
 f.CodeMedi = d.CodeMedi  AND  g.DateSk = e.CodeDate AND
 b.MatMed = e.CodeMedecin
GROUP BY c.nom, c.prenom, b.nom, b.prenom, g.jour, g.mois, g.année, c.hopital,
 b.hopital, d.quantite, f.nom  HAVING f.nom = 'doliprane'  AND b.hopital = c.hopital ;
```

IV. CONCLUSION

The data warehouse at real time has become popular in recent years, predominantly due to a growing wish to have the latest information as possible to beat the competition. A real-time data warehouse aims at decreasing the time it takes to make the business decisions. In fact, there should be minimized latency between the cause and effect of a business decision.

The work presented in this article concerns the pattern-based architecture modeling of real-time data warehouses. Our objective is to be able to generate a layered architecture composed of several modules modeled by means of design patterns.

We have modeled and the most complex components of the architecture. The resulted architecture must provide warehousing systems with generality, extendibility, and reusability. We have effectively used this architecture for modeling and implementing a real-time data warehouse based for storing medical data prescription. The final objective of the project is to integrate up to date heterogeneous data from various domains as medicine, biology, and imaging. Future trends could revolve round, generalizing this model to include other patterns. The OLAP tool must be added in the nearest future. Moreover for generality, other characteristics could be added to the architecture to make generalized architecture for all types of real-time warehouses.

TABLE. 2 –EXAMPLE OF LANGUAGE OFFERED BY THE MEDIATOR THROUGH THE DB CLASS.

```
DB db = DB.getInstance(); //Singleton !
db.setPostgresql("localhost","5432","patients","postgres",  "mohammed");
//Création de la table source Patients qui est une DBTable
buffer=db.generateTable(Patients.TABLE_COLUMN_NAMES,
   Patients.OBJECT_TYPES, Patients.TABLE_COLUMN_NAMES[0]);
db.createTable(Patients.TABLE_NAME,buffer.toString());
try { //Alimentation de la table source  Patients
    Object[] patientFieldValue1={1,"Zriba","Djaouida","1978-04-04",
        "F","Blida",  "Blida","066541237",
         "Rue des frères X, N°15, Zabana, Blida","Artisan",
        "B-",60.5,1.58,"phtysiologie",221};
    db.insertField(patients,Patients.TABLE_NAME,
        Patients.TABLE_COLUMN_NAMES,  patientFieldValue1, false);
```

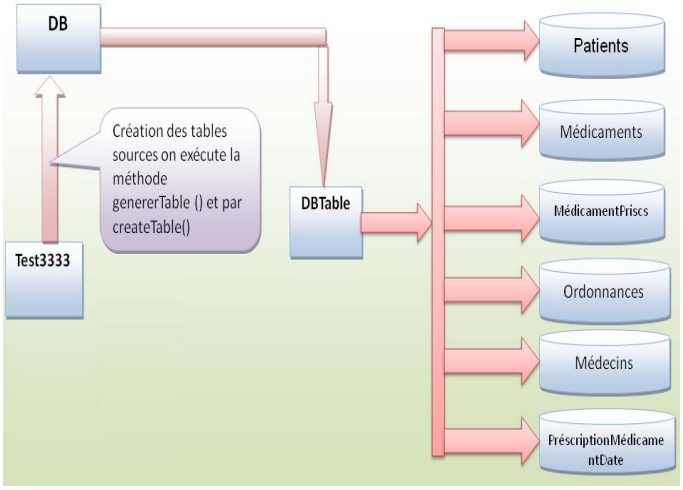

Fig. 7. The extraction component and the patterns.

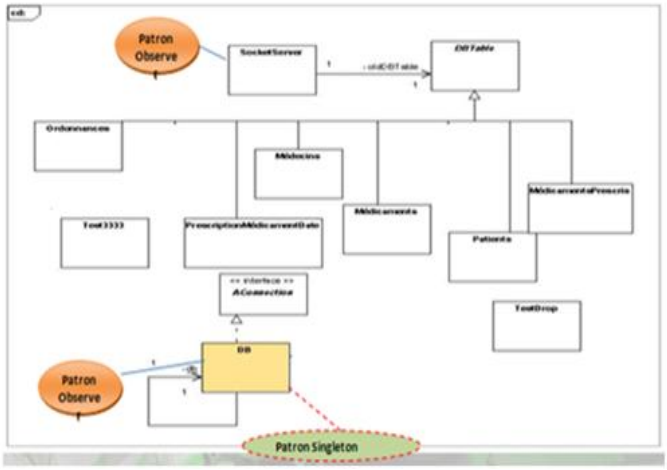

Fig. 8. The extraction component and its associated patterns.

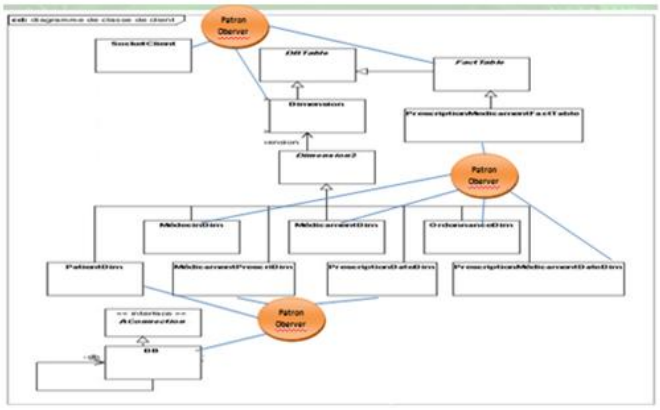

Fig. 9. The warehouse storage component and its associated patterns.

REFERENCEE

[1] G. Barnes Nelson, "Real Time Decision Support: Creating a Flexible Architecture for Real Time Analytics". Paper 109-29 , SUGI 29. ThotWave Technologies, LLC

[2] F. Buschmann, R. Meunier, H. Rohnert, and P. Sommerlad, "Pattern-Oriented Software Architecture - A System of Patterns", Wiley, John Sons, Incorporated, 1996.

[3] L. Djema, F.O.Boumghar, S. Debiane, "L'imagerie Médicale Dans une Base De Données Distribuée Multimédia Sous Oracle 9i", 4th International Conference: Sciences of Electronic, Technologies of Information and Telecommunications, SETIT 2007 – Tunisia.

[4] E. Gamma, R. Helm, R.E. Johnson, et J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison Wesley, Reading, Massachusetts (USA), 1994.

[5] B. Inmon, D. Strauss, G. Neushloss, DW 2.0 – "Architecture for the Next Generation of Data Warehousing", 2008.

[6]S.J. Metsker, "Design Patterns in Java. The software patterns series", 2006.

[7] J. Ranjan, "Real Time Scientific Data Architecture for Indian Scientific Community". Journal of Theoretical and Applied Information Technology, JATIT, 2009.

[8] P. Russom, "Data Integration for Real-Time Data Warehousing. Informatica Web Seminar". TDWI Research, 2010.

[9] M.T. Serna Encinas, "Entrepôts de données pour l'aide à la décision médicale: conception et expérimentation". Thèse de Doctorat, Université Joseph Fourier, 2005.

[10] P. Scott, "Getting Real - an introduction to real time data warehousing", 2010, http://www.rittmanmead.com

[11] P. Sharma, "Advanced Applications of Data Warehousing Using 3-tier Architecture".DESIDOC Journal of Library & Information Technology, Vol. 29, No. 2, March 2009, pp. 61-66, DESIDOC.

,

.