From Reference Architecture towards Software Architecture for Knowledge Discovery in Database Systems

Lucio Geronimo Valentin[1], Maria Madalena Dias[1] and Roberto C. S. Pacheco[2]

[1] Departamento de Informática, Universidade Estadual de Maringá
Avenida Colombo, 5790, CEP: 87020-900, Maringá, Paraná, Brasil
[2] Programa de Pós-Graduação em Engenharia e Gestão do Conhecimento,
Universidade Federal de Santa Catarina,
B. Trindade, CEP 88040-970, Florianópolis, Santa Catarina, Brasil
lgvalentin@utfpr.edu.br, mmdias@din.uem.br, pacheco@egc.ufsc.br

Abstract. Currently, most of the companies have computational systems to provide support for their operational routines. Many times, those systems are not integrated, generating duplicated and inconsistent information. Such situation makes difficult the search for necessary and trustworthy information for decision-making. Technologies of data warehousing and data mining have appeared to solve that type of problem. Many existing solutions do not enclose those two technologies; some of them are directed to the construction of a data warehouse and others to the application of data mining techniques. There are OLAP tools that do not enclose the activities of preparation of data. This paper presents a reference architecture and a software architecture that define the components necessary for implementation of knowledge discovery systems in database including activities of data warehouse and data mining. Some standards of best practices in projects and in software development were used since the definition until the implementation.

Keywords: Reference Architecture, Software Architecture, Knowledge Discovery in Database, Data Warehouse, and Data Mining.

1 Introduction

With the evolution of computer science and its increasing use in different areas of knowledge, information systems are becoming bigger and more complex. The faster development of hardware at lower prices has been changing methodologies used for software development. There was also, an alteration in the organization of the world-wide markets, which turned the information into a precious item.

In this scene of increasing technological evolution, the search for knowledge became a necessity of governmental institutions and private sectors. However, there is a certain difficulty in the acquisition of information and knowledge because the volume of data accumulated by institutions, during much time, is enormous, and such data are used only in the operational level.

XXXIV Conferencia Latinoamericana de Informática (CLEI 2008), pp. 679-688, 2008.

Several researches have been developed in the area concerning search of information and knowledge in large databases, involving, mainly, Data Warehousing and Data Mining (DM). Data Warehousing technology defines activities for search of large volume of data in heterogeneous operational environments, collecting and organizing the data in a more homogeneous and optimized environment for queries. Data mining is a stage in the process of knowledge discovery in database (KDD). It deals with the search of interesting knowledge that can be hidden in one determined data set.

This paper presents a software architecture specified for KDD systems. In order to facilitate the definition of such architecture, it was necessary to construct a reference model and a reference architecture. The reference model represents the significant relationships between the entities of a environment. The reference architecture defines the common infrastructure for systems and interfaces of the main components that will be enclosed in those systems.

In the next section of the present paper, some related works are presented. Following up in Sections 2, 3 and 4, respectively, the reference model, the reference architecture and software architecture for KDD systems are described. In Section 5, it is shown a comparison of the proposal software architecture with other software architectures with the same application domain. Finally, in Section 6, some conclusions of this study and suggestions for future investigation are presented.

2 Related works

As related works, following ones can be listed: Oracle Warehouse Builder [8]; I-MIN [5]; GridMiner (Brezany et. Al, 2003) [3]; KDB2000 [1].

Oracle Warehouse Builder is a tool that is part of the set of solutions for database of Oracle Company. That environment allows operations such as: the access to databases, definition of transformations, creation of new databases and multidimensional databases operation. Its graphical interface is sufficiently elaborated, with advanced resources for visual edition of the constructed DW model. The environment is integrated with the Oracle database and it provides extensions to load data from other, non Oracle, databases and from files of plain texts.

Gupta [5] presents an architecture for knowledge discovery and management called I-MIN. For the execution of activities of KDD process a language is used. That language is analyzed by a specific compiler present in the architecture proposed. The compiler controls each activity. The I-MIN architecture is divided into small modules.

GridMiner tool was constructed on a framework developed by the same group. That research team includes works about parallel computation in the development of systems for knowledge discovery. The components that comprise that tool are classified in five groups: Fabric, Grid Core, GridMiner Base, GridMiner Core, and GridMiner Workflow. That tool also uses a language that describes the activities that must be executed during KDD process.

KDB2000 integrates, in only one tool, the database access, the preprocessing and transformation of data, algorithms of data mining and tools for visualization. Such integration provides support for the KDD process, thus being possible to observe the

results after executing some of the activities. However, that tool does not provide support for the repetition of the same process executed by the user.

3 Reference Model

The reference model proposed in the present study (Figure 1) enables the integration of the necessary functions for carrying out the construction of a DW, of OLAP tools (On-line Analytical Processing) and of DM in only one system. Such integration provides a high reuse of the system components and, consequently, it contributes with its construction if compared with proposals that isolate the construction of a DW from other parts of a KDD system.

The reference model provided the base for both definitions: definition of the reference architecture and the software architecture.

The reference model (Figure 1) defines three different data sources to be used in OLAP or DM processing: Source Bases, Staging Area and Data Warehouse. Moreover, beyond those databases, the model presents a warehouse to store the results obtained using OLAP or DM.

The module *ET Processing* groups the activities of search and transformation of operational data to store them in the staging area. These activities use metadata that define the structures of the databases and the actions of transformation to be executed.

When data are in the staging area, they can be stored in the DW through the *Load Processing*. Main operations in this processing are summarization, grouping of information and storage of data in multidimensional structures in the DW. This processing also makes use of metadata from definition of database and from data operations to guide yours activities.

The use of metadata to perform definitions and processing provides support to automatic load of data into the DW. Thus, same process can be executed by user repeatedly.

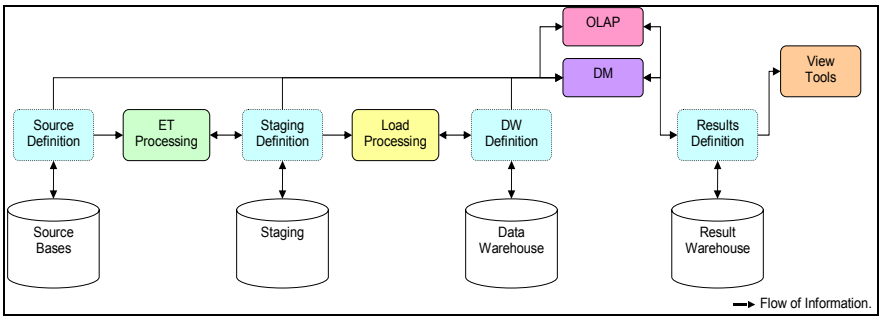

Fig. 1. Reference Model proposed for a KDD system.

OLAP and DM modules are essential for the architecture proposed in the present

study. They integrate OLAP and DM processes in just one environment, reusing operations, such as extraction, transformation and load, and sharing repositories of data. The processing results are stored in the *Result Warehouse*. These results are accessed by the module *View Tools* and can be exported or integrated to another tool for analysis.

The non-functional requirements considered for the reference model were, mainly, matters concerning the software quality, such as security, reliability, availability, performance, usability, maintainability, and portability. These non-functional requirements, not graphically represented in the reference model, influenced directly the definition of the system's architectural components.

In the reference model proposed, can be observed that some functions are shared between the modules of the model, as for example, the routines to access the databases. These functions could remain in a lower layer to be available to the upper layers.

Figure 2 represents layers defined in the architecture. Some managers were defined for each layer to control the creation and the access of layer's components. Components of each layer have access to its manager. Each manager has reference to others managers of lower layer. This makes possible the communication between components from distinct layers and keeps them independents. Thus, the communication between components occurs only through the manager responsible for the layer they belong to.

The organization of the layers offers a low coupling between processes, services and others components. Another advantage observed in that proposal is that the function of each layer is well explicit. Managers supply a well defined or specific interface for each layer, facilitating the learning and the use of the architecture proposed.

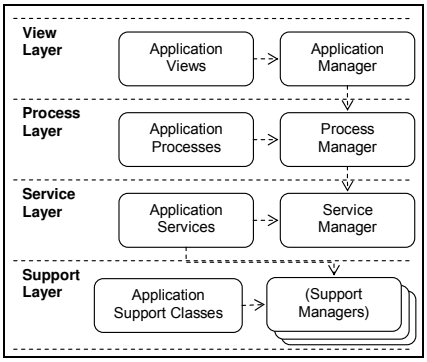

Fig. 2. Layers that compose the proposal Reference Architecture.

The **view layer** is responsible for accommodating the components that show the graphical interfaces (screens) and control the interaction between the user and the system. This layer is formed by two types of components, the application manager and application views. The first one is responsible for supply an access way to lower layers. Application views represent visualization techniques and tools to build the graphical user interface (GUI).

The **process layer** has a manager that knows all the processes instanced by the architecture. Thus, to access any process, the upper layer must ask the manager for instance of the process desired. Two characteristics must be pointed out here: 1) processes are authenticated by the layer and are available only for users whose access is allowed; 2) one instance of the process is created for each user.

The **service layer** has a manager that knows all the services that are instanced by the architecture. Also, this layer provides an interface that allows the upper layer to execute an available service. This layer is not authenticated and all the users share the same instance of a determined service.

The **support layer** was defined to organize system's functions which give support for upper layers. Some services, which are mapped in the upper layer, need a more specific management of specific technologies to perform their functions. In this case, a manager can be implemented on this layer to support the manipulation of such technologies and the service can be linked with this specific manager.

Following the architectural style, and having the layers well defined for the work, the requirements from KDD system have been divided into interface, process, service and support requirement, respectively. The task to define, exactly, what must be assigned to each of the layers is not a easy way, mainly for people having the first contacts with such architecture. About the question "*Where must I implement this or that?*", the documentation of the architecture is an important assistant.

To document the reference architecture, was necessary an adaptation of the methodology presented by [7]. Such methodology defines that an architecture can be documented in terms of views. Each view points out either static or dynamic aspects of the architecture. But, not all the views are necessary for all the projects.

To illustrate the developed reference architecture, the view of module definition and its relationship are shown in Figure 3.

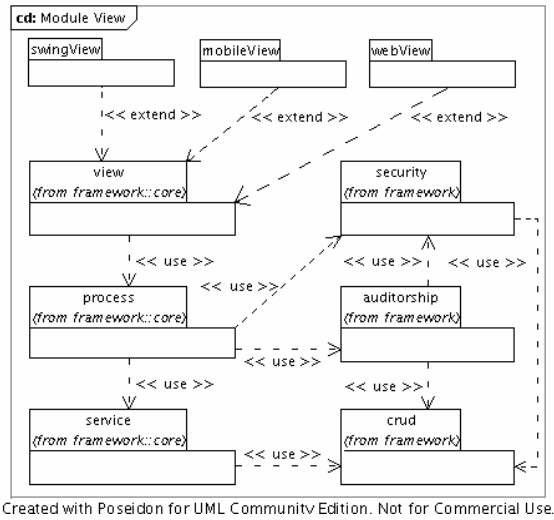

Fig. 3. Modules view that shows the packages of reference architecture.

It is important to point out in the Figure 3 that the *view package* was extended by others packages that implement different visualization technologies, turning the business processes completely independent from interface technologies.

The **view module** provides access for several controllers of interface that manipulate business processes.

The **process module** is authenticated. Thus, it uses the *security* and *auditorship* modules. Information about execution of processes is registered into *auditorship*.

The manager of the service layer is situated in the **service module**. In this module, the services are executed inside of a transactional environment. Services can access and manipulate metadata (business entity/ persistent data) by using the *crud* module.

In the **security module** are included the security entities, users, groups and rights. All services and processes for user's authentication, verification and definition of access rights are defined in this module. The *crud* module is used so that the security module can access the entities.

On the other hand, the **auditorship module** provides some services to register all activities executed in processes and operations (creation, edition and removal) performed in entities.

The **crud module** comprises a manager of entities, services and processes for persistence of objects. However, the main function of that module is to provide an abstraction of business entity and data persistence. Such abstraction consists of joining the object instances with a metadata structure that contains additional information about the entity. Thus, when an entity is recovered from the object repository, it comes with its basic data, and also, with a set of additional information describing the entity and its properties. The additional information set is useful for validation and exhibition of the entity data.

4 Software Architecture

After constructing a reference model and a reference architecture, the next step was to define the software architecture. Both, the view of modules (Figure 4) and a case diagram of KDD system (Figure 5) were used to represent the software architecture and your requisites.

With exception of the databases, a module for each element presented in the reference model was defined. Those modules contain the basic functions for accomplishing the activities from a KDD system. The present investigation enables each system development team to perform a detailed definition of the resources necessary for the execution of its own KDD process.

A diagram, regarding use cases, was constructed to assist both, the learning of functional requirements and the conception of software architecture. That type of diagram was defined by UML and it is used so that the Unified Process can capture the requirements [6]. Figure 5 represents the main use cases that were identified in the reference model. Those use cases have guided the development of the prototype for validation of the architecture.

Following the reference architecture, each module defines its own views, processes, services and support components. This is a cohesive proposal.

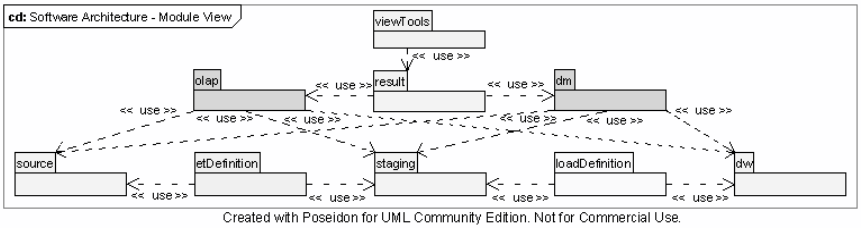

Fig. 4. Modules view of the Software Architecture of KDD system.

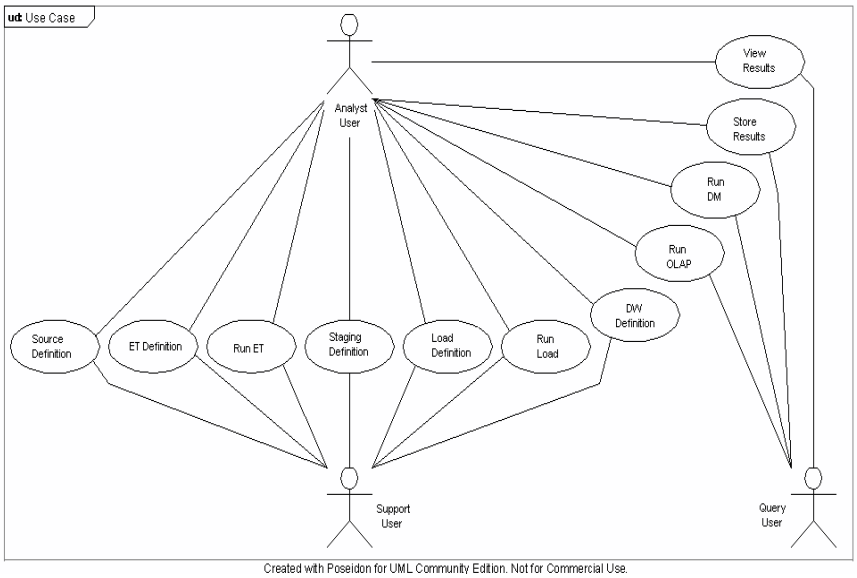

Fig. 5. Use cases of KDD system presented in the reference model.

The software architecture is defined extending the functions found in the reference architecture.

The software architecture is a specific architecture for KDD domain. As the reference architecture solves the majority of the structural problems, the definition of software architecture is facilitated and its components are abstracted from the concepts defined by the reference architecture. Thus, the software architecture proposed consists of the accomplishment of the identified use cases, represented in Figure 5.

Following, the description of the use case *Source Definition* is presented to illustrate how a function can be carried out having as basis the components predefined in the reference architecture.

4.1 Source Definition

To exemplify, Figure 6 shows the entities, the services, the processes and the visualizations that have been obtained from the use case known as *source definition*. The *source bases* structures are defined by the legacy systems, which are responsible for those bases. However, in that proposal, it was established the necessity of execution of a selection process that defines what tables and data item may be accessed by KDD system, in order to organize the manipulation of the *source data*. The selection of tables and items of data is stored in internal entities of the system.

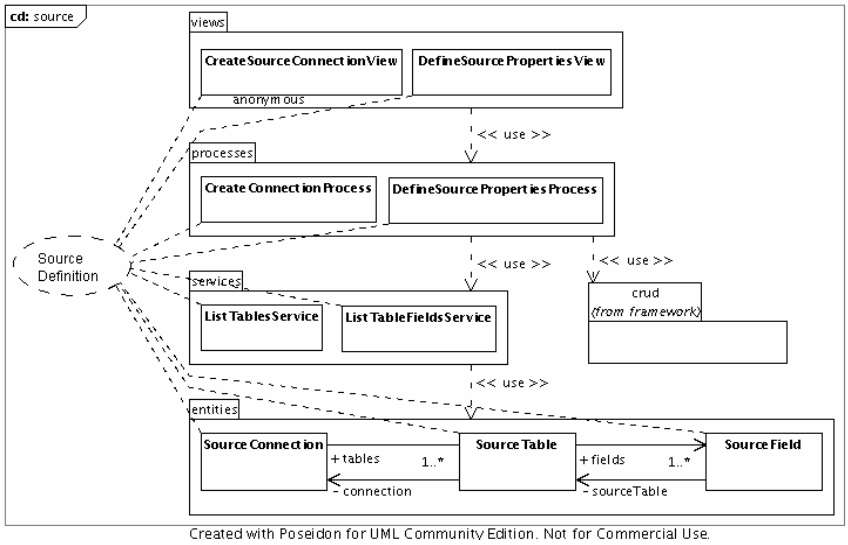

Fig. 6. Views, processes, services and entities defined by use case SourceDefinition.

The *SourceConnection* entity stores the information of a connection together with the source databases. Each connection is related with many *SourceTable* entities that describe the source table structure. Each *SourceTable* contains a set of *SourceField* entities, which describe items of data from the source tables.

Two processes were created to manipulate the entities represented in Figure 6 and to accomplish the source definition activities: *CreateConnectionProcess* and *DefineSourceProperties* Process.

The process named *CreateConnectionProcess* controls the creation of a new connection. It requests, validates and persists necessary information for a database connection.

The process named *DefineSourcePropertiesProcess* controls the definition of both, tables and items of data from the source databases that will be accessible or available for KDD system.

Services from the *crud* module are used to list the connections registered within the register of KDD system. Services such as *ListTablesService* and *ListTableFieldsService* are used to list tables from a connection and items of data from a table, respectively.

4.2 Checking Architecture Proposed and Related Works differences

The main characteristics of works related to the area of knowledge discovery in database were considered in the present architecture model.

The differential of this work is the proposal of a reference architecture with basics modules for composition of a KDD system. These modules are supported by the software architecture proposed and it can be extended in accordance with necessity of the development team.

The reference architecture proposed serves as a guideline, a way to be followed in the development of KDD system. Many architectural problems are decided by the reference architecture. This makes possible which development teams have focus in the problems of its KDD domain.

Table 1 presents a comparison between the architecture proposed and the related works.

Table 1. Best characteristics of architecture proposed and related works

Characteristics	Oracle (2005)	Gupta (2004)	GridMiner (2003)	KDB200 (2006)	Architecture proposal
Descriptive language		X	X		
Interactive activities				X	X
Software architecture reusable	[1]	[2]	[3]	[4]	X[5]
Sources, targets and processes metadata	X	X			X
DW building	X		X		X
OLAP activities		X	X	X	X
DM activities		X	X	X	X

The numbers presented in the "software architecture reusable" characteristic of Table 1 represent: 1) Oracle proprietor architecture; 2) Architecture defined only for the author's environment; 3) Set of components that can be reused; 4) Architecture defined only for the author's KDB2000 application; 5) Architecture defined so that development teams can extend it, by using the reference architecture previously developed.

5 Conclusions

Following the construction standard of architecture presented by Bass [2], a reference model was defined and a reference architecture was developed, in order to propose a software architecture for KDD systems.

The reference model defined proposes the integration of main functions found in the construction of a DW, in OLAP and data mining tools in an unique system.

The software architecture proposed was based on the reference model, and on use cases identified for a KDD system, being validated through a prototype. The reference architecture, specified and implemented, facilitated the development of this prototype.

The solutions proposed in the development of the reference architecture are not

exclusively bound to KDD system domain, once they are more generic solutions. Problems that demand a specific solution for KDD domain are solved in the software architecture.

With the use of architecture views, presented by Merson [7], static and dynamic aspects of architecture can be understood. The views facilitate the learning and allow presenting both, the general organization of an architecture, and its internal functions.

The reference architecture proposed is sufficiently independent from KDD domain, thus making it reusable in other domains. Matters regarding platform, data, and interface portability are basic for any software that intends to integrate corporative environments.

The *crud* module suggested in this architecture is an advanced mechanism to persist and control the entities of internal data of a KDD system. Such entities are the system metadata. They describe the organization of KDD process, indicate the connections that are used, the number of tables and items of data are available; the transformation and load functions are used. Those metadata entities are the technician metadata defined by David [4].

As it could be seen, the definition of the software architecture's components was based on concepts defined in the reference architecture, having as main focus the application domain. Questions related to the communication between components, security and other matters remain in the definition of reference architecture, thus making possible a better definition of both, the application domain and the system functions in the software architecture.

The software architecture proposed and presented in this paper are not restricted to the specification of functions described by the use case. That architecture can be used as an initial guide to solve trivial problems.

References

1. Appice, A., Ceci, M., Malerba, D., *KDB2000: An integrated knowledge discovery tool*. Dipartimento di Informatica,University of Bari, Italy, (2007). Retrieved November 30, 2007, from http://www.di.uniba.it/~malerba/software/kdb2000
2. Bass, L., Clements, P., Kazman, R., *Software Architecture in Practice. Second Edition.* Boston, MA: Addison-Wesley, 2003.
3. Brezany, P., Hofer, J., Tjoa, A., Wöhrer, A., *Gridminer: An Infrastructure for Data Mining on Computational Grids*. APAC'03 Conference, Gold Coast, Australia, October 2003
4. David, M. *Building and managing the metadata Repository: A Full Lifecycle Guide*, Paperback, 2001.
5. Gupta, S. K.; Bhatnagar, V.; Wasan, S.K., *Architecture for knowledge discovery and knowledge management*. Knowledge and Information Systems, 2004.
6. Jacobson, I., Booch, G., *The Unified Software Development Process*. Addison Wesley, 1998.
7. Merson, P., *How to Represent the Architecture of Your Enterprise Application Using UML 2.0 and More*. (2007). Retrieved November 30, 2007, from http://developers.sun.com/learning/javaoneonline/2006/coreenterprise/TS-4619.pdf.
8. *Oracle Warehouse Builder Documentation*. (2007). Retrieved November 30, 2007, from http://www.oracle.com/technology/documentation/warehouse.html.