Physical Modeling of Data Warehouses Using UML Component and Deployment Diagrams: Design and Implementation Issues

Sergio Luján-Mora, Juan Trujillo
Department of Software and Computing Systems
University of Alicante
Spain
{slujan,jtrujillo}@dlsi.ua.es

Abstract

Several approaches have been proposed to model different aspects of a Data Warehouse (DW) during recent years, such as the modeling of a DW at the conceptual and logical level, the design of the ETL (Extraction, Transformation, Loading) processes, the derivation of the DW models from the enterprise data models, and customization of a DW schema. At the end of the design, a DW has to be deployed in a database environment, requiring many decisions of a physical nature. However, few efforts have been dedicated to the modeling of the physical design of a DW from the early stages of a DW project. In this article, we argue that some physical decision can be taken from gathering main user requirements. In this paper, we present physical modeling techniques for DWs using the *component diagrams* and *deployment diagrams* of the Unified Modeling Language (UML). Our approach allows the designer to anticipate important physical design decisions that may reduce the overall development time of a DW such as replicating dimension tables, vertical and horizontal partitioning of a fact table, and the use of particular servers for certain ETL processes. Moreover, our approach allows the designer to cover all main design phases of DWs, from the conceptual modeling phase to the final implementation. To illustrate our techniques, we show a case study that is implemented on top of a commercial DW management server.

Keywords: data warehouse, configuration, deployment, component, UML, physical design implementation

1 Introduction

Data warehouses (DW) provide organization with historical information to support a decision. It is widely accepted that these systems are based on multidimensional (MD) modeling. Thus, research on the design of a DW has been mainly addressed from the conceptual and logical point of view through multidimensional (MD) data models (Blaschka, Sapia, Höfling, & Dinter, 1998; Abelló, Samos, & Saltor, 2001). However, to the best of our knowledge, there are not any standard methods or models that allows us to model all aspects of a DW. Moreover, as most of the research efforts in designing and modeling DWs have been focused on the development of MD data models, the attention to the physical design of DWs from the early stages of a DW project has been very little. Nevertheless, the physical design of a DW is of a vital importance and highly influences the overall performance of the DW (Nicola & Rizvi, 2003) and the following maintenance; even more, a well-structured physical design policy can provide the perfect roadmap for implementing the whole warehouse architecture (Triantafillakis, Kanellis, & Martakos, 2004).

Although in some companies the same employee may take on both the role of DW designer and DW administrator, other organizations may have separate people working on each task.

Regardless of the situation, modeling the storage of the data and how it will be deployed across different components (servers, drives, etc.) helps implementing and maintaining a DW. In traditional software products or transactional databases, physical design or implementation issues are not considered until the latest stages of a software project. Then, if the final product does not satisfy user requirements, designers do a feedback taking into consideration (or at least bearing in mind) some final implementation issues.

Nevertheless, due to the specific characteristics of DWs, we can address several decisions regarding the physical design of a DW from the early stages of a DW project with no need to leave them until the final implementation stage. DWs, mainly built for analytical reasons, are queried by final users trying to analyze historical data on which they can base their strategy decisions. Thus, the performance measure for DWs is the amount of queries that can be executed instead of the amount of processes or transactions that it supports. Moreover, the kinds of queries on DWs are demonstrated to be much more complex than the queries normally posed in transactional databases (Kimball, 1996; Poe, Klauer, & Brobst, 1998). Therefore, poor performance of queries has a worse impact in DWs than in transactional databases. Furthermore, the set of OLAP (On-Line Analytical Processing) operations that users can execute with OLAP tools on DWs depends so much on the design of the DW, i.e., on the multidimensional model underneath (Sapia, 1999; Trujillo, Palomar, Gómez, & Song, 2001).

Based on our experience in real world DW projects, physical storage and query performance issues can be discussed in the early stages of the project. The reason is that in DW projects, final users, analysts and business managers, DW designers, and database administrators participate, at least, in first meetings. Therefore, we believe that some decisions on the physical design of DWs can be made in the beginning. Some examples of these decision are as follows: (i) the size and the speed of the hard disk needed to deal with the fact table and the corresponding views, (ii) a coherent partitioning of both fact and dimension tables based on data and user requirements, (iii) the estimation of the workload needed and the time boundaries to accomplish it. Based on our experience, we believe that making these decisions in the early stages of a DW project will reduce the total development time of the DW.

At this point, we must point out that we are not suggesting that the conceptual modeling of a DW take into account physical issues. Instead, we advocate that the physical aspects and following implementation details from the conceptual modeling of the DW from the early stages of a DW project will benefit the implementation.

In previous works (Luján-Mora & Trujillo, 2003, 2004a), we have proposed a DW development method, based on the Unified Modeling Language (UML) (Object Management Group (OMG), 2003) and the Unified Process (UP) (Jacobson, Booch, & Rumbaugh, 1999), to properly design all aspects of a DW. So far, we have dealt with the modeling of different aspects of a DW by using the UML (Object Management Group (OMG), 2003): MD modeling (Trujillo et al., 2001; Luján-Mora, Trujillo, & Song, 2002a, 2002b), modeling of the ETL processes (Trujillo & Luján-Mora, 2003), and modeling data mappings between data sources and targets (Luján-Mora, Vassiliadis, & Trujillo, 2004). In this paper, we complement all of these previous works with a proposal to accomplish the physical design of DWs from the early stages of a DW project. To accomplish these goals, we propose the use of the *component diagrams* and *deployment diagrams* of UML. Both *component* and *deployment* diagrams must be defined at the same time by DW designers and DW administrators who will be in charge of the subsequent implementation and maintenance. This is mainly due to the fact that while the former know how to design and build a DW, the latter have a better knowledge in the corresponding implementation and the real hardware and software needs for the correct functioning of the DW.

The modeling of the physical design of a DW from the early stages of a DW project with our approach provides us many advantages:

- We deal with important aspects of the implementation before we start with the implementation process, and therefore, we can reduce the total development time of the DW. This is mainly due to the fact that, after the conceptual modeling has been accomplished, we can have enough information to make some decisions regarding the implementation of the DW structures such as replicating dimension tables or designing the vertical and horizontal partitioning of a fact table.

- We have rapid feedback if there is a problem with the DW implementation as we can easily track a problem to find out its main reasons.
- It facilitates communication between all people involved in the design of a DW since all of them use the same notation (based on UML) for modeling different aspects of a DW. Moreover, making sure that the crucial concepts mean the same to all groups and are not used in different ways is critical. In this way, our approach helps achieve a coherent and consistent documentation during the DW development life cycle.
- It helps us choose both hardware and software on which we intend to implement the DW. This also allows us to compare and evaluate different configurations based on user requirements.
- It allows us to verify that all different parts of the DW (fact and dimension tables, ETL processes, OLAP tools, etc.) perfectly fit together.

A short version of this paper was presented previously (Luján-Mora & Trujillo, 2004b). In this long version, we have added new stereotypes to consider more physical decisions such as *views* and *indices*. Furthermore, we have also included details on the implementation of a case study on a commercial database management server from our *component* and *deployment diagrams*, showing the benefit of our approach.

The rest of the paper is organized as follows. In Section 2, we briefly comment other works that have dealt with the conceptual, logical and physical design and/or deployment of a DW. In Section 3, we briefly introduce our overall method to design all aspects of a DW. In Section 4, we present main issues that can be specified by using both component and deployment diagrams of UML. In Section 5, we describe our approach for using both component and deployment diagrams for the physical design of DWs. In Section 6, we provide a deep detail on how to use our component and deployment diagrams to implement a DW on a commercial database management server. Finally, in Section 7, we present our conclusions and main future work.

2 Related Work

As this paper focuses in the design of DWs, and more specifically, the physical design of DWs, the related work is organized into three subsections, about multidimensional modeling, physical design and implementation of DWs and UML extensibility mechanisms.

2.1 Multidimensional Modeling

Several multidimensional (MD) data models have been proposed for DWs. Some of them fall into the logical level (such as the well-known star schema by Kimball (1996)). Others may be considered as formal models as they provide a formalism to consider main MD properties. A review of the most relevant logical and formal models can be found in Blaschka et al. (1998).

In this sub-section, we will only make brief reference to the most relevant models that we consider "pure" conceptual MD models as this paper we focus on the physical design of DWs from the early stages of a DW project. These models provide a high level of abstraction for the main MD modeling properties (e.g., facts, dimensions, classification hierarchies defined along dimensions, the additivity of measures, etc.) and are independent from implementation issues. One interesting feature provided by these models is that they provide a set of graphical notations (such as the classical and well-known EER model) that facilitates their use and reading. These are as follows: *The Dimensional-Fact (DF) Model* by Golfarelli, Maio, and Rizzi (1998), *The Multidimensional/ER (M/ER) Model* by Sapia, Blaschka, Höfling, and Dinter (1998), *The starER Model* by Tryfona, Busborg, and Christiansen (1999), the Model proposed by Hüsemann, Lechtenbörger, and Vossen (2000), and *The Yet Another Multidimensional Model (YAM^2)* by Abelló, Samos, and Saltor (2002).

However, none of these approaches for MD modeling considers the design of physical aspects of DWs as an important issue of their modeling, and therefore, they do not solve the problem of physical modeling from the early stages of a DW project.

2.2 Physical Design and Implementation Issues of Data Warehouses

Both the research community and companies have devoted few efforts to the physical design of DWs from the early stages of a DW project, and incorporate it within a global method that allows designing all main aspects of DWs. In this subsection, we are not presenting research on physical issues of DWs such as new algorithms for defining and managing indices, view materialization, query processing or performance as these and other physical aspects of DWs are out of the scope of this paper; instead, we will concentrate on the modeling of the physical design of DWs from the first stages of a DW project.

Ralph Kimball *et al* study the lifecycle of a DW and propose a method for the design, development and deployment of a DW (Kimball, Reeves, Ross, & Thornthwaite, 1998). They discuss the planning of the deployment of a DW and they recommend documenting all different deployment strategies. However, they do not provide a standard technique for the formal modeling of the deployment of a DW.

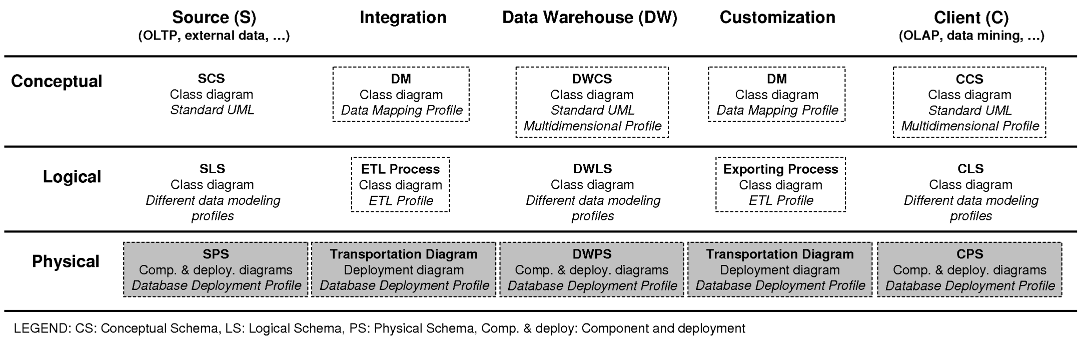
Poe et al. (1998) address the design of a DW from conceptual modeling to implementation. They propose the use of non-standard diagrams to represent the physical architecture of a DW: on one hand, to represent data integration processes and, on the other hand, to represent the relationship between the *enterprise data warehouse* and the different *data marts* that are populated from it. Nevertheless, these diagrams represent the architecture of the DW from a high level without providing different levels of detail of the subsequent implementation of the DW.

Giovinazzo (2000) discusses several aspects of a DW implementation. Although in this book, other aspects of a DW implementation such as the parallelism, the partitioning of data in a RAID (*Redundant Array of Inexpensive Disk*) system or the use of a distributed database are addressed, authors do not provide a formal or standard technique to model all these aspects.

Finally, Rizzi (2003) states that one of the current open problems regarding DWs is the lack of a formal documentation that covers all design phases and provides multiple levels of abstraction (low level for designers and people devoted to the corresponding implementation, and high level for final users). The author argues that this documentation is basic for the maintenance and the ulterior extension of the DW. In this work, three different detailed levels for DWs are proposed: *data warehouse level*, *data mart level* and *fact level*. At the first level, the use of the deployment diagrams of UML are proposed to document a DW architecture from a high level of detail. However, these diagrams are not integrated at all with the rest of techniques, models and/or methods used in the design of other aspects of the DW.

On the other hand, Naiburg and Maksimchuk (2001) have studied the use of UML for the design of databases. Their work is structured around the database design process, therefore, it contains a chapter devoted to database deployment. In this book it is stated that, from the database designers' point of view, in a real database development project, "the biggest benefit in using the UML is the ability to model the tablespaces and quickly understand what tablespaces exist and how tables are partitioned across those tablespaces". On the other hand, using UML for designing databases has the advantage that a different UML diagram can be used (e.g., package diagram, class diagram, component diagram, and deployment diagram) depending on the particular aspect modeled, and then, many transformations between these diagrams have been widely and recently proposed (Whittle, 2000; Selonen, Koskimies, & Sakkinen, 2003).

Therefore, we argue that there is a still a need for providing a standard technique that allows modeling the physical design of a DW from the early stages of a DW project. Another important issue is that this technique is integrated in an overall approach that allows coverage of other aspects of the DW design such the conceptual or logical design of the DW or the modeling of ETL processes.



	Source (S) (OLTP, external data, ...)	Integration	Data Warehouse (DW)	Customization	Client (C) (OLAP, data mining, ...)
Conceptual	**SCS** Class diagram *Standard UML*	**DM** Class diagram *Data Mapping Profile*	**DWCS** Class diagram *Standard UML* *Multidimensional Profile*	**DM** Class diagram *Data Mapping Profile*	**CCS** Class diagram *Standard UML* *Multidimensional Profile*
Logical	**SLS** Class diagram *Different data modeling profiles*	**ETL Process** Class diagram *ETL Profile*	**DWLS** Class diagram *Different data modeling profiles*	**Exporting Process** Class diagram *ETL Profile*	**CLS** Class diagram *Different data modeling profiles*
Physical	**SPS** Comp. & deploy. diagrams *Database Deployment Profile*	**Transportation Diagram** Deployment diagram *Database Deployment Profile*	**DWPS** Comp. & deploy. diagrams *Database Deployment Profile*	**Transportation Diagram** Deployment diagram *Database Deployment Profile*	**CPS** Comp. & deploy. diagrams *Database Deployment Profile*

LEGEND: CS: Conceptual Schema, LS: Logical Schema, PS: Physical Schema, Comp. & deploy: Component and deployment

Figure 1: Data warehouse design framework

2.3 UML Extensibility Mechanism

The UML Extensibility Mechanism package is the subpackage from the UML metamodel that specifies how specific UML model elements are customized and extended with new semantics by using stereotypes, tagged values, and constraints. A coherent set of such extensions, defined for specific purposes, constitutes a UML *profile*. For example, the UML 1.5 (Object Management Group (OMG), 2003) includes a standard profile for modeling software development processes and another one for business modeling.

A *stereotype* is a model element that defines additional values (based on tagged values), additional constraints, and optionally a new graphical representation (an icon): a stereotype allows us to attach a new semantic meaning to a model element. A stereotype is either represented as a string between a pair of guillemots (≪ ≫) or rendered as a new icon.

A *tagged value* specifies a new kind of property that may be attached to a model element. A tagged value is rendered as a string enclosed by brackets ([ ]) and placed below the name of another element.

A *constraint* can be attached to any model element to refine its semantics; Warmer and Kleppe (1998) state, "A constraint is a restriction on one or more values of (part of) an object-oriented model or system". In the UML, a constraint is rendered as a string between a pair of braces ({ }) and placed near the associated model element. A constraint on a stereotype is interpreted as a constraint on all types on which the stereotype is applied. A constraint can be defined by means of an informal explanation or by means of OCL (Warmer & Kleppe, 1998; Object Management Group (OMG), 2003) expressions. The OCL is a declarative language that allows software developers to write constraints over object models.

3 Data Warehouse Design Framework

The architecture of a DW is usually depicted as various layers of data in which data from one layer is derived from data of the previous layer (Jarke, Lenzerini, Vassiliou, & Vassiliadis, 2003). In a previous work (Luján-Mora & Trujillo, 2004a), we have presented a DW development method, based on UML (Object Management Group (OMG), 2003) and the UP (Jacobson et al., 1999), that addresses the design and development of both the DW back-end and front-end. In our approach, we consider that the development of a DW can be structured into an integrated framework with five stages and three levels that define different diagrams for the DW model, as shown in Figure 1 and summarized next:

- **Stages**: we distinguish five stages in the definition of a DW:
  - Source, that defines the data sources of the DW, such as OLTP systems, external data sources (syndicated data, census data), etc.
  - Integration, that defines the mapping between the data sources and the DW.

    - Data Warehouse, that defines the structure of the DW.
    - Customization, that defines the mapping between the DW and the clients' structures.
    - Client, that defines special structures that are used by the clients to access the DW, such as data marts (DM) or OLAP applications.

- **Levels**: each stage can be analyzed at three different levels or perspectives:
    - Conceptual: it defines the DW from a conceptual point of view.
    - Logical: it addresses logical aspects of the DW design, such as the definition of the ETL processes.
    - Physical: it defines physical aspects of the DW, such as the storage of the logical structures in different disks, or the configuration of the database servers that support the DW.

- **Diagrams**: each stage or level requires different modeling formalisms. Therefore, our approach is composed of 15 diagrams, but the DW designer does not need to define all the diagrams in each DW project: for example, if there is a straightforward mapping between the Source Conceptual Schema (SCS) and the Data Warehouse Conceptual Schema (DWCS), the designer may not need to define the corresponding Data Mapping (DM). In our approach, we use UML (Object Management Group (OMG), 2003) as the modeling language, because it provides enough expressiveness power to address all the diagrams. As UML is a general modeling language, we can use UML extension mechanisms (stereotypes, tag definitions, and constraints) to adapt UML to specific domains. A stereotype is a UML modeling element that extends the UML metamodel in a controlled way, i.e., a stereotype is a specialized version of a standard UML element; a tag definition allows additional information about a standard UML element to be specified; and a constraint is a rule that limits the behavior of a UML element. Figure 1 contains the following information for each diagram:
    - Name (**in bold face**): the name we have coined for this diagram.
    - UML diagram: the UML diagram we use to model this DW diagram. Currently, we use class, deployment, and component diagrams.
    - Profile (*in italic font*): the dashed boxes show the diagrams where we propose a new profile[1]; in the other boxes, we use a standard UML diagram or a profile from other authors.

The different diagrams of the same DW are not independent but overlapping: they depend on each other in many ways, therefore, they can not be created in any order[2]. For example, changes in one diagram may imply changes in another, and a large portion of one diagram may be created on the basis of another diagram. For example, the Data Mapping (DM) is created by importing elements from the Source Conceptual Schema (SCS) and the Data Warehouse Conceptual Schema (DWCS). Moreover, our approach is flexible in the sense that the DW designer does not need to define all the diagrams, but he or she can use what is needed when it is needed and can continue moving forward as necessary.

In previous works, we have presented some of the diagrams and the corresponding profiles shown in white dashed boxes in Figure 1: *Multidimensional Profile* (Luján-Mora et al., 2002a, 2002b) for the DWCS and the Client Conceptual Schema (CCS), the *ETL Profile* (Trujillo & Luján-Mora, 2003) for the ETL Process and the Exporting Process, and the *Data Mapping Profile* (Luján-Mora et al., 2004) for the DM between the SCS and the DWCS, and between the DWCS and the CCS. Finally, in light gray dashed boxes, we show the profile we present in this paper, the *Database Deployment Profile*, for modeling a DW at a physical level.

Figure 2 shows a symbolic diagram to summarize our approach and the relationships between the different diagrams (DWCS, DWLS, and DWPS):

[1]A profile is an extension to the UML that uses stereotypes, tagged values, and constraints to extend the UML for specialized purposes.

[2]Due to the lack of space, we do not show the order we propose in our DW design method.

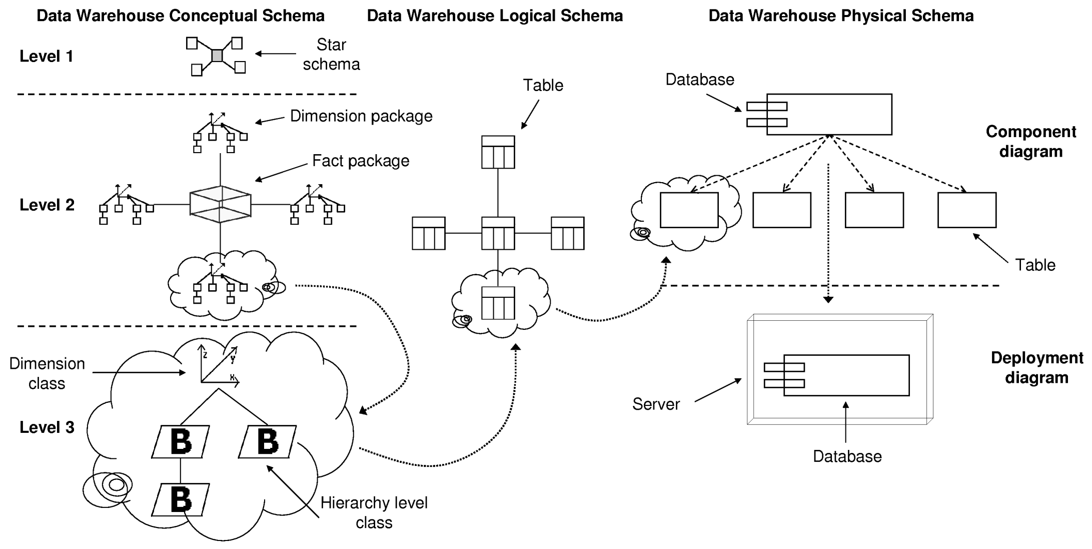


Figure 2: From the conceptual to the physical level

- On the left hand side of this figure we have represented the DWCS, which is structured into three levels: Level 1 or *Model definition*, Level 2 or *Star schema definition*, and Level 3 or *Dimension/fact definition*. The different elements drawn in this diagram are stereotyped packages and classes[3] that represent MD concepts.
- From the DWCS, we develop[4] the logical model (DWLS, represented in the middle of Figure 2) according to different options, such as ROLAP[5] (*Relational OLAP*) or MOLAP[6] (*Multidimensional OLAP*). In this example, we have chosen a ROLAP representation and each element corresponds to a table in the relational model.
- Finally, from the DWLS we derive the DWPS, which is represented on the right hand side of Figure 2. The DWPS shows the physical aspects of the implementation of the DW. This diagram is divided up into two parts: the component diagram, which shows the configuration of the logical structures used to store the DW, and the deployment diagram, which specifies different aspects relative to the hardware and software configuration.

Figure 2 shows how our approach allows the designer to trace the design of an element from the conceptual to the physical level. For example, in this figure, we have drawn a cloud around different elements that represent the same entity in different diagrams.

In the following section, we summarize the basic concepts about the UML component and deployment diagrams that we apply for the physical design of DWs in Section 5

4 UML Component and Deployment Diagrams

According to the UML Specification (Object Management Group (OMG), 2003), "Implementation diagrams show aspects of physical implementation, including the structure of components and the run-time deployment system. They come in two forms: 1) component diagrams show the structure of components, including the classifiers that specify them and the artifacts that

[3]An icon, a new graphical representation, can be associated to a stereotype in UML.

[4]The transformation process from the DWCS to the DWLS is outside the scope of this paper.

[5]ROLAP is a storage model that uses tables in a relational database to store multidimensional data.

[6]MOLAP is a storage mode that uses a proprietary multidimensional structure to store multidimensional data.

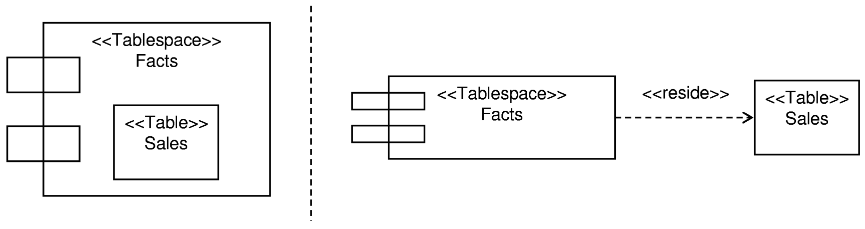


Figure 3: Different component representations in a component diagram

implement them; and 2) deployment diagrams show the structure of the nodes on which the components are deployed".

In Section 4.1 we summarize the main concepts about the UML component diagram, whereas in Section 4.2 we introduce the deployment diagram.

4.1 Component Diagram

The UML Specifion says that "A component represents a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces". Components represent physical issues such as Enterprise JavaBeans, ActiveX components or configuration files. A component is typically specified by one or more classifiers (e.g., classes and interfaces) that reside on the component. A subset of these classifiers explicitly define the component's external interfaces. Moreover, a component can also contain other components. However, a component does not have its own features (attributes and operations).

On the other hand, a component diagram is a graph of components connected by dependency relationships that shows how classifiers are assigned to components and how the components depend on each other. In a component diagram (Figure 3), a component is represented using a rectangular box, with two rectangles protruding from the left side.

Figure 3 shows the two different representations of a component and the classifiers it contains:

- On the left hand side of the figure, the class (Sales) that resides on the component (Facts) is shown as nested inside the component (this indicates residence and not ownership).
- On the right hand side of the figure, the class is connected to the component by a ≪reside≫ dependency.

In these examples, both the component and the class are stereotyped: the component is adorned with the ≪Tablespace≫ stereotype and the class with the ≪Table≫ stereotype; these stereotypes are defined by Naiburg and Maksimchuk (2001).

4.2 Deployment Diagram

According to the UML Specification, "Deployment diagrams show the configuration of run-time processing elements and the software components, processes, and objects that execute on them". A deployment diagram is a graph of nodes connected by communication associations. A deployment model is a collection of one or more deployment diagrams with their associated documentation.

In a deployment diagram, a node represents a piece of hardware (e.g., a computer, a device, an interface) or a software artifact (e.g., web server, database) in the system, and it is represented by a three-dimensional cube. A node may contain components that indicate that the components run or execute on the node.

An association of nodes, which is drawn as a solid line between two nodes, indicates a line of communication between the nodes; the association may have a stereotype to indicate

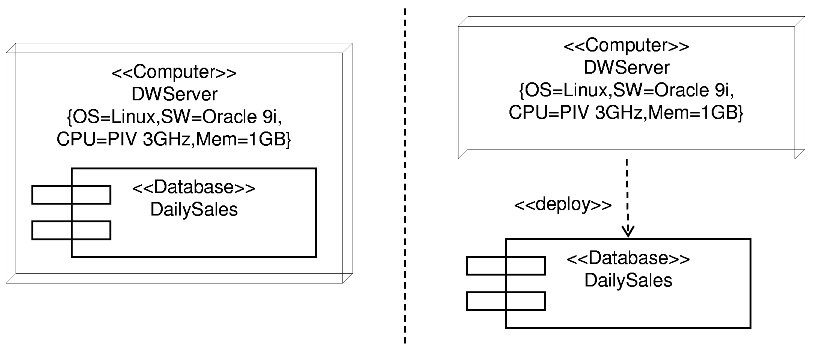


Figure 4: Different node representations in a deployment diagram

the nature of the communication path (e.g., the kind of channel, communication protocol or network).

There are two forms of deployment diagram:

1. The descriptor form: it contains types of nodes and components. This form is used as a first-cut deployment diagram during the design of a system, when there is not a complete decision about the final hardware architecture.

2. The instance form: it contains specific and identifiable nodes and components. This form is used to show the actual deployment of a system at a particular site, therefore it is normally used in the last steps of the implementation activity, when the details of the deployment site are known.

According to Ambler (2002), a deployment diagram is normally used to:

- Explore the issues involved with installing your system into production.
- Explore the dependencies that your system has with other systems that are currently in, or planned for, your production environment.
- Depict a major deployment configuration of a business application.
- Design the hardware and software configuration of an embedded system.
- Depict the hardware/network infrastructure of an organization.

UML deployment diagrams normally make extensive use of visual stereotypes because it is easy to read the diagrams at a glance. Unfortunately, there are no standard palettes of visual stereotypes for UML deployment diagrams.

As it is suggested by Ambler (2002), each node in a deployment diagram may have tens if not hundreds of software components deployed to it; the goal is not to depict all of them, but to depict those components that are vital to the understanding of the system.

Figure 4 shows two different representations of a node and the components it contains:

1. On the left hand side of the figure, the component (DailySales) that is deployed on the node (DWServer) is shown as nested inside the node.

2. On the right hand side of the figure, the component is connected to the node by a ≪deploy≫ dependency.

In this example, both the node and the component are stereotyped: the node with the ≪Computer≫ stereotype and the component with the ≪Database≫ stereotype. Moreover,

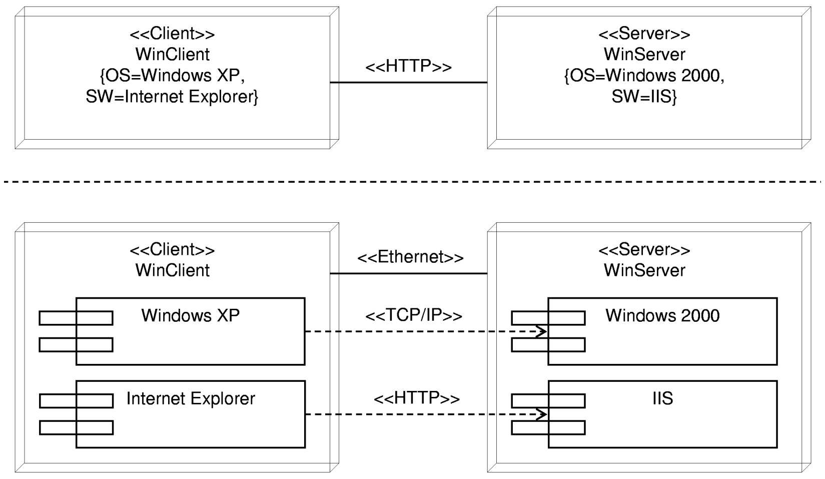


Figure 5: Different levels of detail in a deployment diagram

the node DWServer contains a set of tagged values (OS, SW, CPU, and Mem) that allow the designer to describe the particular characteristics of the node.

A deployment diagram can be specified at different levels of detail. For example, Figure 5 shows two versions of the same deployment diagram. At the top of Figure 5, the software deployed in the nodes is specified by means of tagged values. Moreover, the association between the nodes is only adorned with the ≪HTTP≫ stereotype (*HyperText Transfer Protocol*), although different protocols can be used in the communication. At the bottom of Figure 5, the software deployed in the nodes is depicted as components and different stereotyped dependencies (≪TCP/IP≫ and ≪HTTP≫) indicate how one component uses the services of another component. However, there are more display possibilities; for example, the designer can omit the tagged values in the diagram and capture them only in the supported documentation.

5 Data Warehouse Physical Design

In Section 3, we have briefly described our design method for DWs. Within this method, we use the component and deployment diagrams to model the physical level of DWs. To achieve this goal, we propose the following five diagrams, which correspond to the five stages presented in Section 3:

- Source Physical Schema (SPS): it defines the physical configuration of the data sources that populate the DW.
- Integration Transportation Diagram (ITD): it defines the physical structure of the ETL processes that extract, transform and load data into the DW. This diagram relates the SPS and the next diagram.
- Data Warehouse Physical Schema (DWPS): it defines the physical structure of the DW itself.
- Customization Transportation Diagram (CTD): it defines the physical structure of the exportation processes from the DW to the specific structures employed by clients. This diagram relates the DWPS and the next diagram.
- Client Physical Schema (CPS): it defines the physical configuration of the structures employed by clients in accessing the DW.

The SPS, DWPS, and CPS are based on the UML component and deployment diagrams, whereas ITD and CTD are only based on the deployment diagrams. These diagrams reflect the modeling aspects of the storage of data (Naiburg & Maksimchuk, 2001), such as the database size, information about where the database will reside (hardware and software), partitioning of the data, properties specific to the DBMS (*Database Management System*) chosen, etc.

The five proposed diagrams use an extension of UML that we have called *Database Deployment Profile*, which is formed by a series of stereotypes, tagged values and constraints.

Throughout the rest of this paper, we use an example to introduce the different diagrams we propose. This example is partly based on the enterprise DW sample database schema from Silverston, Inmon, and Graziano (1997). In this example, final users need a DW in order to analyze the main operations of the company. Because of this, the DW contains information about customers, customer invoices, budget details, products, and suppliers. Moreover, data about the employees, such as salaries, positions, and categories are also stored in the DW.

The operational data sources are stored in three servers:

1. The sales server, which contains the data about transactions and sales.

2. The CRM (*Customer Relationship Management*) server, which contains the data about the customers who buy products.

3. The HRM (*Human Resource Management*) server, which contains the data about employees, positions, salaries, etc.

Following our approach (Luján-Mora et al., 2002b), we structure the conceptual model into three levels:

Level 1 : Model definition. A package represents a star schema of a conceptual MD model. A dependency between two packages at this level indicates that the star schemas share at least one dimension, allowing us to consider conformed dimensions.

Level 2 : Star schema definition. A package represents a fact or a dimension of a star schema. A dependency between two dimension packages at this level indicates that the packages share at least one level of a dimension hierarchy.

Level 3 : Dimension/fact definition. A package is exploded into a set of classes that represent the hierarchy levels defined in a dimension package, or the whole star schema in the case of the fact package.

Figure 6 shows the first level of the DWCS, which represents the conceptual model of the DW. In our example, the first level is formed by three packages called Customer_Invoices Star, Positions Star, and Purchase_Invoices Star. A dashed arrow from one package to another one denotes a dependency between packages, i.e., the packages have some dimensions in common. The direction of the dependency indicates that the common dimensions shared by the two packages were first defined in the package pointed to by the arrow (to start with, we have to choose a star schema to define the dimensions, and then, the other schemas can use them with no need to define them again). If the common dimensions had been first defined in another package, the direction of the arrow would have been different. In any case, it is better to group together the definition of the common dimensions in order to reduce the number of dependencies. From now on, we will focus our discussion on the Customer_Invoices Star. This star schema represents the invoices belonging to customers.

Figure 7 shows the second level of the DWCS. The fact package Customer_Invoices is represented in the middle of the figure, while the dimension packages are placed around the fact package. As seen in Figure 7, a dependency is drawn from the fact package Customer_Invoices to each one of the dimension packages (Customers, Internal_Org, Products, and Time), because the fact package comprises the whole definition of the star schema, and therefore, uses the definitions of dimensions related to the fact. At level 2, it is possible to create a dependency from a fact package to a dimension package or between dimension packages, but we do not

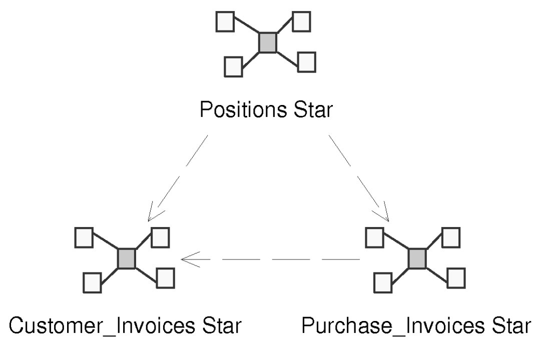


Figure 6: Data Warehouse Conceptual Schema: Level 1

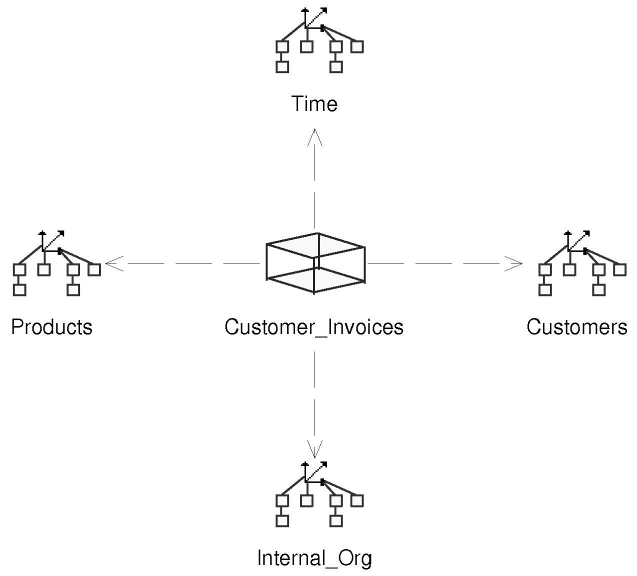


Figure 7: Data Warehouse Conceptual Schema: Level 2 of Customer_Invoices Star)

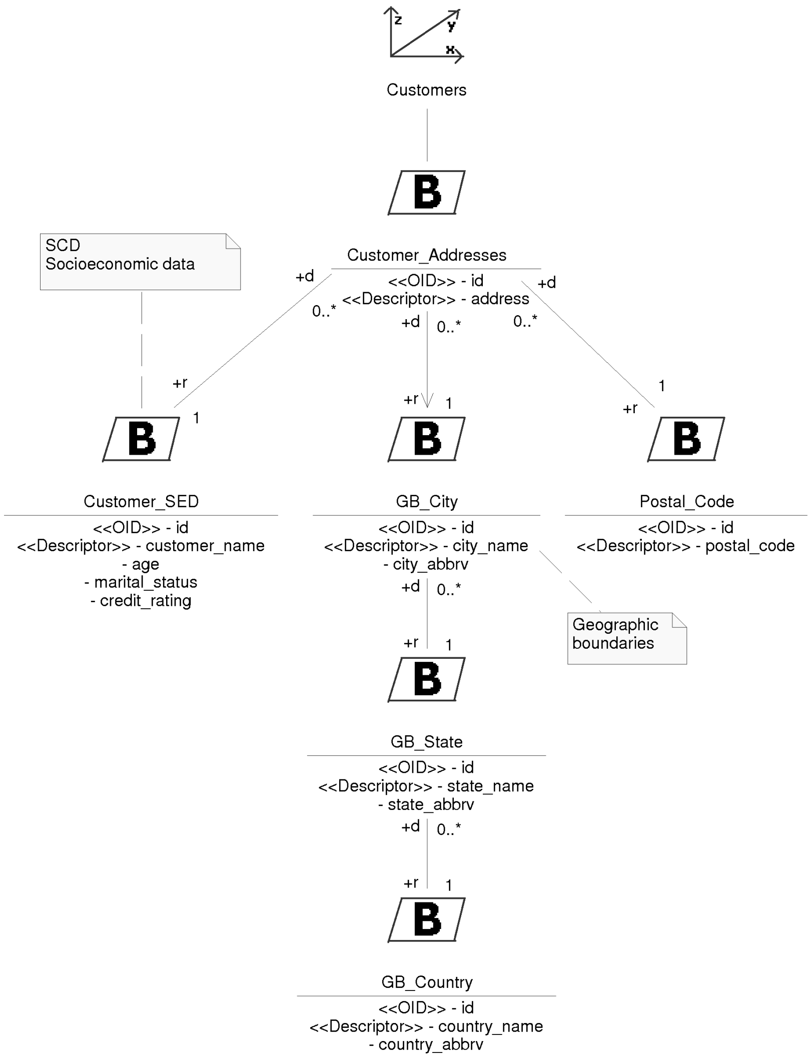


Figure 8: Data Warehouse Conceptual Schema: Customers dimension

allow a dependency from a dimension package to a fact package, since it is not semantically correct in our technique.

The content of the dimension and fact packages is represented at level 3. The diagrams at this level are only comprised of classes and associations among them. Figure 8 shows the level 3 of the Customers dimension package (Figure 7), which contains the definition of the dimension (Customers) and the different hierarchy levels (Customer_Addresses, Customer_SED, GB_City, Postal_Code, GB_City[7], etc.). The hierarchy of a dimension defines how the different OLAP operations (roll up, drill down, etc.) can be applied. In a UML note we highlight that Customer_SED (SED means *socioeconomic data*) is a *Slowly Changing Dimension* (SCD) and some kind of solution has to be selected during the implementation (Kimball, 1996).

Figure 9 shows the level 3 of the Products dimension. This dimension contains two alternative hierarchies: the category of the product (Category) and the supplier of the product (Products_Supplier, City, State, and Country). In Products_Supplier hierarchy level, msrp means *manufacturer's suggested retail price* and uom is the standard *unit of measure* used for the product.

[7] GB means *geographic boundaries*.

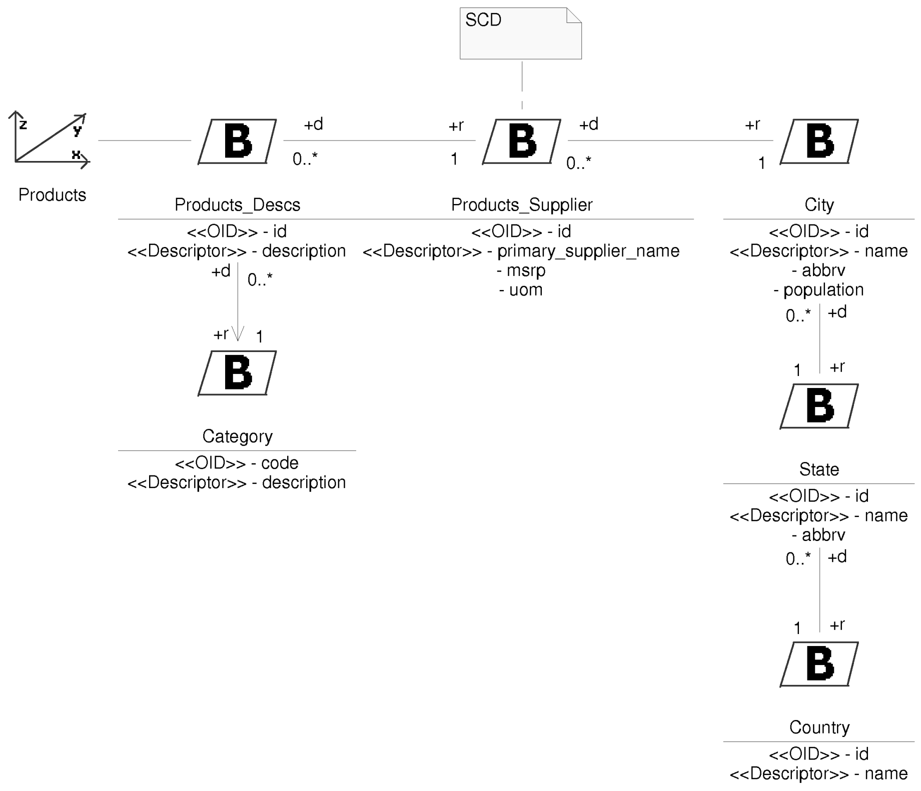


Figure 9: Data Warehouse Conceptual Schema: Products dimension

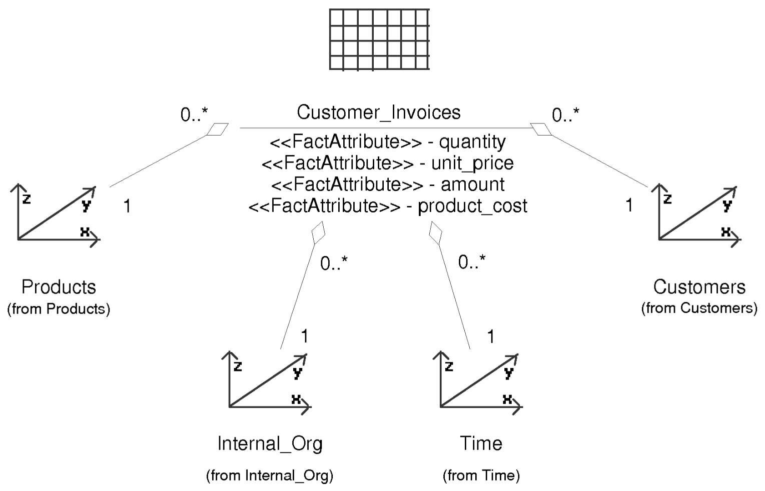


Figure 10: Data Warehouse Conceptual Schema: Customer_Invoices fact

Figure 10 shows the level 3 of the Customer_Invoices fact package. In this package, the whole star schema is displayed: the fact class is defined in this package and the dimensions with their corresponding hierarchy levels are imported from the dimension packages. Because of this, the name of the package where they have been previously defined appears below the package name, e.g., (from Products), (from Internal_Org). In order to avoid a cluttered diagram, we only show the attributes of the fact class (Customer_Invoices) and we hide the attributes and the hierarchy levels of the dimensions.

Figure 11 shows the *Data Warehouse Logical Schema* (DWLS), which represents the logical model of the DW. In this example, a ROLAP system has been selected for the implementation of the DW, which means the use of the relational model in the logical design of the DW. In Figure 11, seven classes adorned with the stereotype ≪Table≫ are shown: customers, customer_addresses, customer_invoices, internal_org_addresses, geographic_boundaries, product_snapshots, and products.

In the customer_invoices table, the attributes customer_id, bill-to-address, organization_id, org_address_id, and product_code are the foreign keys that connect the fact table with the dimension tables, whereas the attributes quantity, unit_price, amount, and product_cost represent the measures of the fact table. The attribute invoice_date represents a degenerate dimension[8], whereas the attribute load_date is the date the record was loaded into the DW and it is is used in the refresh process.

In the products and product_snapshots tables, these tables contain all the attributes of the different dimension levels (Figure 9) following the star schema approach (Kimball, 1996); some attributes have changed their names in order to avoid repeated names and some design decisions have been made. Moreover, we observe that we use UML notes to provide additional information to the diagram.

The following subsections present the five diagrams we propose for the physical design of DWs. In Section 5.1

5.1 Source Physical Schema

The SPS describes the origins of data of the DW from a physical point of view. Figure 12 shows the SPS of our example, which is formed by three servers called SalesServer, CRMServer, and HRMServer; for each one of them, the hardware and software configuration is displayed by means of tagged values. The first server hosts a database called Sales, whereas the second server hosts a database called Customers.

In our *Database Deployment Profile*, when the storage system is a RDBMS (*Relational Database Management System*), we make use of the *UML for Profile Database* (Naiburg & Maksimchuk, 2001) that defines a series of stereotypes including ≪Database≫, ≪Schema≫, or ≪Tablespace≫. Moreover, we have defined our own set of stereotypes: in Figure 12, we can see the stereotypes ≪Server≫ that defines a computer that performs server functions, ≪Disk≫ to represent a physical disk drive and ≪InternalBus≫ to define the type of communication between two elements. In our approach, we represent the configuration parameters of the tablespaces (e.g., size of the tablespace) by means of tagged values; however, these parameters vary greatly depending on the DBMS, so we only provide a set of common parameters. As UML is extensible, the designer can add additional tagged values as needed to accomplish all the modeling needs of a particular DBMS.

Moreover, whenever we need to specify additional information in a diagram, we make use of the UML notes to incorporate it. For example, in Figure 12 we have used two notes to indicate how the data is distributed into the two existing tablespaces; the tablespace TS_Sales98 holds the data about the sales before or in 1998, whereas the tablespace TS_Sales holds the sales after 1998.

[8]Kimball (1996) coined the term *degenerate dimensions* for data items that perform much the same function as dimensions but they are stored in the fact table, but they are not foreign key links through to dimension tables.

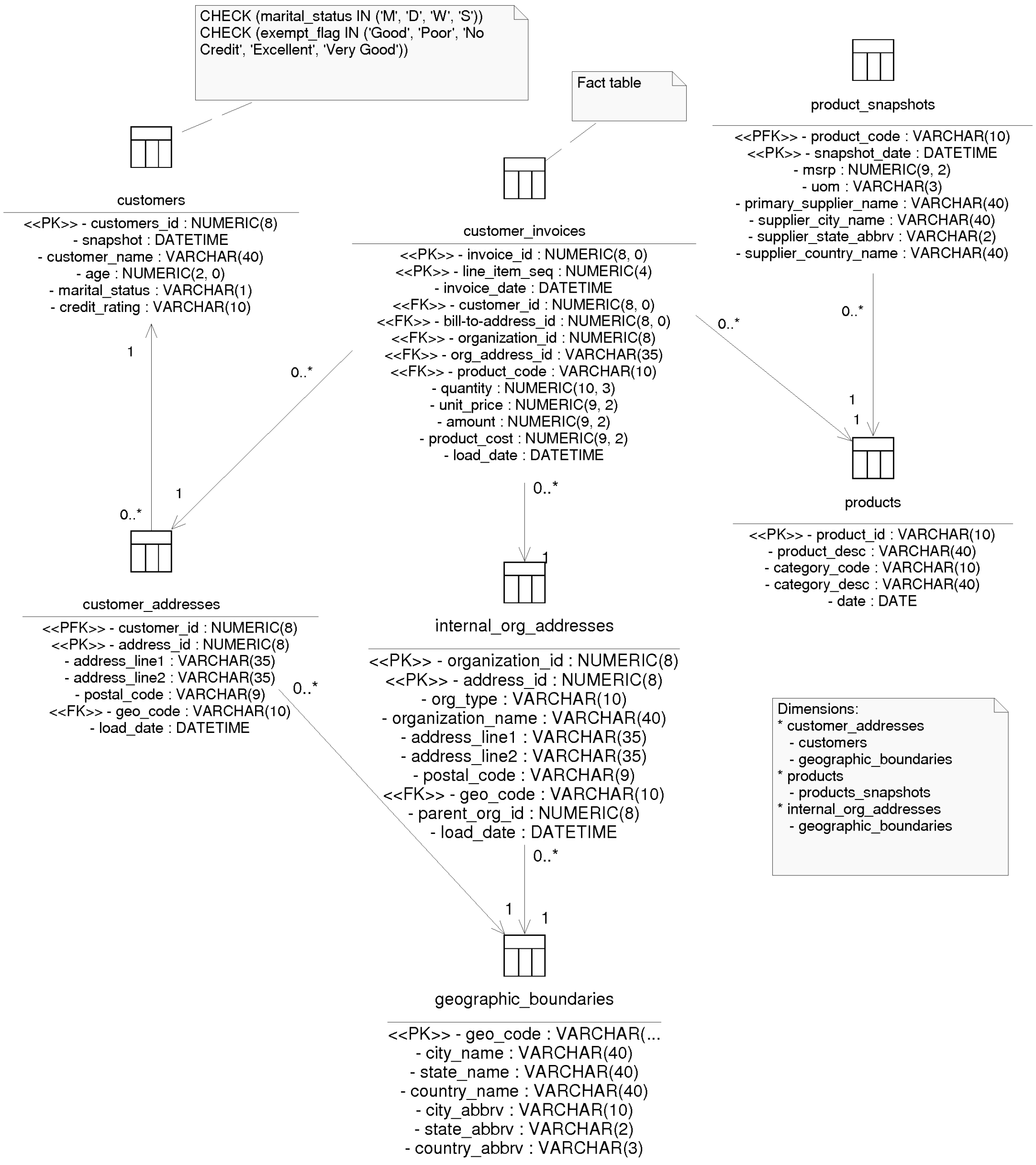

CHECK (marital_status IN ('M', 'D', 'W', 'S'))
CHECK (exempt_flag IN ('Good', 'Poor', 'No Credit', 'Excellent', 'Very Good'))
Fact table
product_snapshots
<<PFK>> - product_code : VARCHAR(10)
<<PK>> - snapshot_date : DATETIME
- msrp : NUMERIC(9, 2)
- uom : VARCHAR(3)
- primary_supplier_name : VARCHAR(40)
- supplier_city_name : VARCHAR(40)
- supplier_state_abbrv : VARCHAR(2)
- supplier_country_name : VARCHAR(40)
customers
<<PK>> - customers_id : NUMERIC(8)
- snapshot : DATETIME
- customer_name : VARCHAR(40)
- age : NUMERIC(2, 0)
- marital_status : VARCHAR(1)
- credit_rating : VARCHAR(10)
customer_invoices
<<PK>> - invoice_id : NUMERIC(8, 0)
<<PK>> - line_item_seq : NUMERIC(4)
- invoice_date : DATETIME
<<FK>> - customer_id : NUMERIC(8, 0)
<<FK>> - bill-to-address_id : NUMERIC(8, 0)
<<FK>> - organization_id : NUMERIC(8)
<<FK>> - org_address_id : VARCHAR(35)
<<FK>> - product_code : VARCHAR(10)
- quantity : NUMERIC(10, 3)
- unit_price : NUMERIC(9, 2)
- amount : NUMERIC(9, 2)
- product_cost : NUMERIC(9, 2)
- load_date : DATETIME
0..*
1
0..*
1
1
0..*
1
0..*
1
0..*
1
0..*
1
products
<<PK>> - product_id : VARCHAR(10)
- product_desc : VARCHAR(40)
- category_code : VARCHAR(10)
- category_desc : VARCHAR(40)
- date : DATE
customer_addresses
<<PFK>> - customer_id : NUMERIC(8)
<<PK>> - address_id : NUMERIC(8)
- address_line1 : VARCHAR(35)
- address_line2 : VARCHAR(35)
- postal_code : VARCHAR(9)
<<FK>> - geo_code : VARCHAR(10)
- load_date : DATETIME
0..*
internal_org_addresses
<<PK>> - organization_id : NUMERIC(8)
<<PK>> - address_id : NUMERIC(8)
- org_type : VARCHAR(10)
- organization_name : VARCHAR(40)
- address_line1 : VARCHAR(35)
- address_line2 : VARCHAR(35)
- postal_code : VARCHAR(9)
<<FK>> - geo_code : VARCHAR(10)
- parent_org_id : NUMERIC(8)
- load_date : DATETIME
0..*
1
1
Dimensions:
* customer_addresses
- customers
- geographic_boundaries
* products
- products_snapshots
* internal_org_addresses
- geographic_boundaries
geographic_boundaries
<<PK>> - geo_code : VARCHAR(...
- city_name : VARCHAR(40)
- state_name : VARCHAR(40)
- country_name : VARCHAR(40)
- city_abbrv : VARCHAR(10)
- state_abbrv : VARCHAR(2)
- country_abbrv : VARCHAR(3)

Figure 11: Logical model (ROLAP) of the data warehouse

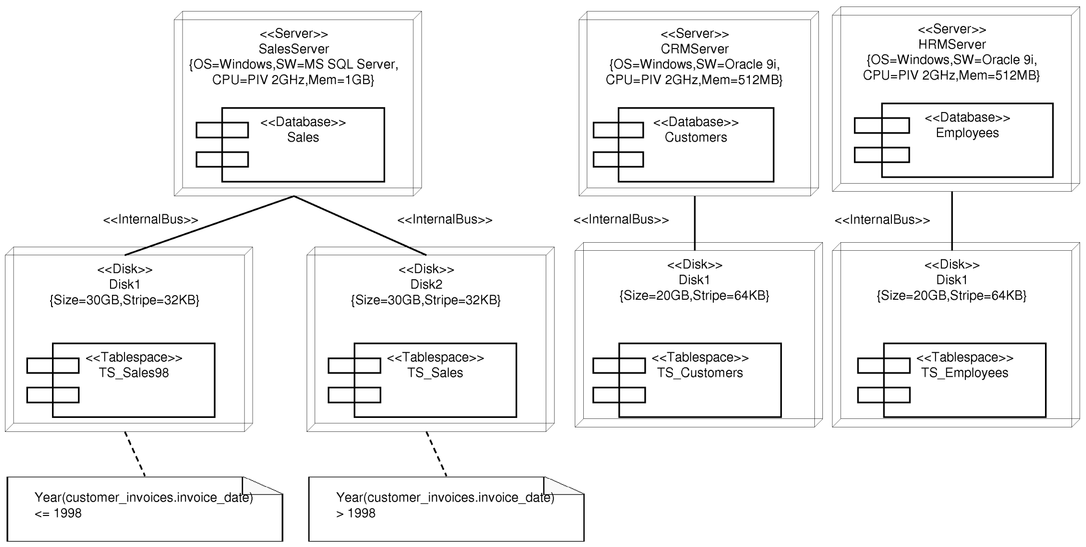


Figure 12: Source Physical Schema: deployment diagram

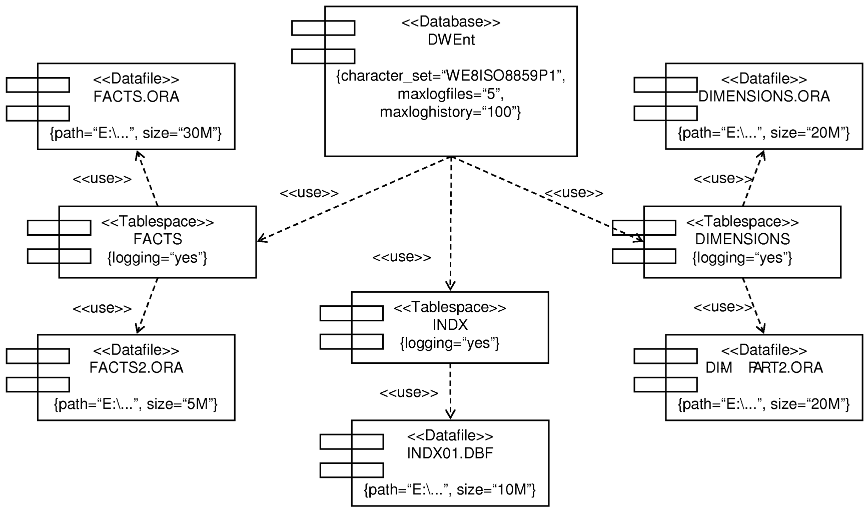
5.2 Data Warehouse Physical Schema

The DWPS shows the physical aspects of the implementation of the DW. This diagram is divided into two parts: the component diagram and the deployment diagram. In the first diagram, the configuration of the logical structures used to store the DW is shown. For example, in Figure 13, the DW is implemented by means of a database called DWEnt, which is formed by three tablespaces called FACTS, DIMENSIONS, and INDX. The datafiles that the tablespaces use are given as well: FACTS.ORA, FACTS2.ORA, DIMENSIONS.ORA, DIM-PART2.ORA, and INDX01.DBF.

Figure 14 shows a part of the definition of the tablespaces: the tablespace FACTS hosts the table customer_invoices and the tablespace DIMENSIONS hosts the dimension tables customers, products, product_snapshots, and the rest of the tables (not shown in the diagram for the sake of simplicity). Below the name of each table, the text (from ROLAP1) is included, which indicates that the tables have been previously defined in a package called ROLAP1 (Figure 11). It is important to highlight that the logical structure defined in the DWLS are reused in this diagram and, therefore, we avoid any possibility of ambiguity or incoherence.

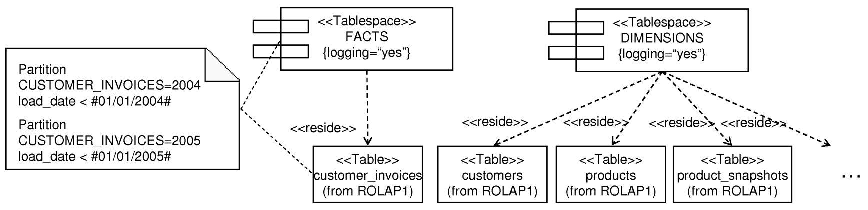
In the second diagram, the deployment diagram, different aspects relative to the hardware and software configuration are specified. Moreover, the physical distribution of the logical structures previously defined in the component diagrams is also represented. For example, in Figure 15, we can observe the configuration of the server that hosts the DW: the server is composed of two disks, one for the operating system (Linux) and the applications (Oracle) and another one for the different datafiles (FACTS.ORA, FACTS2.ORA, etc.) that are used by the database (Figure 13).

One of the advantages of our approach is that it allows evaluation and discussion of different implementations during the first stages in the design of a DW. In this way, the designer can anticipate some implementation or performance problems. For example, an alternative configuration of the physical structure of the DW can be established, as shown in Figure 16. In this second alternative, a RAID 0 systems has been chosen to host the datafiles that are used by the tablespace FACTS in order to improve the response time of the disk drive and the performance of the system in general. From these two alternative configurations, the DW designer and the DW administrator can discuss the pros and cons of each option.



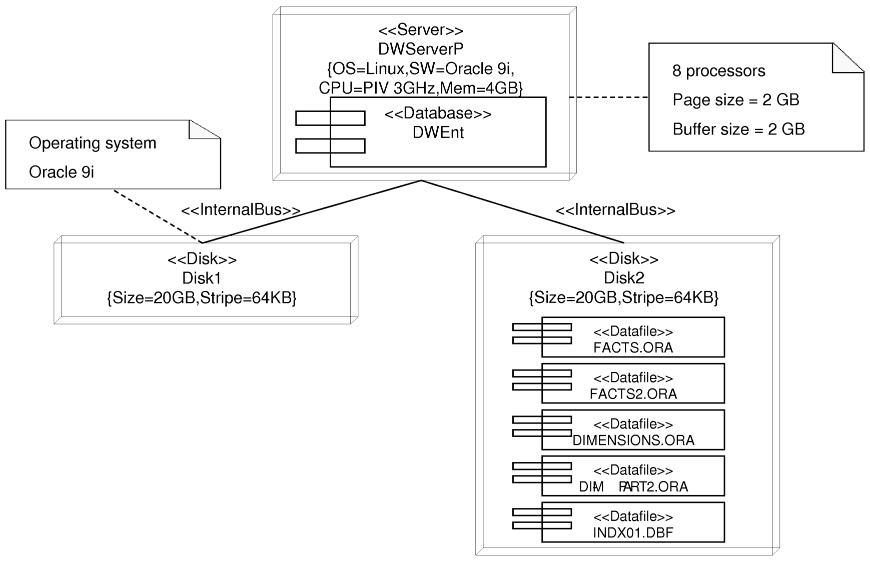
<<Database>> DWEnt {character_set="WE8ISO8859P1", maxlogfiles="5", maxloghistory="100"}
<<Datafile>> FACTS.ORA {path="E:\...", size="30M"}
<<Datafile>> DIMENSIONS.ORA {path="E:\...", size="20M"}
<<Tablespace>> FACTS {logging="yes"}
<<Tablespace>> DIMENSIONS {logging="yes"}
<<Tablespace>> INDX {logging="yes"}
<<Datafile>> FACTS2.ORA {path="E:\...", size="5M"}
<<Datafile>> DIM_PART2.ORA {path="E:\...", size="20M"}
<<Datafile>> INDX01.DBF {path="E:\...", size="10M"}
<<use>>

Figure 13: Data Warehouse Physical Schema: component diagram (part 1)



Partition CUSTOMER_INVOICES=2004 load_date < #01/01/2004#
Partition CUSTOMER_INVOICES=2005 load_date < #01/01/2005#
<<Tablespace>> FACTS {logging="yes"}
<<Tablespace>> DIMENSIONS {logging="yes"}
<<reside>>
<<Table>> customer_invoices (from ROLAP1)
<<Table>> customers (from ROLAP1)
<<Table>> products (from ROLAP1)
<<Table>> product_snapshots (from ROLAP1)
...

Figure 14: Data Warehouse Physical Schema: component diagram (part 2)



<<Server>> DWServerP {OS=Linux,SW=Oracle 9i, CPU=PIV 3GHz,Mem=4GB}
<<Database>> DWEnt
8 processors
Page size = 2 GB
Buffer size = 2 GB
Operating system
Oracle 9i
<<InternalBus>>
<<Disk>> Disk1 {Size=20GB,Stripe=64KB}
<<Disk>> Disk2 {Size=20GB,Stripe=64KB}
<<Datafile>> FACTS.ORA
<<Datafile>> FACTS2.ORA
<<Datafile>> DIMENSIONS.ORA
<<Datafile>> DIM_PART2.ORA
<<Datafile>> INDX01.DBF

Figure 15: Data Warehouse Physical Schema: deployment diagram (version 1)

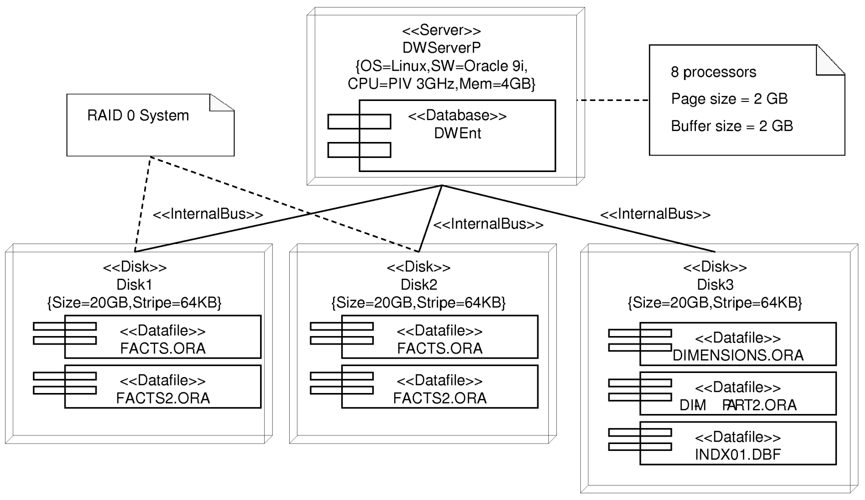


Figure 16: Data Warehouse Physical Schema: deployment diagram (version 2)

5.3 Integration Transportation Diagram

The ITD defines the physical structure of the ETL processes used in the loading of data in the DW from the data sources. On the one hand, the data sources are represented by means of the SPS and, on the other hand, the DW is represented by means of the DWPS. Since the SPS and the DWPS have been defined previously, in this diagram they are imported.

For example, the ITD for our running example is shown in Figure 17. On the left hand side of this diagram, different data source servers are represented: SalesServer, CRMServer, and HRMServer, which have been previously defined in Figure 12; on the right hand side, the DWServerP, previously defined in Figure 15, is shown. Moreover, the DWServerS, a physical standby database[9] is also included in the design.

In Figure 17, the ETLServer is introduced, an additional server that is used to execute the ETL processes. This server communicates with the rest of the servers by means of a series of specific protocols: OLEDB to communicate with SalesServer because it uses Microsoft SQLServer[10] and OCI (*Oracle Call Interface*) to communicate with CRMServer, HRMServer, and DWServer because they use Oracle.

5.4 Client Physical Schema

The CPS defines the physical structure of the specific structures that are used by the clients to access the DW. Diverse configurations exist that can be used: exportation of data to *data marts*, use of an OLAP server, etc. In our example, we have chosen a client/server architecture and the same DW server provides access to data for the clients. Therefore, we do not need to define a specific structure for the clients.

5.5 Customization Transportation Diagram

The CTD defines the exportation processes from the DW towards the specific structures used by the clients. In this diagram, the DW is represented by means of the DWPS and clients are represented by means of the CPS. Since the DWPS and the CPS have been previously defined, in this diagram we do not have to define them again, but they are directly imported.

[9]A physical standby database is a byte by byte exact copy of the primary database. The primary database records all changes and sends them to the standby database. A standby database environment is meant for disastrous failures.

[10]The configuration of a server is defined by means of tagged values: OS, SW, CPU, etc.

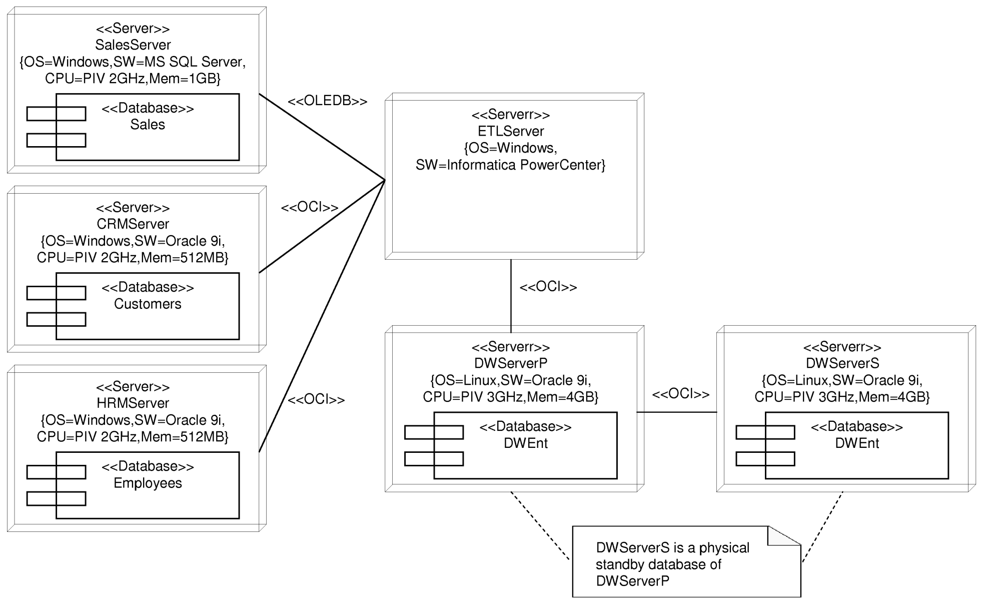

<<Server>>
SalesServer
{OS=Windows,SW=MS SQL Server, CPU=PIV 2GHz,Mem=1GB}
<<Database>>
Sales
<<Server>>
CRMServer
{OS=Windows,SW=Oracle 9i, CPU=PIV 2GHz,Mem=512MB}
<<Database>>
Customers
<<Server>>
HRMServer
{OS=Windows,SW=Oracle 9i, CPU=PIV 2GHz,Mem=512MB}
<<Database>>
Employees
<<OLEDB>>
<<OCI>>
<<OCI>>
<<Serverr>>
ETLServer
{OS=Windows, SW=Informatica PowerCenter}
<<OCI>>
<<Serverr>>
DWServerP
{OS=Linux,SW=Oracle 9i, CPU=PIV 3GHz,Mem=4GB}
<<Database>>
DWEnt
<<OCI>>
<<Serverr>>
DWServerS
{OS=Linux,SW=Oracle 9i, CPU=PIV 3GHz,Mem=4GB}
<<Database>>
DWEnt
DWServerS is a physical standby database of DWServerP

Figure 17: Integration Transportation Diagram: deployment diagram

For example, in Figure 18, the CTD of our running example is shown. On the left hand side of this diagram, part of the DWPS, which has been previously defined in Figure 15, is shown; on the right hand side, three types of clients who will use the DW are shown: a Web client with operating system Apple Macintosh, a Web client with operating system Microsoft Windows and, finally, a client with a specific desktop application (MicroStrategy) with operating system Microsoft Windows. Whereas both Web clients communicate with the server by means of HTTP, the desktop client uses ODBC (the *Open Database Connectivity*).

In the following section, we explain how to use our component and deployment diagrams to implement a DW on an Oracle database server.

6 Implementation in Oracle

Creating a database consists basically of the following steps:

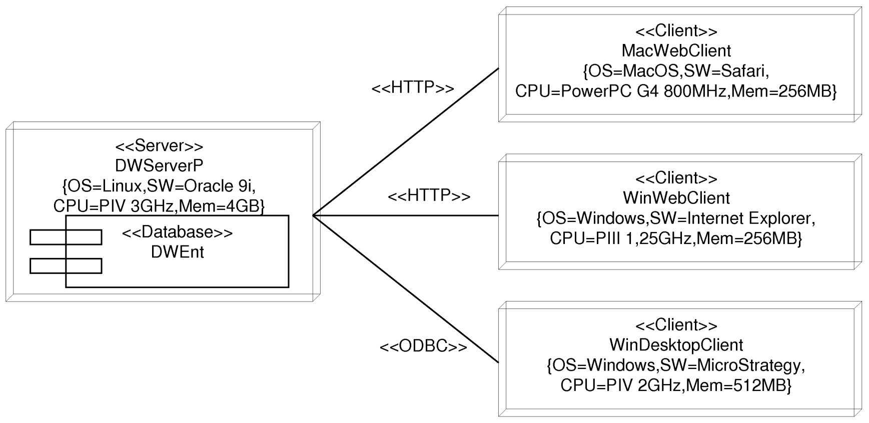

<<Server>>
DWServerP
{OS=Linux,SW=Oracle 9i, CPU=PIV 3GHz,Mem=4GB}
<<Database>>
DWEnt
<<HTTP>>
<<HTTP>>
<<ODBC>>
<<Client>>
MacWebClient
{OS=MacOS,SW=Safari, CPU=PowerPC G4 800MHz,Mem=256MB}
<<Client>>
WinWebClient
{OS=Windows,SW=Internet Explorer, CPU=PIII 1,25GHz,Mem=256MB}
<<Client>>
WinDesktopClient
{OS=Windows,SW=MicroStrategy, CPU=PIV 2GHz,Mem=512MB}

Figure 18: Customization Transportation Diagram: deployment diagram

- Creating the database's datafiles[11], its control files[12] and its redo log files[13].
- Creating the system tablespaces.
- Creating the data dictionary (tables, views, etc.).

6.1 Creating the Database

Before proceeding, we should point out that the information about the database repository to be created (Figure 13) was passed to our database administrator, who created the database. Following Oracle recommendations, the database creation is an operation that should only be executed by the administrator, who is also responsible for granting database permissions to users. Then, if users need to create several databases or wish to organize a big database consisting of a considerable amount of tables, the DW administrator should organize them by specifying *tablespaces* and decide which tables to locate in each created *tablespace*. Therefore, the concept of database for Oracle is at a high administrative level, avoiding programmers and even database designers to create databases. For this reason, we do not do a deep study on the *create database* statement in this section, instead, we will concentrate on the statements and stages that the DW designer has accomplished from our UML *component* and *deployment diagrams* described in previous sections. In the following subsections, we will show some SQL sentences automatically generated by the *Oracle Enterprise Manager Console* tool to implement the DW and some snapshots from the same tool to see the created *tablespaces*, *tables*, *indexes*, etc.

6.2 Creating the Tablespaces

The created database is called DWEnt. Then, the first accomplished task has been to specify the *tablespaces* where allocating the tables. As seen in Figure 13, we need to create three tablespaces, one for the facts (FACTS), another one for the dimensions (DIMENSIONS), and the last one for the indexes (INDX); furthermore, according to the same component diagram, the tablespaces for the facts and the dimensions need to be defined in two *datafiles*. Datafiles are the logical structure in which Oracle structures a database, i.e., the place where allocating the structures (e.g., tables, indices, etc.) defined within a tablespace. Due to amount of data to be stored in the tablespaces for facts and dimensions, the designer decided to specify two datafiles for each tablespace (Figure 13).

Figure 19 shows the definition of the tablespaces as seen in the *Oracle Enterprise Console Manager*. In the following SQL statements, we can see the SQL patterns generated by this management tool for defining tablespaces. We can also notice the datafiles that will be used for the DW to store fact tables, dimension tables, views, and so on.

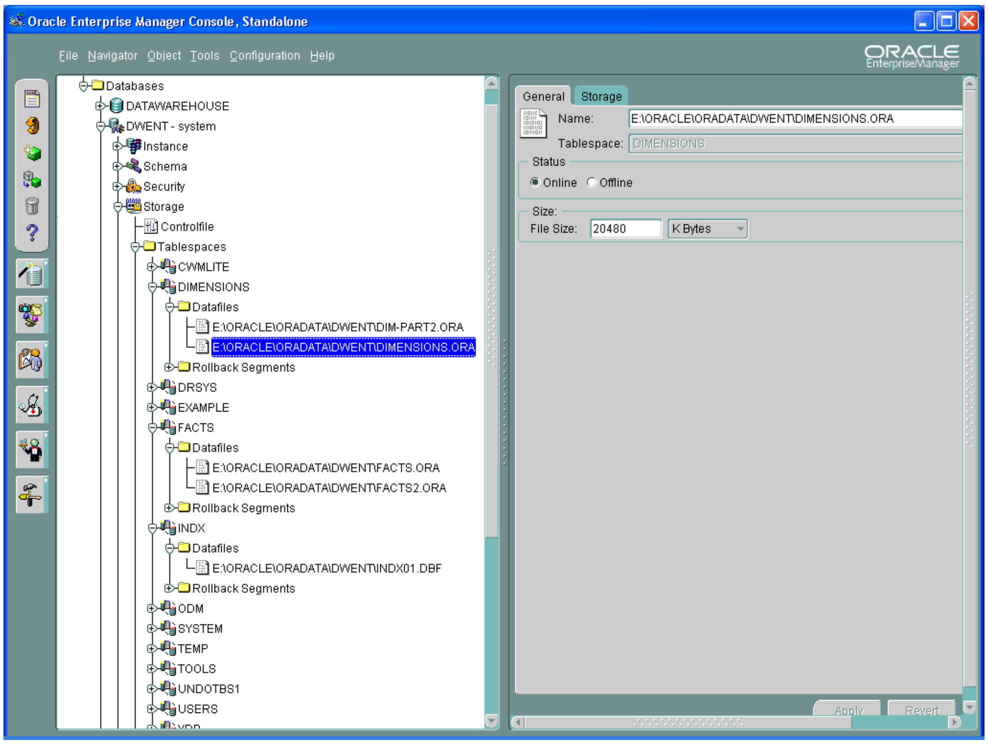
```
CREATE TABLESPACE "FACTS"
    LOGGING
    DATAFILE 'E:\ORACLE\ORADATA\DWENT\FACTS.ora' SIZE 30M,
    'E:\ORACLE\ORADATA\DWENT\FACTS2.ORA' SIZE 5M
    EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT  AUTO

CREATE TABLESPACE "DIMENSIONS"
    LOGGING
    DATAFILE 'E:\ORACLE\ORADATA\DWENT\DIMENSIONS.ora' SIZE 20M,
```

[11]Files where the database server will host the data in the database structures, such as tables, materialized views, etc.

[12]The control files of a database store the status of the physical structure of the database. It contains (but is not limited to) the following types of information: database name and creation date, tablespace and datafile records, etc.

[13]The redo log files record all changes made in datafiles. If something happens to one of the datafiles of a database, a backed up datafile can be restored and the redo, that was written since, which brings the datafile to the state it had before it became unavailable (crash recovery).



Name: E:\ORACLE\ORADATA\DWENT\DIMENSIONS.ORA
Tablespace: DIMENSIONS
Status
[✓ Online] Offline
Size:
File Size: 20480 K Bytes

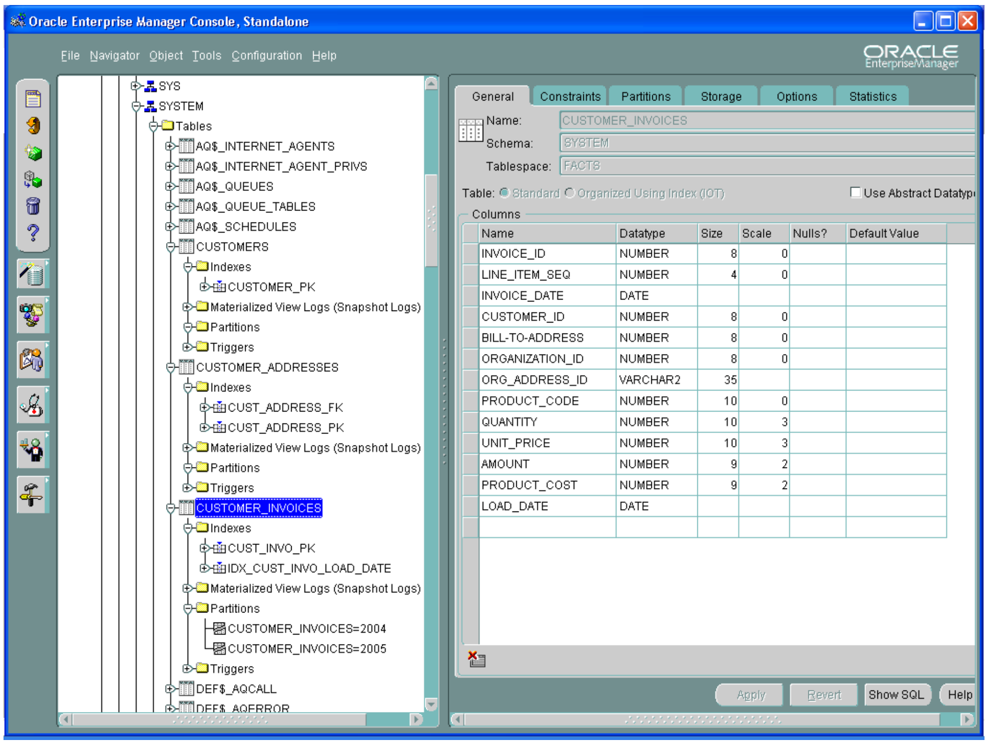
Figure 19: Tablespaces definition in Oracle

```
    'E:\ORACLE\ORADATA\DWENT\DIM-PART2.ORA' SIZE 20M
    EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT  AUTO
```

6.3 Creating the Data Dictionary

Once both tablespaces and datafiles have been created within a database, the next step is to define fact and dimension tables. First of all, in the following SQL sentences, we can see how we have specified the *product dimension* table. Apart from the columns, we should point out the fact that the table has been created partitioning it into two different partitions, once for products in year 2004, and another one for new products in 2005. Furthermore, due to the fact that a partitioning has been defined, an index for the column in which we define the partitioning (i.e. date), has automatically been created. The SQL statements for the customers and customer_addresses tables are more simple as no partitioning was defined for these tables in Figure 14. Besides, all dimension tables are defined in the *dimension tablespace*.

```
CREATE TABLE "SYSTEM"."PRODUCTS" (
    "PRODUCT_ID" NUMBER(10) NOT NULL,
    "PRODUCT_DESC" VARCHAR2(40) NOT NULL,
    "CATEGORY_CODE" VARCHAR2(10) NOT NULL,
    "CATEGORY_DESC" VARCHAR2(40) NOT NULL,
    "DATE" DATE NOT NULL,
    CONSTRAINT "PRODUCT_PK" PRIMARY KEY("PRODUCT_ID"))
    TABLESPACE "DIMENSIONS"
    PARTITION BY RANGE ("DATE") (PARTITION "PRODUCT=2004"
    VALUES LESS THAN  (TO_DATE('2004-1-1','YYYY-MM-DD'))
    TABLESPACE "DIMENSIONS" ,
    PARTITION "PRODUCT=2005"
    VALUES LESS THAN  (TO_DATE('2005-1-1','YYYY-MM-DD'))
```


Oracle Enterprise Manager Console, Standalone
File Navigator Object Tools Configuration Help
ORACLE EnterpriseManager
General Constraints Partitions Storage Options Statistics
Apply Revert Show SQL Help

SYS
SYSTEM
Tables
AQ$_INTERNET_AGENTS
AQ$_INTERNET_AGENT_PRIVS
AQ$_QUEUES
AQ$_QUEUE_TABLES
AQ$_SCHEDULES
CUSTOMERS
Indexes
CUSTOMER_PK
Materialized View Logs (Snapshot Logs)
Partitions
Triggers
CUSTOMER_ADDRESSES
Indexes
CUST_ADDRESS_FK
CUST_ADDRESS_PK
Materialized View Logs (Snapshot Logs)
Partitions
Triggers
CUSTOMER_INVOICES
Indexes
CUST_INVO_PK
IDX_CUST_INVO_LOAD_DATE
Materialized View Logs (Snapshot Logs)
Partitions
CUSTOMER_INVOICES=2004
CUSTOMER_INVOICES=2005
Triggers
DEF$_AQCALL
DEF$_AQERROR

Name: CUSTOMER_INVOICES
Schema: SYSTEM
Tablespace: FACTS
Table: ● Standard ○ Organized Using Index (IOT) ☐ Use Abstract Datatype
Columns

Name	Datatype	Size	Scale	Nulls?	Default Value
INVOICE_ID	NUMBER	8	0		
LINE_ITEM_SEQ	NUMBER	4	0		
INVOICE_DATE	DATE				
CUSTOMER_ID	NUMBER	8	0		
BILL-TO-ADDRESS	NUMBER	8	0		
ORGANIZATION_ID	NUMBER	8	0		
ORG_ADDRESS_ID	VARCHAR2	35			
PRODUCT_CODE	NUMBER	10	0		
QUANTITY	NUMBER	10	3		
UNIT_PRICE	NUMBER	10	3		
AMOUNT	NUMBER	9	2		
PRODUCT_COST	NUMBER	9	2		
LOAD_DATE	DATE				

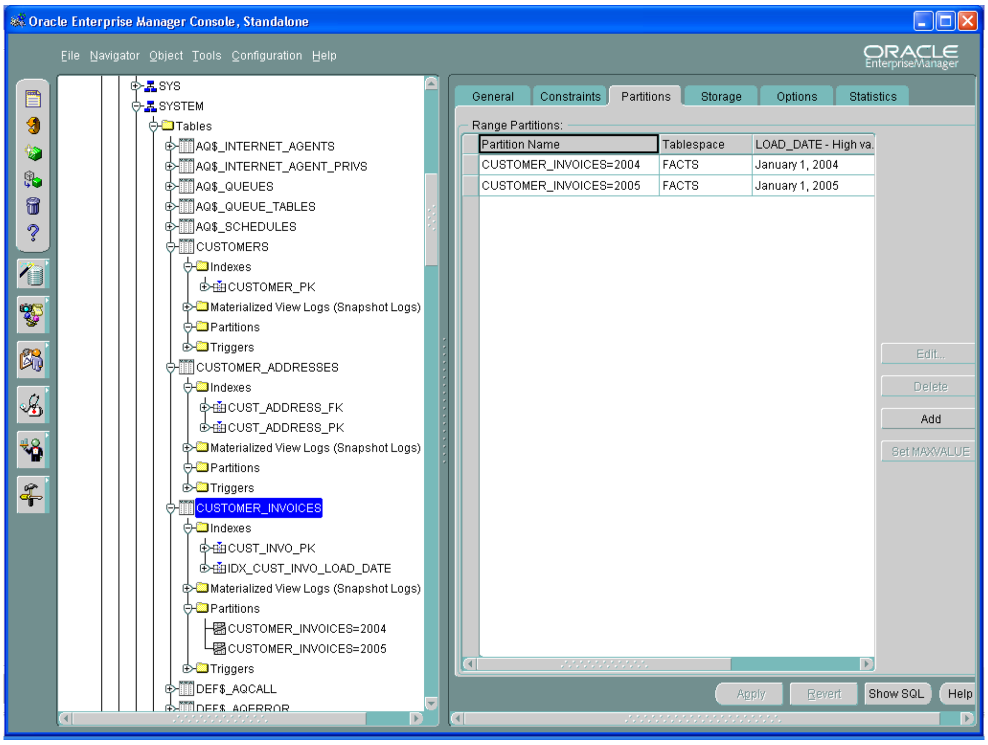
Figure 20: Defintion of customer_invoices table in Oracle

```
    TABLESPACE "DIMENSIONS" );
    CREATE INDEX SYSTEM.IDX_PRODUCTS ON SYSTEM.PRODUCTS ("DATE") LOCAL

CREATE TABLE "SYSTEM"."CUSTOMERS" (
    "CUSTOMERS_ID" NUMBER(8) NOT NULL,
    "SNAPSHOT" DATE NOT NULL,
    "CUSTOMER_NAME" VARCHAR2(40) NOT NULL,
    "AGE" NUMBER(2) NOT NULL,
    "MARITAL_STATUS" VARCHAR2(1) NOT NULL,
    "CREDIT_RATING" VARCHAR2(10) NOT NULL,
    CONSTRAINT "CUSTOMER_PK" PRIMARY KEY("CUSTOMERS_ID"))
    TABLESPACE "DIMENSIONS"

CREATE TABLE "SYSTEM"."CUSTOMER_ADDRESSES" (
    "CUSTOMER_ID" NUMBER(8) NOT NULL,
    "ADDRESS_ID" NUMBER(8) NOT NULL,
    "ADDRESS_LINE1" VARCHAR2(35) NOT NULL,
    "ADDRESS_LINE2" VARCHAR2(35) NOT NULL,
    "POSTAL_CODE" VARCHAR2(9) NOT NULL,
    CONSTRAINT "CUST_ADDRESS_PK" PRIMARY KEY("CUSTOMER_ID", "ADDRESS_ID"),
    CONSTRAINT "CUST_ADDRESS_FK" FOREIGN KEY("CUSTOMER_ID")
    REFERENCES "SYSTEM"."CUSTOMERS"("CUSTOMERS_ID")
    TABLESPACE "DIMENSIONS"
```

Figure 20 shows the definition of the columns of the fact table customer_invoices in Oracle. Another partitioning has been created in this table. Again, our DW is intended to locate data for every year in each different partition (Figure 14), and therefore, the column in which we base our partition is load_date. Instead of the previous dimension tables, this fact table is



Range Partitions:

Partition Name	Tablespace	LOAD_DATE - High va.
CUSTOMER_INVOICES=2004	FACTS	January 1, 2004
CUSTOMER_INVOICES=2005	FACTS	January 1, 2005

Figure 21: Definition of a partition on customer_invoices table in Oracle

defined in the *fact tablespace*. Moreover, the index creation SQL statement has been slightly changed, because the following SQL statement has been automatically generated by the *Oracle Enterprise Manager Console*, and thus, the index name was automatically specified based on the database and table names. Then, this index name exceeded the longest index name allowed by Oracle. Therefore, the index name was manually shortened. In Figure 21 we include the definition of the partition on customer_invoices as it is shown in Oracle.

```
CREATE TABLE "SYSTEM"."CUSTOMER_INVOICES" (
    "INVOICE_ID" NUMBER(8) NOT NULL,
    "LINE_ITEM_SEQ" NUMBER(4) NOT NULL,
    "INVOICE_DATE" DATE NOT NULL,
    "CUSTOMER_ID" NUMBER(8) NOT NULL,
    "BILL-TO-ADDRESS" NUMBER(8) NOT NULL,
    "ORGANIZATION_ID" NUMBER(8) NOT NULL,
    "ORG_ADDRESS_ID" VARCHAR2(35) NOT NULL,
    "PRODUCT_CODE" NUMBER(10) NOT NULL,
    "QUANTITY" NUMBER(10, 3) NOT NULL,
    "UNIT_PRICE" NUMBER(10, 3) NOT NULL,
    "AMOUNT" NUMBER(9, 2) NOT NULL,
    "PRODUCT_COST" NUMBER(9, 2) NOT NULL,
    "LOAD_DATE" DATE NOT NULL,
    CONSTRAINT "CUST_INVO_PK" PRIMARY KEY("INVOICE_ID",
    "LINE_ITEM_SEQ"),
    CONSTRAINT "CUST_INVO_FK" FOREIGN KEY("CUSTOMER_ID",
    "BILL-TO-ADDRESS")
    REFERENCES "SYSTEM"."CUSTOMER_ADDRESSES"("CUSTOMER_ID",
    "ADDRESS_ID"),
    CONSTRAINT "CUST_INVO_FK2" FOREIGN KEY("PRODUCT_CODE")
    REFERENCES "SYSTEM"."PRODUCTS"("PRODUCT_ID"),
    CONSTRAINT "CUST_INVO_FK3" FOREIGN KEY("ORGANIZATION_ID",
    "ORG_ADDRESS_ID")
    REFERENCES "SYSTEM"."INTERNAL_ORG_ADDRESSES"("ORGANIZATION_ID",
    "ADDRESS_ID"))
    TABLESPACE "FACTS"
    PARTITION BY RANGE ("LOAD_DATE") (PARTITION
    "CUSTOMER_INVOICES=2004"
    VALUES LESS THAN  (TO_DATE('2004-1-1','YYYY-MM-DD'))
    TABLESPACE "FACTS" ,
    PARTITION "CUSTOMER_INVOICES=2005"
    VALUES LESS THAN  (TO_DATE('2005-1-1','YYYY-MM-DD'))
    TABLESPACE "FACTS" );
    CREATE INDEX SYSTEM.IDX_CUST_INVO_LOAD_DATE ON
    SYSTEM.CUSTOMER_INVOICES ("LOAD_DATE") LOCAL
```

In this section we have shown how to accomplish the implementation of a DW from our *component* and *deployment diagrams*. We believe that the implementation issues considered in our techniques are useful for the final implementation, even more, the DW administrator has implemented the DW according to our physical modeling schema.

7 Conclusions and Future Work

After about 15 years of research and development in Data Warehouses (DW), few efforts have been dedicated to the modeling of the physical design of a DW from the early stages of a project. In this paper, we have presented an adaptation of the component and deployment diagrams of the Unified Modeling Language (UML), the standard graphical notation for modeling software application needs, for the modeling of the physical design of a DW. This technique is part of our DW engineering process (Luján-Mora & Trujillo, 2004a) that addresses the design and development of both the DW back-end and front-end. Our method provides a unifying framework that facilitates the integration of different DW models. For example, the DW designers work with the DW administrators to understand the storage needed for the data. Our approach helps the DW designers to coordinate their efforts to include the hardware configuration (servers and drives necessary for the data) as well as the best way to organize the data into the database logical structures (tablespaces and tables).

Moreover, the UML component and deployment diagrams allow the designers to specify both hardware, software, and middleware needs for a DW project. The main advantages provided by our approach are as follows:

- It is part of an integrated approach in which we use different diagrams, always following the same standard notation based on UML, for modeling all main aspects of a DW.
- It can be used for the traceability of the design of a DW, from the conceptual model to the physical model.
- It allows reducing the overall development cost as we accomplish implementation issues from the early stages of a DW project. We should take into account that modifying these aspects in subsequent design phases may result in increasing the total cost of the project.
- It supports different levels of abstraction by providing different levels of details for the same diagram.

Since our approach is based on UML, there are different CASE (Computer-Aided Software Engineering) tools that can support our approach and having the entire design in one language

(UML) breaks down the barriers of communication between the different participants in a DW project.

Regarding future work, we are working on the formal definition of our approach by means of Object Constraint Language (OCL), because the formal definition is a well-defined way to construct a model and avoids confusion and ambiguity. We plan to provide guidelines on developing a DW by means of our DW engineering process. These guidelines will include validation checks to ensure that nothing is missed when going from one step to the next. Finally, we also plan to quantify the actual improvement in the design of a DW due to our approach.

Acknowledgments

We would like to thank Panos Vassiliadis for his helpful comments during the writing of the first version of this paper.

This work has been partially supported by the METASIGN project (TIN2004-00779) from the Spanish Ministry of Education and Science.

References

Abelló, A., Samos, J., & Saltor, F. (2001). A Framework for the Classification and Description of Multidimensional Data Models. In *Proceedings of the 12th Internatinal Conference on Database and Expert Systems Applications (DEXA'01)* (Vol. 2113, p. 668-677). Munich, Germany: Springer-Verlag.

Abelló, A., Samos, J., & Saltor, F. (2002). YAM2 (Yet Another Multidimensional Model): An Extension of UML. In *International Database Engineering & Applications Symposium (IDEAS'02)* (p. 172-181). Edmonton, Canada: IEEE Computer Society.

Ambler, S. (2002). *A UML Profile for Data Modeling.* Internet: http://www.agiledata.org/essays/umlDataModelingProfile.html.

Blaschka, M., Sapia, C., Höfling, G., & Dinter, B. (1998). Finding your way through multidimensional data models. In *Proceedings of the 9th International Conference on Database and Expert Systems Applications (DEXA'98)* (Vol. 1460, p. 198-203). Vienna, Austria: Springer-Verlag.

Giovinazzo, W. (2000). *Object-Oriented Data Warehouse Design. Building a star schema.* New Jersey, USA: Prentice-Hall.

Golfarelli, M., Maio, D., & Rizzi, S. (1998). The Dimensional Fact Model: A Conceptual Model for Data Warehouses. *International Journal of Cooperative Information Systems (IJCIS)*, *7*(2-3), 215-247.

Hüsemann, B., Lechtenbörger, J., & Vossen, G. (2000). Conceptual Data Warehouse Modeling. In *Proceedings of the 2nd International Workshop on Design and Management of Data Warehouses (DMDW'00)* (p. 6.1-6.11). Stockholm, Sweden.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process.* Addison-Wesley.

Jarke, M., Lenzerini, M., Vassiliou, Y., & Vassiliadis, P. (2003). *Fundamentals of Data Warehouses* (2 ed.). Springer-Verlag.

Kimball, R. (1996). *The Data Warehouse Toolkit.* John Wiley & Sons. ((Last edition: 2nd edition, John Wiley & Sons, 2002))

Kimball, R., Reeves, L., Ross, M., & Thornthwaite, W. (1998). *The Data Warehouse Lifecycle Toolkit.* John Wiley & Sons.

Luján-Mora, S., & Trujillo, J. (2003). A Comprehensive Method for Data Warehouse Design. In *Proceedings of the 5th International Workshop on Design and Management of Data Warehouses (DMDW'03)* (p. 1.1-1.14). Berlin, Germany.

Luján-Mora, S., & Trujillo, J. (2004a). A Data Warehouse Engineering Process. In *Proceedings of the 3rd Biennial International Conference on Advances in Information Systems (ADVIS'04)* (Vol. 3261, p. 14-23). Izmir, Turkey: Springer-Verlag.

Luján-Mora, S., & Trujillo, J. (2004b). Modeling the Physical Design of Data Warehouses from a UML Specification. In *Proceedings of the ACM Seventh International Workshop on Data Warehousing and OLAP (DOLAP 2004)* (p. 48-57). Washington D.C., USA: ACM.

Luján-Mora, S., Trujillo, J., & Song, I. (2002a). Extending UML for Multidimensional Modeling. In *Proceedings of the 5th International Conference on the Unified Modeling Language (UML'02)* (Vol. 2460, p. 290-304). Dresden, Germany: Springer-Verlag.

Luján-Mora, S., Trujillo, J., & Song, I. (2002b). Multidimensional Modeling with UML Package Diagrams. In *Proceedings of the 21st International Conference on Conceptual Modeling (ER'02)* (Vol. 2503, p. 199-213). Tampere, Finland: Springer-Verlag.

Luján-Mora, S., Vassiliadis, P., & Trujillo, J. (2004). Data Mapping Diagrams for Data Warehouse Design with UML. In *Proceedings of the 23rd International Conference on Conceptual Modeling (ER'04)* (Vol. 3288, p. 191-204). Shanghai, China: Springer-Verlag.

Naiburg, E., & Maksimchuk, R. (2001). *UML for Database Design.* Addison-Wesley.

Nicola, M., & Rizvi, H. (2003). Storage Layout and I/O Performance in Data Warehouses. In *Proceedings of the 5th International Workshop on Design and Management of Data Warehouses (DMDW'03)* (p. 7.1-7.9). Berlin, Germany.

Object Management Group (OMG). (2003, March). *Unified Modeling Language Specification 1.5.* Internet: http://www.omg.org/cgi-bin/doc?formal/03-03-01.

Poe, V., Klauer, P., & Brobst, S. (1998). *Building a Data Warehouse for Decision Support* (2 ed.). Prentice-Hall.

Rizzi, S. (2003). Open problems in data warehousing: eight years later. In *Proceedings of the 5th International Workshop on Design and Management of Data Warehouses (DMDW'03).* Berlin, Germany.

Sapia, C. (1999). On Modeling and Predicting Query Behavior in OLAP Systems. In *Proceedings of the 1st International Workshop on Design and Management of Data Warehouses (DMDW'99)* (p. 1-10). Heidelberg, Germany.

Sapia, C., Blaschka, M., Höfling, G., & Dinter, B. (1998). Extending the E/R Model for the Multidimensional Paradigm. In *Proceedings of the 1st International Workshop on Data Warehouse and Data Mining (DWDM'98)* (Vol. 1552, p. 105-116). Singapore: Springer-Verlag.

Selonen, P., Koskimies, K., & Sakkinen, M. (2003). Transformation Between UML Diagrams. *Journal of Database Management*, *14*(3), 37-55.

Silverston, L., Inmon, W., & Graziano, K. (1997). *The Data Model Resource Book: A Library of Logical Data Models and Data Warehouse Designs.* John Wiley & Sons.

Triantafillakis, A., Kanellis, P., & Martakos, D. (2004). Data Warehouse Interoperability for the Extended Enterprise. *Journal of Database Management*, *15*(3), 73-84.

Trujillo, J., & Luján-Mora, S. (2003). A UML Based Approach for Modeling ETL Processes in Data Warehouses. In *Proceedings of the 22nd International Conference on Conceptual Modeling (ER'03)* (Vol. 2813, p. 307-320). Chicago, USA: Springer-Verlag.

Trujillo, J., Palomar, M., Gómez, J., & Song, I. (2001). Designing Data Warehouses with OO Conceptual Models. *IEEE Computer, special issue on Data Warehouses*, *34*(12), 66-75.

Tryfona, N., Busborg, F., & Christiansen, J. (1999). starER: A Conceptual Model for Data Warehouse Design. In *Proceedings of the ACM 2nd International Workshop on Data Warehousing and OLAP (DOLAP'99)* (p. 3-8). Kansas City, USA: ACM.

Warmer, J., & Kleppe, A. (1998). *The Object Constraint Language. Precise Modeling with UML.* Addison-Wesley.

Whittle, J. (2000). Formal Approaches to Systems Analysis Using UML: An Overview. *Journal of Database Management*, *11*(4), 4-13.