Applying the UML and the Unified Process to the design of Data Warehouses

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

Abstract

The design, development and deployment of a data warehouse (DW) is a complex, time consuming and prone to fail task. This is mainly due to the different aspects taking part in a DW architecture such as data sources, processes responsible for Extracting, Transforming and Loading (ETL) data into the DW, the modeling of the DW itself, specifying data marts from the data warehouse or designing end user tools. In the last years, different models, methods and techniques have been proposed to provide partial solutions to cover the different aspects of a data warehouse. Nevertheless, none of these proposals addresses the whole development process of a data warehouse in an integrated and coherent manner providing the same notation for the modeling of the different parts of a DW.

In this paper, we propose a data warehouse development method, based on the Unified Modeling Language (UML) and the Unified Process (UP), which addresses the design and development of both the data warehouse back-stage and front-end. We use the extension mechanisms (stereotypes, tagged values and constraints) provided by the UML and we properly extend it in order to accurately model the different parts of a data warehouse (such as the modeling of the data sources, ETL processes or the modeling of the DW itself) by using the same notation . To the best of our knowledge, our proposal provides a seamless method for developing data warehouses. Finally, we apply our approach to a case study to show its benefit.

Keywords: data warehouse, design, UML, Unified Process

1 Introduction

In the early nineties, W. Inmon provided his famous definition of data warehouses (DW): *"A data warehouse is a subject-oriented, integrated, time-variant, nonvolatile collection of data in support of management's decisions"* [24]. A DW is subject oriented because it is focused on decision support instead of transactional processes and it is designed in order to speed up complex queries (normally using the multidimensional modeling or paradigm). It is “integrated” because data are gathered into the DW from a variety of sources (legacy systems, relational databases, COBOL files, etc.) and merged into a coherent whole. ETL (Extraction-Transformation-Loading) processes are responsible for the extraction of data from heterogeneous operational data sources, their transformation (conversion, cleaning, normalization, etc.) and their loading into DWs. Data are always loaded in a data warehouse together with a time reference as the final aim is to track history. Finally, a DW is normally considered a nonvolatile data repository because ETL processes are the responsible for inserting or updating data into the DW.

Most of the the research effort for designing data warehouses has been devoted to the multidimensional (MD) modeling or paradigm since it is widely accepted that data warehouses, multidimensional databases and On-Line Analytical Applications (OLAP) are based on the multidimensional modeling. The benefit of using this MD modeling is two-fold. On the one hand, the MD model is close to the way of thinking of data analyzers and, therefore, helps users[1] understand data. On the other hand, the MD model supports performance improvement as its simple structure allows us to predict end users intentions. MD modeling structures information into facts and dimensions. A fact contains interesting measures of a business process (sales, deliveries, etc.), whereas a dimension represents the context for analyzing a fact (product, customer, time, etc.).

Nevertheless, a data warehouse is much more than just multidimensional modeling. Building a data warehouse is normally a challenging and complex task because a data warehouse concerns many organizational units and often involves many people from database administrators or designers to non-expert end users. In the last years, there have been several proposals (see related work) to cover different aspects of a data warehouse such as the modeling of ETL processes or the design of the DW repository itself. However, although various methods and approaches have been presented for designing different parts of data warehouses, no general and standard method exists to date for dealing with the whole design of data warehouses considering all their main parts.

In the light of this situation, the goal of our work is to develop a data warehouse engineering process[2] to make the developing process of data warehouses more efficient. Our proposal is an object-oriented (OO) method, based on the Unified Modeling Language (UML) [5, 48, 44] and the Unified Process (UP) [26], which allows us to tackle all data warehouse design stages, from the operational data sources to the final implementation of the data warehouse and including

[1] We distinguish between *DW developers* (technical users, users who design and build the DW) and *DW end users* (users who are only interested in the business content or users who query the DW).

[2] We use the terms “method” and “process” as synonyms: “A software development process is the set of activities needed to transform a user’s requirements into a software system” [26].

the definition of the Extraction, Transformation, and Loading (ETL) processes and the end users' requirements.

The rest of the paper is structured as follows. In Section 2, we briefly present some of the most important related work and point out the main shortcomings. In Section 3, we summarize our data warehouse engineering process and in Section 3.1, we present the diagrams we propose to model the different parts of a data warehouse. In Section 4, we describe the different workflows that make up our process, and we provide short examples in order to make easier the understanding and application of our proposal. In Section 5 we present a more detail case study to show the benefit of our approach. Finally, we present the main conclusions and future work in Section 6.

2 Related Work

During the last few years, different approaches have been proposed for the design of DWs. It is out of the scope of this paper to make a reference to works related to the materialization of views, query processing or partitioning techniques, instead we will make a brief discussion about some of the well-known data warehouse models and methods. Some other methods that we cannot discuss due to the lack of space are [17, 13, 12].

To start with, it is highly recognized that the design and maintenance of these ETL processes is a key factor of success in DW projects for several reasons [52, 53]. Due to the high difficulty in designing and managing these ETL processes, there has lately been a proliferation in the number of available ETL tools that try to simplify this task [16, 29]. During 2001, the ETL market grew to about $667 million [3], although it is widely recognized that the design and maintenance of these ETL processes has not yet been solved [3]. To the best of our knowledge, the best advance in this research line has been accomplished by the Knowledge and Database Systems Laboratory from NTUA [42]. In particular, they have proposed a conceptual model that provides its own graphical notation that allows the designer to formally define most of the usual technical problems regarding ETL processes [58, 57]. Furthermore, this approach is accompanied by an ETL tool called ARKTOS as an easy framework for the design and maintenance of these ETL processes [59].

Lately, differen data models [20, 7, 56, 22, 55, 2, 28, 18, 9, 40], both conceptual and logical, have been proposed. These approaches are based on their own visual modeling languages or make use of a well-known graphical notation, such as the Entity-Relationship (ER) model or the UML. However, none of these approaches has been widely accepted as a standard DW model, because they present some important lacks (e.g. they do not address the whole DW process, they do not include a visual modeling language, they do not propose a clear set of steps or phases, or they are based on a specific logical model, such as ROLAP or MOLAP). Due to space constraints, we refer the reader to [1] for a detailed comparison and discussion about most of these models.

Regarding methods for building data warehouses, in [28], different case studies of data marts (DM) are presented. The MD modeling is based in the use of the *star schema* and its different variations (snowflake and fact constellation). Moreover, the BUS matrix architecture is proposed to build a corporate DW from integrating the design of several DMs. Although we consider this work as a fundamental reference in the MD field (R. Kimball provides a very sound discussion of star schema design), we miss a formal method for the design of DWs. Furthermore, the conceptual and logical models coincide in this proposal, and the concepts on the BUS matrix architecture are a compilation of the personal experiences of the authors and the problems they have faced building enterprise DWs from DMs.

In [19], the authors propose the *Dimensional-Fact Model* (DFM), a particular notation for the DW conceptual design. Moreover, they also propose how to derive a DW schema from the data sources described by Entity-Relationship (ER) schemas. From our point of view, this proposal is only oriented to the conceptual and logical design of DWs, because it does not consider important aspects such as the design of ETL processes. Furthermore, the authors assume a relational implementation of the DWs and the existence of all the ER schemas of the data sources, which unfortunately occurs in few occasions. Finally, we consider that the use of a particular notation makes difficult the application of this proposal.

In [7], the authors present the *Multidimensional Model* (MD), a logical model for OLAP systems, and show how it can be used in the design of MD databases. The authors also propose a general design method, aimed at building an MD schema starting from an operational database described by an ER schema. Although the design steps are described in a logic and coherent way, the DW design is only based on the operational data sources, what we consider insufficient because the end users' requirements are very important in the DW design.

In [40], the building of star schemas (and its different variations) from the conceptual schemas of the operational data sources is proposed. Once again, it is highly supposed that the data sources are defined by means of ER schemas. This approach differs from the above-presented ones in that it does not propose a particular graphical notation for the conceptual design of the DWs, instead it uses the ER graphical notation.

Most recently, in [9] another method for the DW design is proposed. This method is based on a MD model called IDEA and it proposes a set of steps to address the conceptual, logical, and physical design of a DW. One of the most important advantages, with respect to the previous proposals, is that the conceptual schema of the DW is built taking into consideration both the operational data sources and the end users' requirements. Nevertheless, this method only considers the data modeling and does not address other relevant aspects, such as the ETL processes.

In [8], different DW development methods are analysed and a new method is proposed. This method stands out because it integrates the management of metadata. However, it lacks a model that can be used to reflect and document the DW design.

Regarding the UML, some proposals to extend the UML for database design have been presented [46, 38, 41], since the UML does not explicitly include a data model. However, these proposals do not reflect the peculiarities of MD modeling. There have been some approaches that uses UML for the MD modeling up to date (again, and due to space constrains, we refer the reader to [1] for a comprehensive comparison). However, none of these approaches propose a comprehensive method to cover all main DW design phases.

Both W. Inmon and R. Kimball refer to two widely techniques used in real world projects for building a data warehouse: bottom-up or top-down. The former refers to the building of the DW from the data sources that will be used for populating the DW. The latter refers to designing the DW repository from the requirement analysis of end users and then, populate the DW from the data stored in data sources.

On the other hand, the Model Driven Architecture (MDA) [45] is an Object Management Group (OMG) standard that addresses the complete life cycle of designing, deploying, integrating, and managing applications. MDA separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. Thus, MDA encourages specifying a Platform Independent Model (PIM) by using any specification language -typically the Unified Modeling Language (UML). Then, this PIM can be transformed into multiple Platform Specific Models (PSM) in order to be executed on a concrete platform by transforming this PSM into the corresponding Code.

Following these MDA considerations, one incipient proposal of the OMG for aligning the design of DWs with the general MDA paradigm is called Model-Driven Data Warehouse (MDDW) [25]. This approach is based on the Common Warehouse Metamodel (CWM) [43], which is a platform-independent metamodel definition for interchanging DW specifications between different platforms and tools. Basically, the CWM - based on the standards UML, XMI (extensible Markup Interchange) and MOF (Meta Object Facility) - provides a set of metamodels that are comprehensive enough to model an entire DW including data sources, ETL processes, multidimensional modeling, relational implementation of a DW and so on. These metamodels are intended to be generic, external representations of shared metadata. The proposed MDDW is based on the CWM as the PIM for the DW design is accomplished by using the corresponding CWM metamodel (depending on the part of the DW to be modeled). Then, this PIM can be transformed into any PSM metamodel, and then, into the corresponding Code metamodel for a specific implementation. All of these transformations are metamodel transformations.

However, we differ in both the CWM and the MDDW approaches in which CWM metamodels are (i) too generic to represent all peculiarities of MD modeling [39] and (ii) too complex to be handled by both end users and designers. We deeply believe that it is more reliable to design a PIM by using an enriched conceptual modeling approach easy to be handled (e.g. our UML profile for multidimensional modeling) that can be totally represented by using the extended CWM metamodels, and then, any metamodel transformation is feasible, thus assuring the interchange of DW metadata between different tools and platforms.

In conclusion, no general and standard method exists to date for dealing with the whole design of a DW taking into consideration all different parts from data sources to the final implementation. Furthermore, we argue that this standard method should provide the same notation (a standard as possible) to deal with the design of the different parts.

3 Data Warehouse Development

The goal of our work is to develop a data warehouse engineering process [30] to make the developing process of data warehouses more efficient. In order to achieve this goal, we consider the following premises:

- Our method should be based on a standard visual modeling language.
- Our method should provide a clear and seamless method for developing a data warehouse.
- Our method should tackle all data warehouse design stages in an integrated manner, from the operational data sources to the final implementation and including the definition of the ETL processes and the end users' requirements.
- Our method should provide different levels of detail.

Therefore, we have selected the UML as the visual modeling language, our method is based on the well-accepted UP, we have extended the UML in order to accurately represent the different parts of a data warehouse, and we extensively use the UML packages with the aim of providing different levels of detail.

Although the UML and the UP unify many years of effort in object oriented analysis and design, the UML and the UP present some issues and shortcomings. For example, some critics argue UML is complex and difficult to learn and it lacks a method to guide its users. Moreover, "There are different views about how to use UML to develop software systems" [49]. The UML is a complex modeling language and it lacks a systematic way of guiding the users through the development of systems. For example, with respect to *packages*, the UML defines them as a mechanism to simplify complex UML diagrams. However, no formal design guidelines regarding how to properly use them are clearly stated. In this context, many text books and authors have provided guidelines to apply packages in general or in a specific domain. For example, Fowler states [15]: "I use the term package diagram for a diagram that shows packages of classes and the dependencies among them". In a specific domain such as web applications, Conallen states [11] that "A package is merely a mechanism to divide the model into more manageable pieces" and "Try to avoid making the package hierarchy match the semantics of the business, and instead use packages as a means of managing the model". The author proposes to make packages "*comprehensible*, *cohesive*, *loosely coupled* and *hierarchically shallow*".

Despite these drawbacks, our proposal is based on the UML and the UP because:

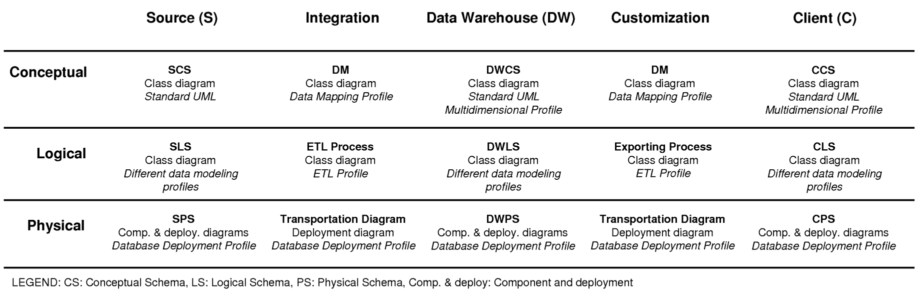
- UML can address many dimensional modeling issues that were impossible to solve with the entity relationship diagram (ERD) [13].
- The theoretical complexity of the UML does not necessarily relate to its practical complexity [50].
- UML provides advanced visualization and modeling capabilities.
- UML allows us to describe the interactions of the data warehouse project team.
- UML provides extensibility and specialization mechanisms to extend it.
- UP integrates current "software development best practices".

In the next sub-section, we present the set of diagramas (based on the UML) that can be used within our method to design data warehouses.

3.1 Data Warehouse Diagrams

The architecture of a data warehouse is usually depicted as various layers of data in which data from one layer is derived from data of the previous layer [27]. Following this consideration, we consider that the development of a data warehouse can be structured into an integrated framework with five stages and three levels that define different diagrams for the data warehouse model, as shown in Figure 1:

- **Stages**: we distinguish five stages in the definition of a data warehouse:
  - Source, which defines the structure of the operational data sources of the data warehouse, such as On-Line Transaction Processing (OLTP) systems, external data sources (syndicated data, census data), etc.
  - Integration, which defines the mapping between the data sources and the data warehouse.
  - Data Warehouse, which defines the structure of the data warehouse.
  - Customization, which defines the mapping between the data warehouse and the clients' structures.
  - Client, which defines special structures that are used by the clients to access the data warehouse, such as data marts or On-Line Analytical Processing (OLAP) applications.
- **Levels**: each stage can be analyzed at three levels or perspectives:
  - Conceptual: it defines the data warehouse from a conceptual point of view.



	Source (S)	Integration	Data Warehouse (DW)	Customization	Client (C)
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

- Logical: it addresses logical aspects of the data warehouse 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 require 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 the UML [44] as the modeling language, because it provides enough expressiveness power to address all the diagrams. As the UML is a general modeling language, we can use the UML extension mechanisms (stereotypes, tag definitions, and constraints) to adapt the UML to specific domains. In Figure 1, we provide 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 data warehouse diagram. Currently, we use class, deployment, and component diagrams.
  - Profile (*in italic face*): we show the diagrams where we propose a new profile; in the other cases, we use a standard UML diagram or a profile from other authors.

The best advantage of our global approach is that we always use the same notation (based on UML) for designing the different data warehouse schemas and the corresponding transformations in an integrated manner. Moreover, the different diagrams of the same data warehouse are not independent but overlapping: they depend on each other in many ways. For example, changes in

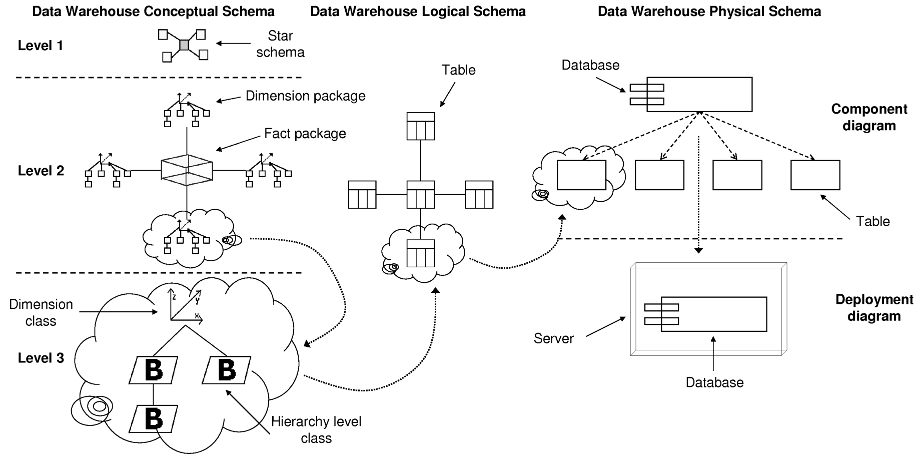


Figure 2: From the conceptual to the physical level

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 DM is created by importing elements from the SCS and the DWCS.

In previous works, we have presented the different diagrams and the corresponding profiles we propose as follows:

- *Multidimensional Profile*, for the Data Warehouse Conceptual Schema (DWCS) and the Client Conceptual Schema (CCS), in [34, 35].
- *Data Mapping Profile*, for the Data Mapping (DM) between the Source Conceptual Schema (SCS) and the DWCS, and between the DWCS and the CCS, in [36].
- *ETL Profile*, for the ETL Process between the Source Logical Schema (SLS) and the Data Warehouse Logical Schema (DWLS), and the Exporting Process between the DWLS and the Client Logical Schema (CLS), in [54].
- *Database Deployment Profile*, for the Source Physical Schema (SPS), the Transportation Diagram, the Data Warehouse Physical Schema (DWPS), and the Client Physical Schema (CPS), in [31, 32].

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

- 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.

4 Data Warehouse Engineering Process

Our method, called *Data Warehouse Engineering Process* (DWEP), is based on the Unified Software Development Process, also known as UP [26]. The UP is an industry standard Software Engineering Process (SEP) from the authors of the UML. Whereas the UML defines a visual modeling language, the UP specifies how to develop software using the UML.

The UP is a generic SEP that has to be instantiated[7] for an organization, project or domain[8]. DWEP is our instantiation of the UP for the development of data warehouses. Some characteristics of our DWEP inherited from UP are: use case (requirement) driven, architecture centric, iterative and incremental. These three key words make the UP unique, therefore it is important to provide a brief overview of them:

- Use case (requirement) driven: means that use cases are used for specifying the requirements of a system, but they also drive its design, implementation, and test.

[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.

[7] Instantiate means add in-house standards, define a lifecycle strategy, select what diagrams to use, define activities and workers, etc.

[8] Some popular instantiations of UP are Rational Unified Process [23] and Enterprise Unified Process [47].

- Architecture centric: the software architecture embodies the most significant static and dynamic aspects of the system and it is described as different views of the system being built.
- Iterative and incremental: the develop of the software products is divided into smaller slices called iterations that results in an increment that refers to growth in the product. Moreover, the diagrams will not stay intact but should be expected to evolve as time passes, as new requirements are uncovered, and as the schedules changes, causing feature changes.

According to the UP, the project lifecycle is divided into four phases (Inception, Elaboration, Construction, and Transition) and five core workflows (Requirements, Analysis, Design, Implementation, and Test). Each phase ends with the delivery of a *product* to the customer:

- Inception ends with commitment to go ahead with the project and a timetable to develop and deploy the DW. This phase involves capturing essential end users needs and outline the architecture of the DW. Therefore, the emphasis is on requirements.
- Elaboration ends with the basic architecture of the system, a timetable to construct the DW, a plan of work, a list of risks, and a first version of the DW.
- Construction ends with a first operating version of the DW. Therefore, the emphasis is on the implementation.
- Transition ends with end users using the DW. The goals of this phase are: correct defects, prepare the user site for the new DW, create user manuals and other documentation.

The meaning of the five core workflows is [4]:

- Requirements: capturing what the system should do.
- Analysis: refining and structuring the requirements.
- Design: realizing the requirements in system architecture.
- Implementation: building the software.
- Test: verifying that the implementation works as desired.

Moreover, we have added two more workflows to the five core workflows: *Maintenance* and *Post-development review*. During the developing of a project, the emphasis shifts over the iterations, from requirements and analysis towards design, implementation, testing, and finally, maintenance and post-development review, but different workflows can coexist in the same iteration.

In Figure 3 (adapted from [26]), we show that the seven workflows (listed in the left-hand column) take place over the four phases. For each workflow, the

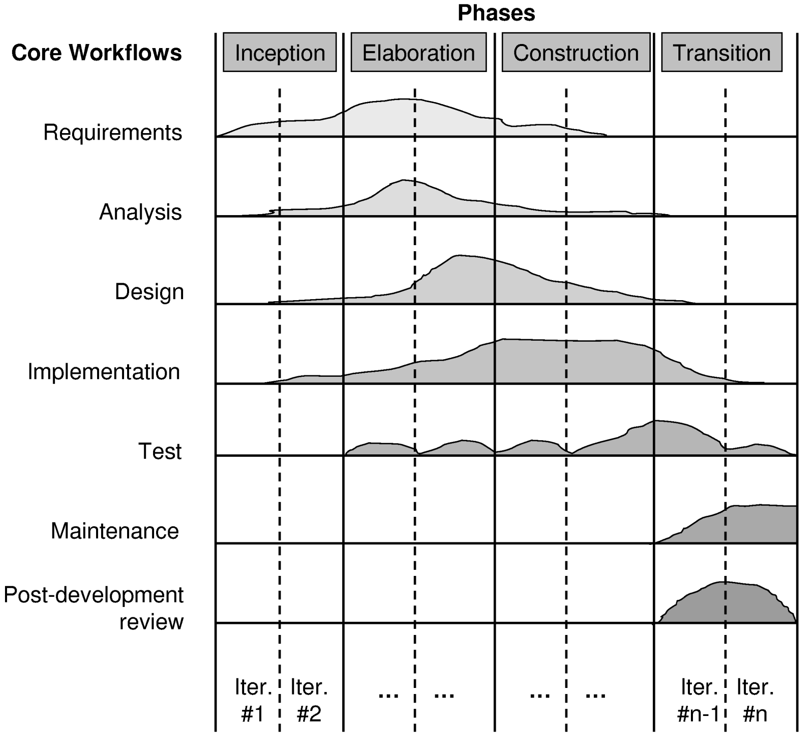


Figure 3: DWEP workflows

curve represents approximately the extent to which the workflow is carried out in each phase. Moreover, each phase is usually subdivided into iterations, and an iteration goes through all the seven workflows.

The development of a DW is divided into a number of small steps (milestones) which are easier to understand, to manage and to complete successfully. Each of these steps is an iteration.

For each one of the workflows, we use different UML diagrams (techniques) to model and document the development process, but a model can be modified in different phases because models evolve over time. In Table 1, we show a summary of all the diagrams introduced in Section 3.1 and their usages in different workflows. In the following sections, we present the main details of the workflows and highlight the diagrams we use in each workflow.

4.1 Requirements

A key factor in the success of a DW are the needs of end users, because DWs are very unique to an organization. During the Requirements workflow, what the

DWEP workflow	**Diagram**
Requirements	Use cases Activity Statechart
Analysis	Source Conceptual Schema Source Logical Schema Source Physical Schema
Design	Date Warehouse Conceptual Schema Client Conceptual Schema Data Mapping
Implementation	Data Warehouse Logical Schema Data Warehouse Physical Schema Client Logical Schema Client Physical Schema ETL Process Exportation Process Transportation
Test	None
Maintenance	None
Post-development review	None

Table 1: Summary of all the diagrams and their usages in different workflows

end users expect to do with the data warehouse is captured: the end users should specify the most interesting measures and aggregations, the analysis dimensions, the queries used to generate periodical reports, the update frequency of the data, etc. As proposed in [6], we model the requirements with use cases. The rationale of use cases is that focusing "on what the users need to do with the system is much more powerful than other traditional elicitation approaches of asking users what they want the system to do" [6]. Once the requirements have been defined, the data warehouse project is established and the different roles are designated.

Use case modeling is a simple way to [37]:

- Understand the system's existing functions.
- Elicit the desired requirements and functions for the new system is being created.
- Establish who will be interacting with the system and how.

The UML provides the use case diagram for visual modeling of uses cases. Use case diagrams identify the functionality provided by a system, the users who interact with the system, and the association between the users and the functionality. Nevertheless, there is no UML standard for a use case specification. However, we follow the common template defined in [4], which specifies for every use case a name, a unique identifier, the actor involved in the use case, the system state before the use can begin (*preconditions*), the actual steps of the use case (*flow of events*), and the system state when the use case is over (*postconditions*). In [10, 41], different use case description templates with a great variety of styles can be found.

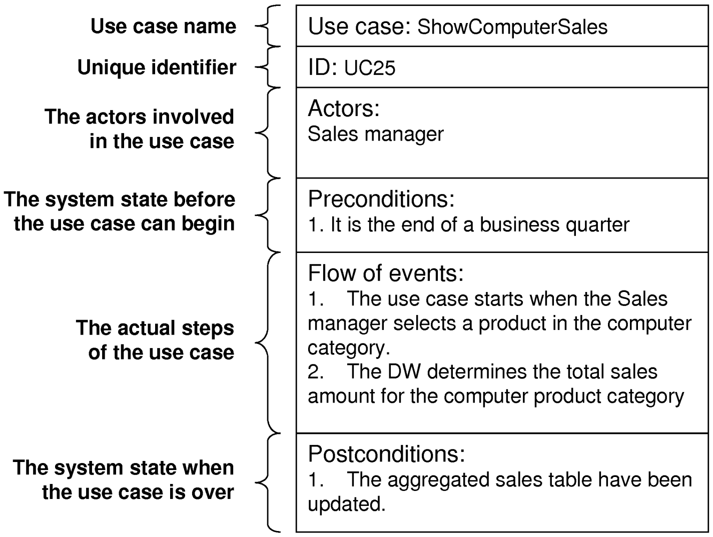
For example, the use case in Figure 4 is about sales managers making a query about the quarterly sales of the products in the computer category.

Moreover, we also apply UML statechart and activity diagrams to show the states and the flow of a use case. Both diagrams allow us to understand more deeply how use cases are accomplished. A statechart diagram represents the behavior of entities of a system by specifying its response to the receipt of events; an activity diagram shows a sequence of activities, their ordering, and decisions that are made to indicate which activity performs next.

4.2 Analysis

The goal of this workflow is to refine and structure the requirements output in the previous workflow. Moreover, the pre-existing operational systems that will feed the data warehouse are also documented: the different candidate data sources are identified, the data content is revised, etc.

We use the Source Conceptual Schema, Source Logical Schema, and the Source Physical Schema (SCS, SLC, and SPS) (Figure 1) to model the data sources at different levels of detail. To get quality data in the data warehouse, the different data sources must be well identified.


Use case name
Unique identifier
The actors involved in the use case
The system state before the use case can begin
The actual steps of the use case
The system state when the use case is over

Use case: ShowComputerSales

ID: UC25

Actors:
Sales manager

Preconditions:
1. It is the end of a business quarter

Flow of events:
1. The use case starts when the Sales manager selects a product in the computer category.
2. The DW determines the total sales amount for the computer product category

Postconditions:
1. The aggregated sales table have been updated.

Figure 4: UML use case template

For example, in Figure 5 we show the Source Logical Schema (SLS) of a transactional system that manages the sales of a company. This system will feed with data the data warehouse that will be defined in the following workflow. For the SLS we make use of the *UML Profile for Database Design* [41] that defines a series of stereotypes like «Database», «Schema», «Tablespace», or «Table». In the diagram shown in Figure 5, each element represents a class with the stereotype «Table» that represents a table in a relational database[9]; we have hidden the attributes (fields or columns) for the sake of simplicity.

4.3 Design

At the end of this workflow, the structure of the data warehouse is defined. The main output of this workflow is the conceptual schema of the data warehouse. Moreover, the source to target data map is also developed at a conceptual level.

In this workflow, the main diagrams are the Data Warehouse Conceptual Schema (DWCS), the Client Conceptual Schema (CCS), and the Data Mapping (DM). The DM shows the relationships between the SCS and the DWCS and between the DWCS and the CCS.

For the DWCS and the CCS, we have presented in [34, 35] an extension of the UML by means of a UML profile. This profile is defined by a set of stereotypes and tagged values to elegantly represent main multidimensional properties at

[9] A class symbol with a stereotype icon may be "collapsed" to show just the stereotype icon, with the name of the class either inside the class or below the icon. Other contents of the class may be suppressed.

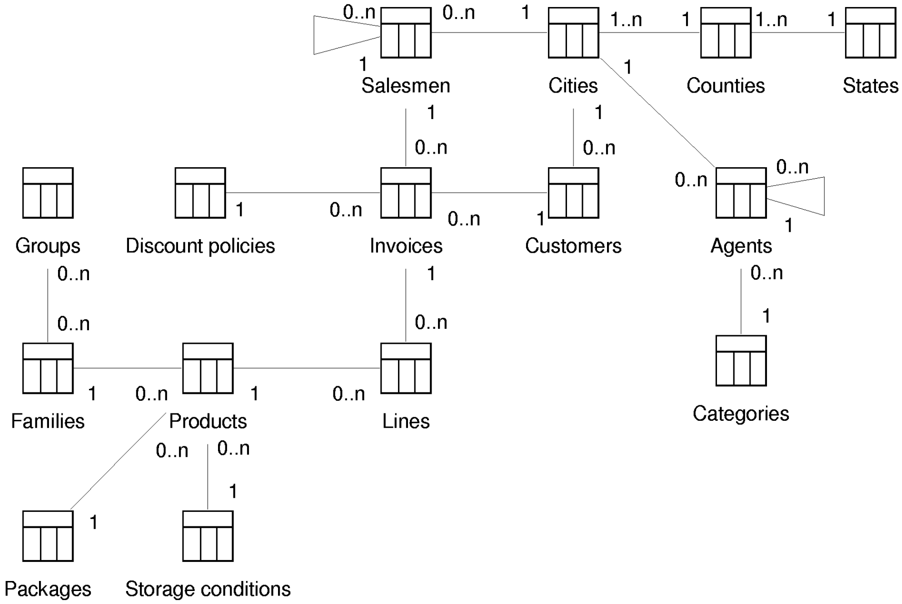


Figure 5: Source Logical Schema

the conceptual level. We make use of the Object Constraint Language (OCL) to specify the constraints attached to the defined stereotypes, thereby avoiding an arbitrary use of these stereotypes. The main advantage of our proposal is that it is based on a well-known standard modeling language (UML), thereby designers can avoid learning a new specific notation or language for multidimensional systems.

For example, in Figure 6 we show level 1 of a Data Warehouse Conceptual Schema, composed of three schemas (Production schema, Sales schema, and Salesmen schema). The dashed arrows that connect the different schemas indicate that the schemas share some dimensions (*common dimensions*) that have been firstly defined in the Sales schema.

In Figure 7 we show level 2 of the Sales schema from level 1, composed of one fact (Sales fact) and four dimensions (Stores dimension, Times dimension, Products dimension, and Customers dimension). In Figure 8, the definition of the Customers dimension with the different hierarchy levels is showed and finally, in Figure 9 the whole star schema with the fact Sales and the four dimensions is shown. We do not represent the hierarchy levels of the dimensions Stores dimension and Times dimension for the sake of simplicity.

For the DM, we have presented [36] the *Data Mapping Profile* that introduces the data mapping diagram. In this new diagram, we treat attributes as first-class modeling elements of the model. In this way, attributes can partici-

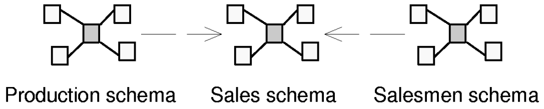


Figure 6: Data Warehouse Conceptual Schema (level 1)

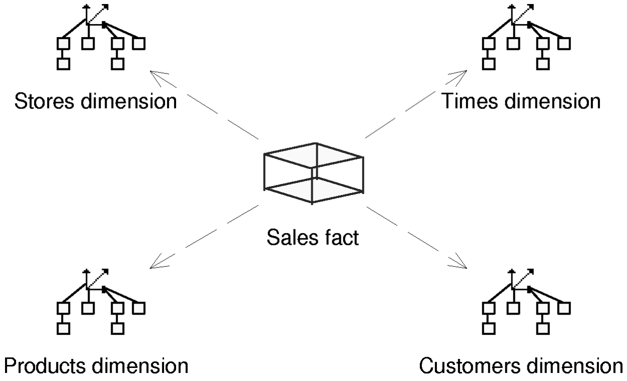


Figure 7: Data Warehouse Conceptual Schema (level 2)

pate in associations that determine the inter-attribute mapping, along with any necessary transformation and constraints. Since a data mapping diagram can be very complex, our approach offers the possibility to organize it in different levels thanks to the use of UML packages.

In the DW context, when two design elements (e.g., two tables, or two attributes) share the same piece of information, possibly through some kind of filtering or transformation, this constitutes a semantic relationship between them. This relationship, involves three logical parties: (a) the provider entity (schema, table, or attribute), responsible for generating the data to be further propagated, (b) the consumer, that receives the data from the provider and (c) their intermediate matching that involves the way the mapping is done, along with any transformation and filtering.

In Figure 10, we show a data mapping between Products from Data sources and Products dim. In this data mapping, we have defined a transformation between the Products.Price attribute and the ProdDescription.Price attribute; this transformation converts the price of products from dollars to euros. The transformation is defined inside a UML note attached to the mapping relationship.

4.4 Implementation

During this workflow, the data warehouse is built: the physical data warehouse structures are built, the data warehouse is populated with data from heteroge-

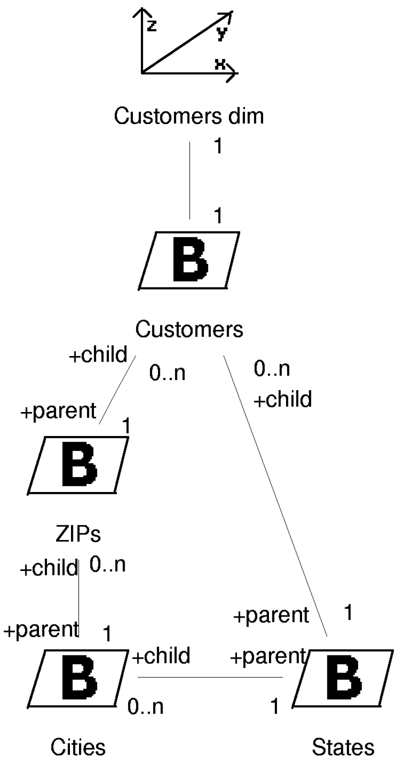


Figure 8: Data Warehouse Conceptual Schema (level 3, Customers dimension)

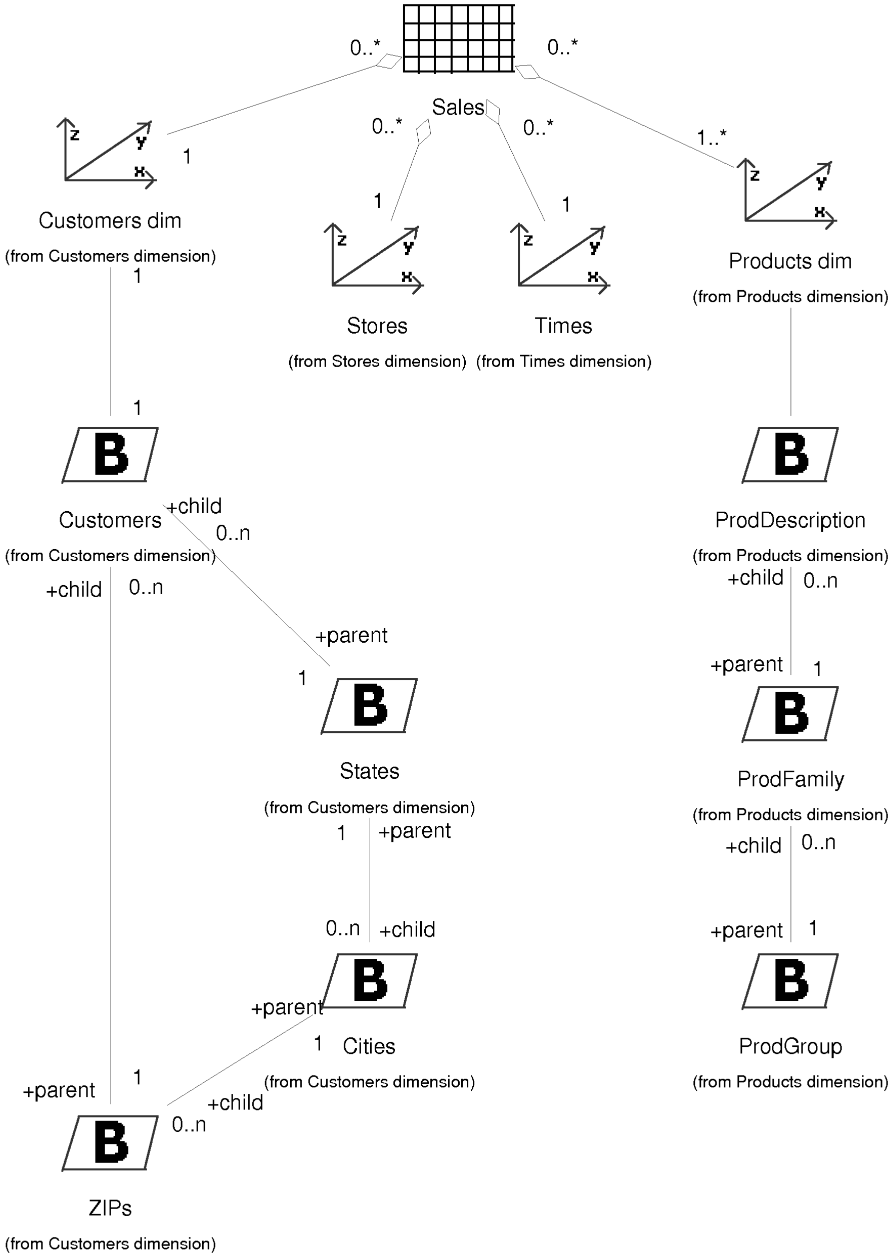


Figure 9: Data Warehouse Conceptual Schema (level 3, Sales fact)

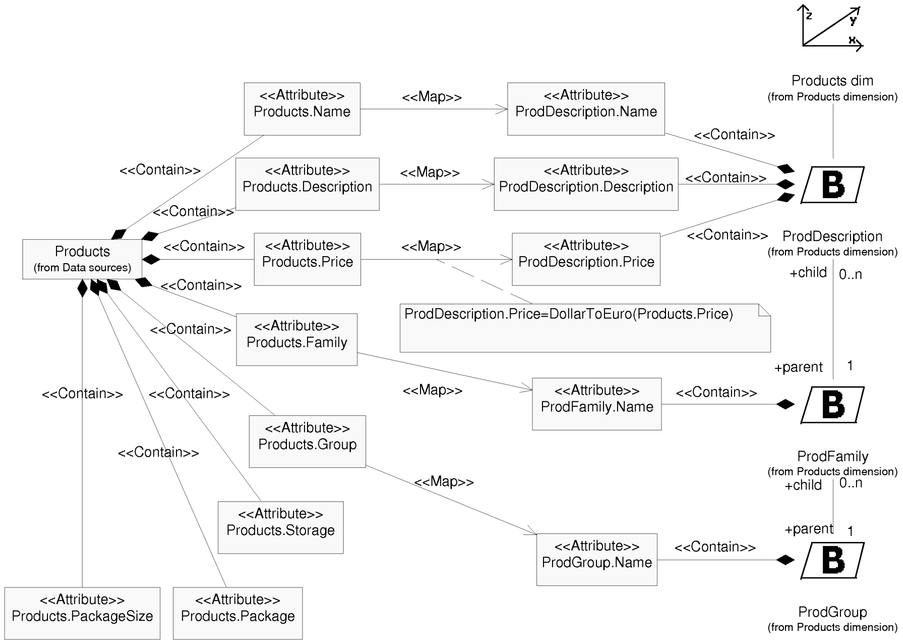


Figure 10: Data Mapping between Products and Products dim

nous data sources, the data warehouse is tuned for an optimized running, etc. Different implementation diagrams can be created to help this workflow.

The main diagrams in this workflow are the Data Warehouse Logical Schema, the Data Warehouse Physical Schema, the Client Logical Schema, the Client Physical Schema, the ETL Process, the Exportation Process, and the Transportation Diagram.

In the ETL Process, the cleansing and quality control activities are modelled. We provide the necessary mechanisms for an easy and quick specification of the common operations defined in these ETL processes such as, the integration of different data sources, the transformation between source and target attributes, the generation of surrogate keys and so on. The design of an ETL process is usually composed of six tasks [54]:

1. Select the sources for extraction: the data sources to be used in the ETL process are defined. It is very common in an ETL process to access different heterogeneous data sources.

2. Transform the sources: once the data have been extracted from the data sources, they can be transformed or new data can be derived. Some of the common tasks of this step are: filtering data, converting codes, performing table lookups, calculating derived values, transforming between different data formats, automatic generation of sequence numbers (surrogate keys), etc.

3. Join the sources: different sources can be joined in order to load together the data in a unique target.

4. Select the target to load: the target (or targets) to be loaded is selected.

5. Map source attributes to target attributes: the attributes (fields) to be extracted from the data sources are mapped to the corresponding target attributes.

6. Load the data: the target is populated with the transformed data from the sources.

For example, in Figure 11 we show the ETL process that loads Customers dimension. We use three Join mechanisms (Cus-Cities, Cus-Cities-Counties, CompleteCustomers) to join the different data sources and we apply two Conversion mechanisms (CleanCustomers, CleanStates) to change data types and formats or to calculate and derive new data from existing data.

Regarding our proposal for the modeling of the physical design of DWs, we use the component and deployment diagrams of the 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, the use of particular servers for certain ETL processes, the estimation of the workload needed and the time boundaries to accomplish it, and so on.

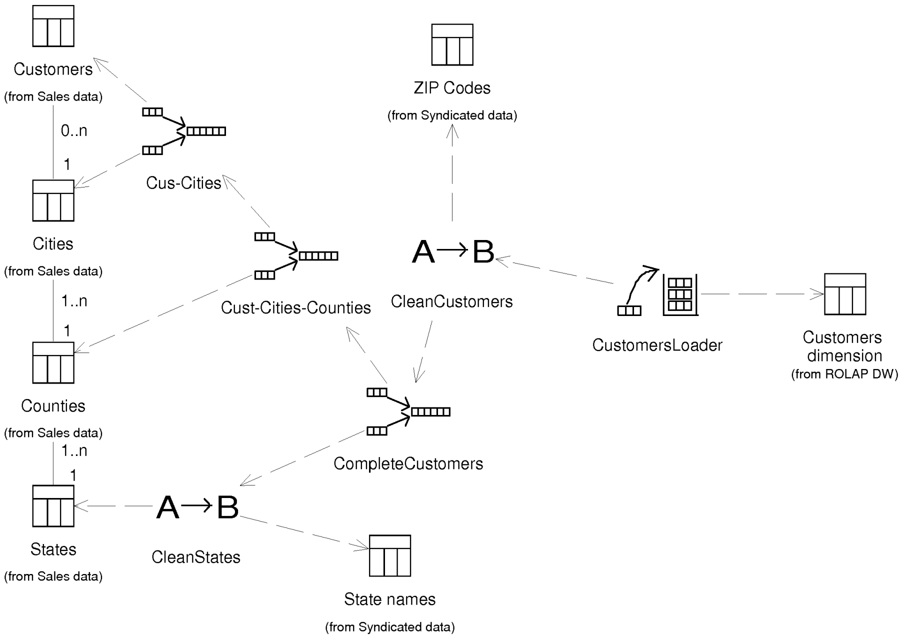


Figure 11: ETL Process

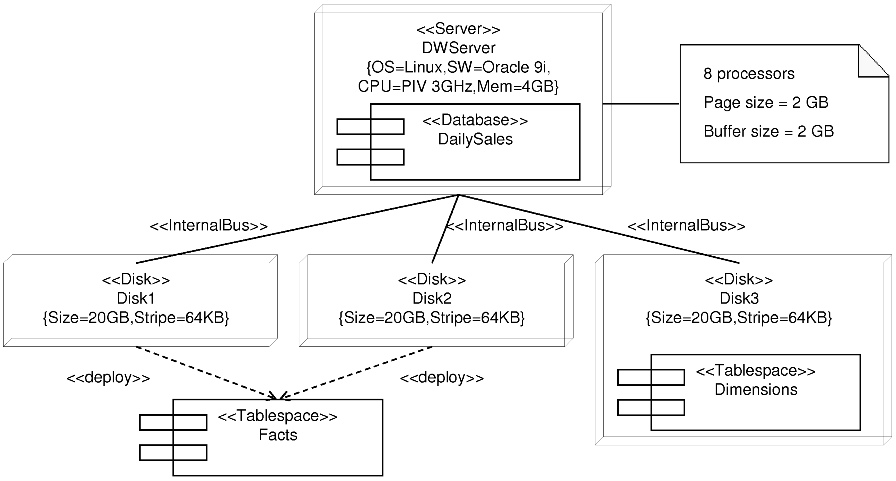


Figure 12: Data Warehouse Physical Schema

For example, in Figure 12 we show part of a Data Warehouse Physical Schema. For this diagram, we have presented the *Database Deployment Profile* [32, 33]. In this example, both the components and the nodes are stereotyped: the components are adorned with the «Database» and «Tablespace» stereotypes, and the nodes with the «Server» and «Disk» stereotypes.

4.5 Test

The goal of this workflow is to verify that the implementation works as desired. More specifically, the purposes of testing are to:

- Plan the tests required. Different tests can be used: unit tests, system tests, integration tests, acceptance tests, and scenario tests.
- Design and implement the tests by creating test cases.
- Perform the tests and analyze the results of each test.

No new diagrams are created, but previous diagrams (mainly design and implementation diagrams) may be modified according the corrective actions that are taken.

4.6 Maintenance

Unlike most systems, the data warehouse is never done. The goal of this workflow is to define the refresh and loading processes needed for keeping the data warehouse up to date. This workflow starts when the data warehouse is built and delivered to the end users, but it does not have an end date (it lasts during the life of the data warehouse).

During this workflow, the end users can state new requirements, such as new queries, which triggers the beginning of a new iteration (UP is an iterative process) with the Requirements workflow.

4.7 Post-development Review

This is not a workflow of the development effort, but a review process for improving future projects. We look back at the development of the data warehouse, revise the documentation created, and try to identify both opportunities for improvement and major successes that should be taken into account. If we keep track of the time and effort spent on each phase, this information can be useful in estimating time and staff requirements for future projects.

4.8 Top-down or Bottom-up?

Nowadays, there are two basic strategies in the building of a data warehouse: the top-down and bottom-up approaches [21, 14, 60]. The top-down approach recommends the construction of a data warehouse first and then the construction of data marts from the parent data warehouse. The bottom-up approach uses a series of incremental data marts that are finally integrated to build the goal of the data warehouse. Each approach has its own set of strengths and weaknesses. However, in almost all projects, the data marts are built rather independently without the construction of an integrated data warehouse, which is indeed viewed no more as a monolithic repository but rather as a collection of data marts.

In the early years, the top-down approach was favored and it is considered the most elegant design approach [21]. However, high rates of failure for initial data warehouse projects have led the majority of current projects to the bottom-up approach. With the bottom-up approach, the results are seen considerable sooner than implementing the data warehouse using a top-down approach. Moreover, risk is minimized and it is more feasible to successfully deploy the data mart in time.

Our method also allows both approaches. In the top-down approach (see Figure 13), the data warehouse is built first and the data sources are the transactional systems; then, each data mart is built independently by using our method, and the data warehouse becomes the only data source for all of them. Nevertheless, in the bottom-up approach (see Figure 14), the data marts are built first from the transactional systems; then, the data warehouse is built and the data sources are the data marts.

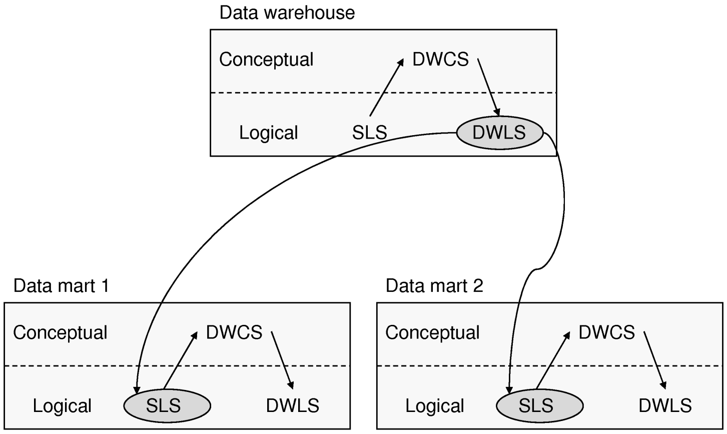


Figure 13: Top-down approach

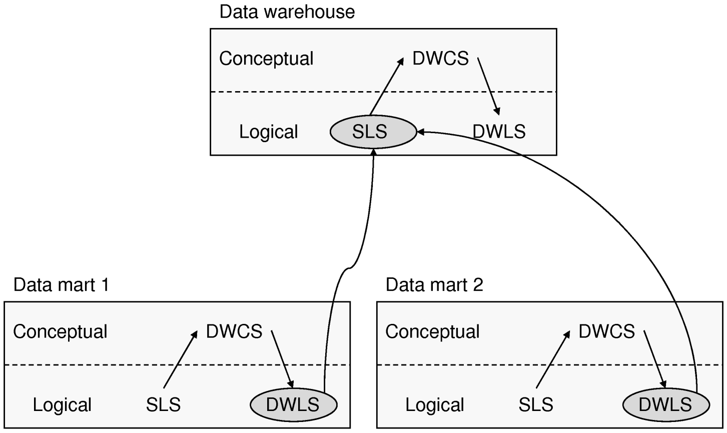


Figure 14: Bottom-up approach

5 A Case Study

Throughout the rest of this paper, we use a case study to show how to apply our Data Warehouse Engineering Process (DWEP) and how the different diagrams we proposed can be applied to real world projects. This case study is partly based on the enterprise DW sample database schema from [51]. In this example, end users need a DW in order to analyze the main operations of the company. Thus, 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.

5.1 Requirements

The first step is to define the scope of the DW. Basically, end users indicate reports and queries they may need and the DW administrators define all the tasks necessary to manage the DW.

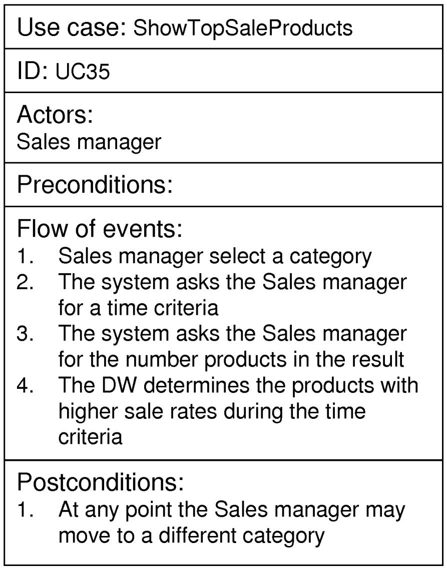
Requirements can be basically classified into functional and nonfunctional requirements. Following [26], we use UML use cases, activity diagrams, and statechart diagrams to represent functional requirements, whereas nonfunctional requirements, that are common for many use cases, are defined in a separate document. The overall requirements definition of a DW is essential to ensure we understand the need of end users.

In this section, we model some examples of functional and nonfunctional requirements for our case study. From interviews to the end users, we have detected two functional requirements related to sales managers, so the resulting DW will be able to achieve these requirements:

- A sales manager needs to analyze the top sales rates through a time period regarding the category of the products.
- A sales manager needs to display sales per customer, per region, per location, per product, per time period, or any combination of the above; optionally, a report may be generated

These two functional requirements influence how sales managers query the DW and they are useful to define the dimension hierarchies of the DW.

Regarding the first functional requirement, we show the corresponding use case in Figure 15, following the use case description template we have introduced in Section 4.1. This use case defines the actor (Sales manager), the preconditions



Use case: ShowTopSaleProducts

ID: UC35

Actors:
Sales manager

Preconditions:

Flow of events:

1. Sales manager select a category
2. The system asks the Sales manager for a time criteria
3. The system asks the Sales manager for the number products in the result
4. The DW determines the products with higher sale rates during the time criteria

Postconditions:

1. At any point the Sales manager may move to a different category

Figure 15: Use case definition for the first requirement

(none), the flow of events (sales managers making a query about the top sale products in a time period), and the postconditions after performing this use case.

Use cases can not be used to show the inside small processes in modeling. For this activity diagrams and statechart diagrams are used. Activity diagrams are useful for analyzing a use case by describing what actions need to take place and when they should occur. In Figure 16, we show an activity diagram for the use case shown in Figure 15.

Regarding the second functional requirement, we show the corresponding use case diagram in Figure 17. This use case diagram shows the different criteria that sales managers will use when consulting the DW.

The nonfunctional requirements can not be associated with any particular use case, but these requirements have impact on different use cases or none at all. Examples are performance and physical requirements or design and implementation constraints. They are usually captured as a list of requirements without the help of any visual artifact.

Regarding our case study, we have detected two nonfunctional requirements. The first requirement specifies that products and customers invoices have to be partitioned into two different partitions, once for products in year 2004, and another one for new products in 2005. This requirement will be used in Section 5.4 in the definition of the tablespaces and partitions of the DW.

The second requirement specifies what the users have defined about the hardware and software requirements for the DW server and the computers that

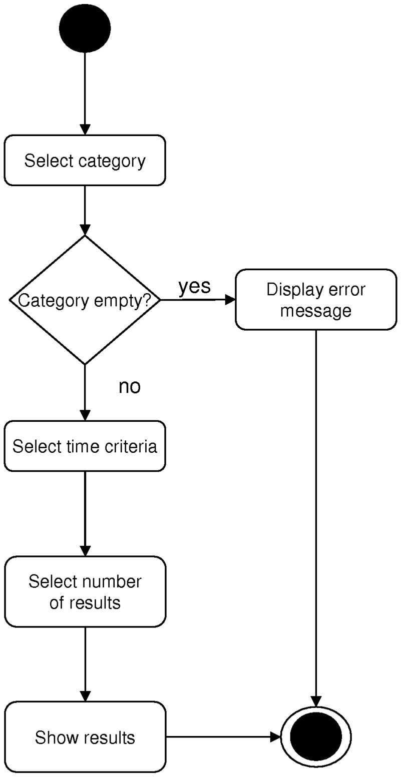


Figure 16: Activity diagram of the use case definition

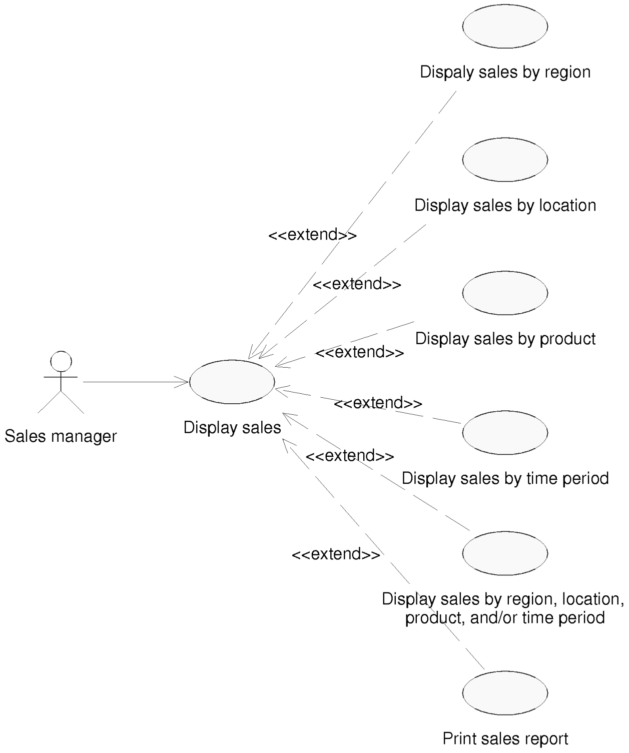


Figure 17: Use case diagram for the second requirement

end users will use to access the DW:

```
Server
------
Hardware: PC Pentium IV 3 GHz, 4 GB
Software: Linux, Oracle 9i

Clients
-------
** Macintosh
Hardware: PowerPC G4 800 MHz, 256 MB
Software: MacOS, Safari

** Windows (web)
Hardware: PIII 1,25 GHz, 256 MB
Software: Windows, Internet Explorer

** Windows (desktop)
Hardware: PIV 2 GHz, 512 MB
Software: Windows, MicroStrategy
```

This second nonfunctional requirement will be used in Section 5.4 in the definition of the CTD.

5.2 Analysis

The goal of the SCS is to know what data is available for the DW. For the SCS, we apply UML in a plain style by simply using classes, attributes and their associations to other entities.

The operational data sources are stored in three servers: the sales server, the CRM server, and the HRM server. From now on, we will focus on the sales saver, which contains the data about transactions and sales.

In Figure 18, we show the SCS of the sales server. In this diagram, we capture the most important concepts in the context of the system that we are modeling. For the sake of clearness, we have omitted some attributes.

To achieve the logical modeling of the data sources (SLS), we apply the *UML Profile for Database Design* [41]. In Figure 19, we show the logical model of the relational database of the sales server. The attributes of the tables of the database schema are not shown in order to clarify this diagram. This database schema is composed of six tables (Product, Catalog, Category, Family, SalesLines, and Sales) and two views (Customers and Salesperson). The views indicate data stored in other servers (Customers in CRM server and Salesperson in HRM Server).

Finally, the SPS describes the origins of data of the DW from a physical point of view. Figure 20 shows the SPS of our example, which is formed by three servers called SalesServer, CRMServer, and HRMServer; for each one of

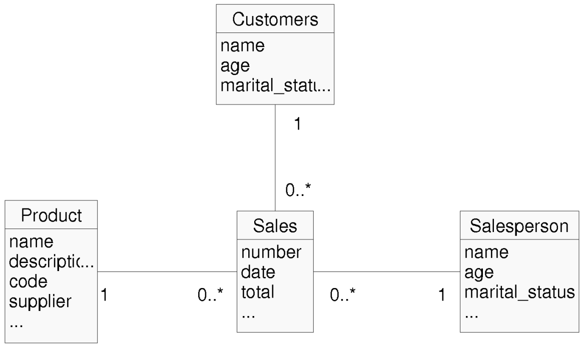


Figure 18: Source Conceptual Schema

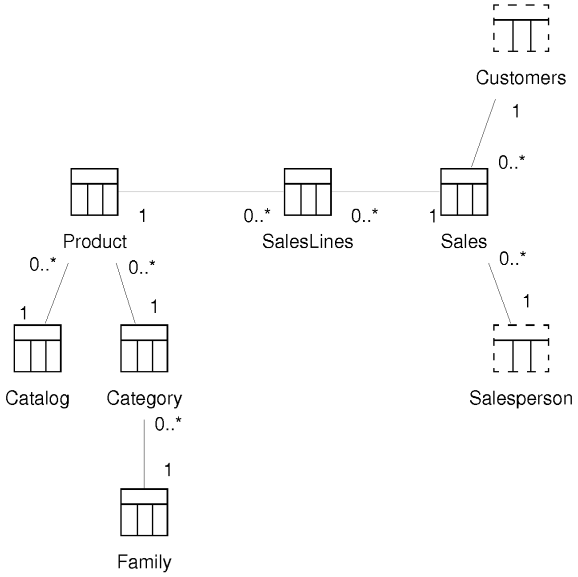


Figure 19: Source Logical Schema

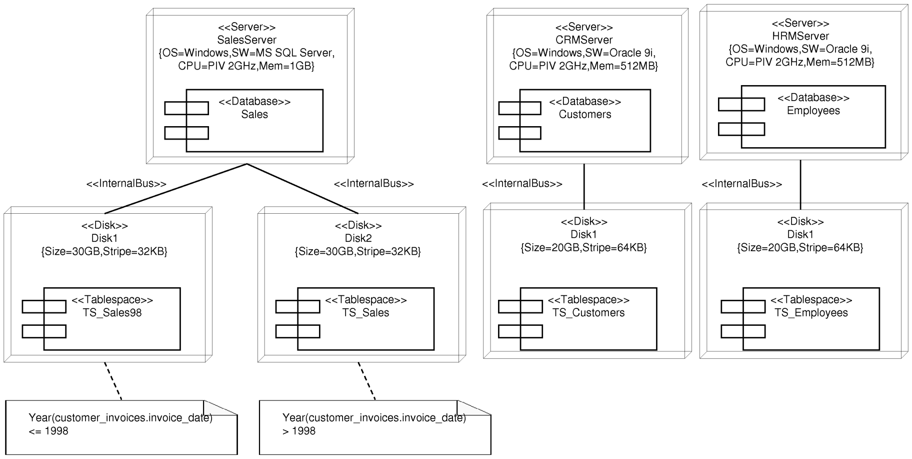


Figure 20: Source Physical Schema: deployment diagram

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.

5.3 Design

Following our approach [35], 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 21 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

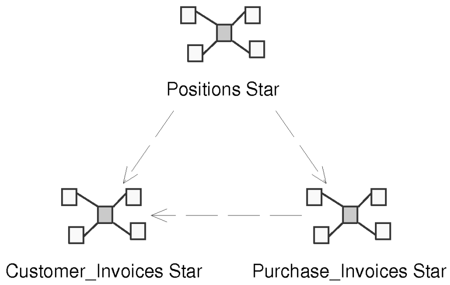


Figure 21: Data Warehouse Conceptual Schema: Level 1

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 22 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 22, 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 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 23 shows the level 3 of the Customers dimension package (Figure 22), which contains the definition of the dimension (Customers) and the different hierarchy levels (Customer_Addresses, Customer_SED, GB_City, Postal_Code, GB_City[10], 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 [28].

Figure 24 shows the level 3 of the Products dimension. This dimension contains two alternative hierarchies: the category of the product (Category) and

[10] GB means *geographic boundaries*.

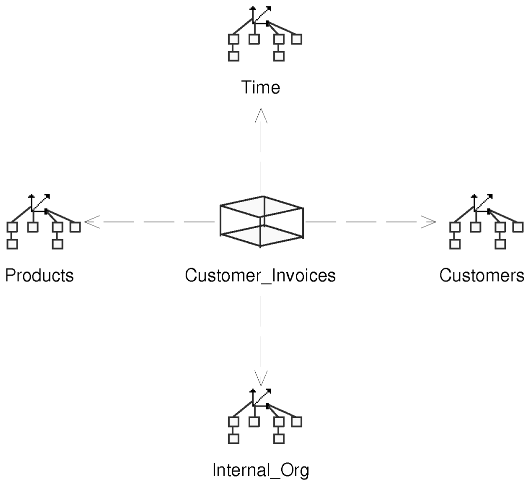


Figure 22: Data Warehouse Conceptual Schema: Level 2 of Customer_Invoices Star)

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.

Figure 25 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 26 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 26, 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[11], whereas the

[11] [28] 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

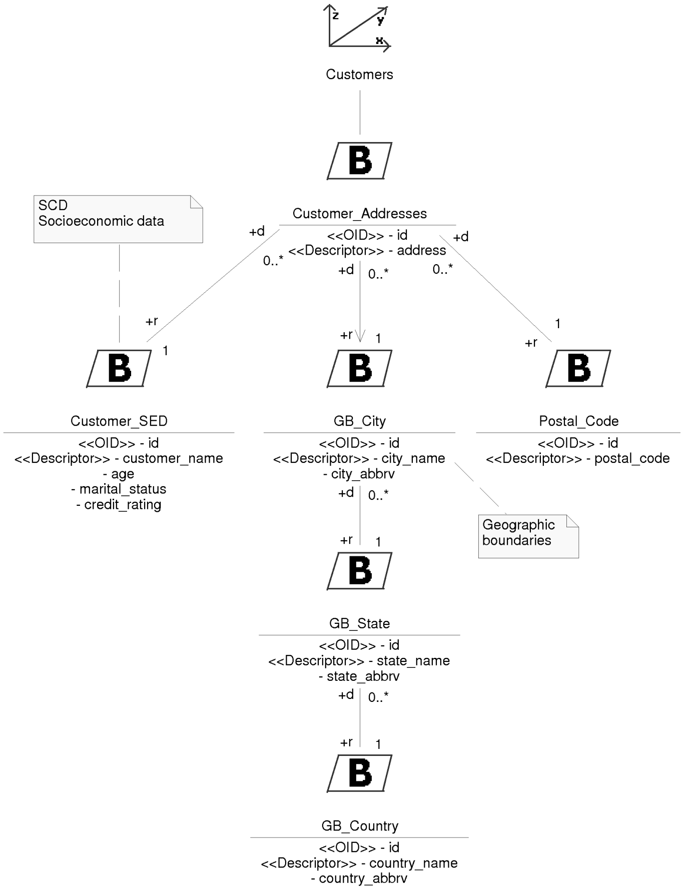


Figure 23: Data Warehouse Conceptual Schema: Customers dimension

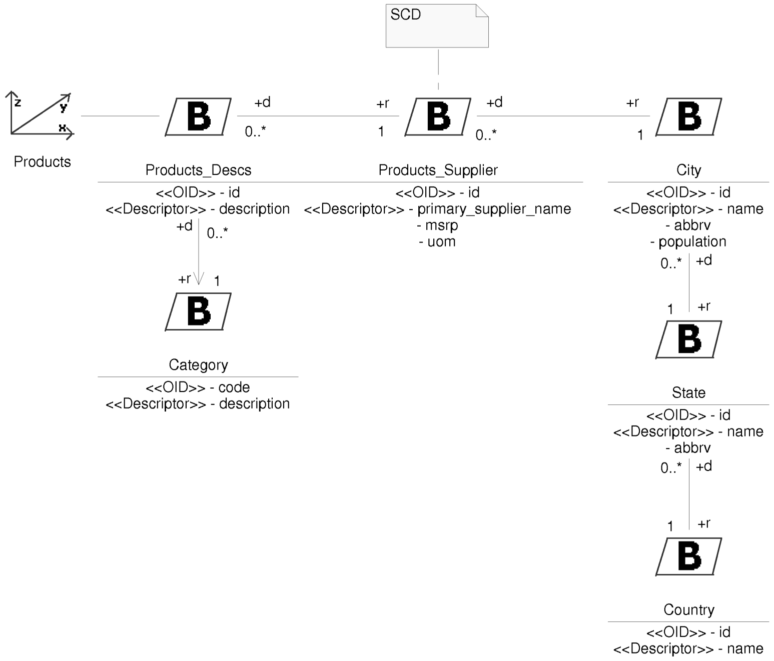


Figure 24: Data Warehouse Conceptual Schema: Products dimension

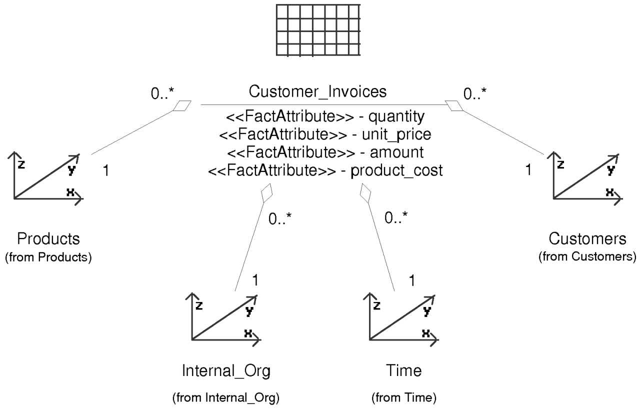


Figure 25: Data Warehouse Conceptual Schema: Customer_Invoices fact

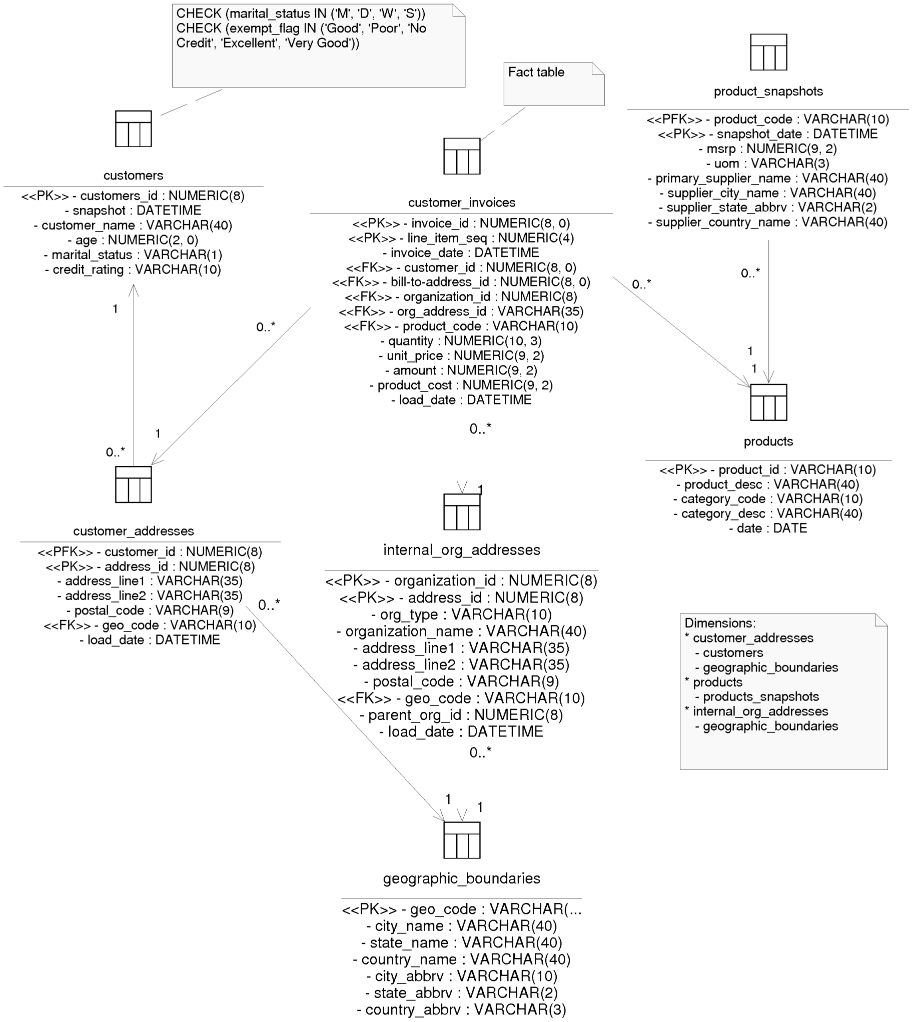

CHECK (marital_status IN ('M', 'D', 'W', 'S'))
CHECK (exempt_flag IN ('Good', 'Poor', 'No Credit', 'Excellent', 'Very Good'))
Fact table
product_snapshots
<<PFK>> - product_code : VARCHAR(10)
<<PK>> - snapshot_date : DATETIME
- msrp : NUMERIC(9, 2)
- uom : VARCHAR(3)
- primary_supplier_name : VARCHAR(40)
- 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
1
0..*
0..*
0..*
1
1
1
0..*
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..*
1
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 26: Logical model (ROLAP) of the data warehouse

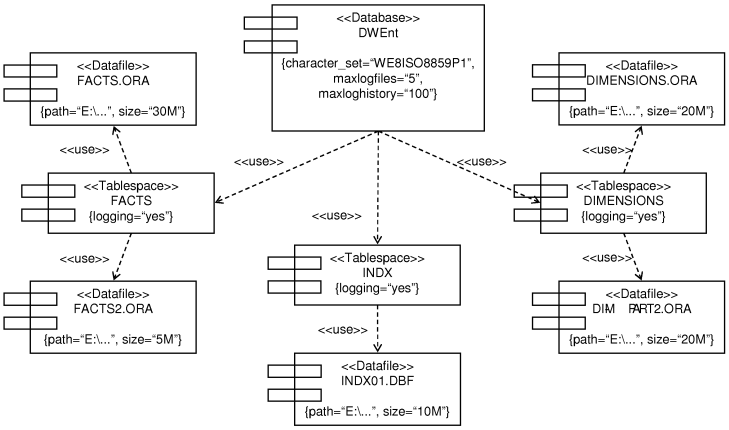


Figure 27: Data Warehouse Physical Schema: component diagram (part 1)

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 24) following the star schema approach [28]; 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.

5.4 Implementation

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 27, 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 28 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 26). 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

through to dimension tables.

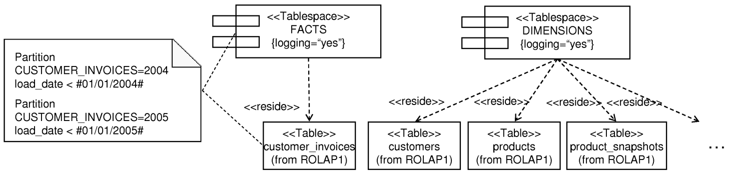


Figure 28: Data Warehouse Physical Schema: component diagram (part 2)

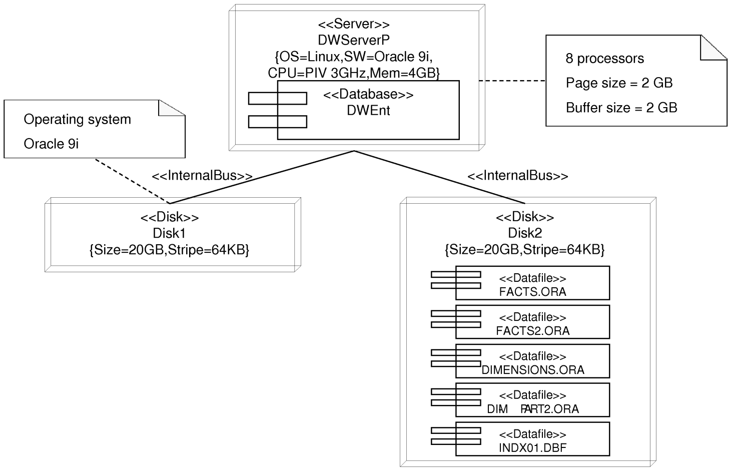


Figure 29: Data Warehouse Physical Schema: deployment diagram

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 29, 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 27).

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 30. On the left hand side of this diagram, different data source servers are represented: SalesServer, CRMServer, and HRMServer, which have been previously defined

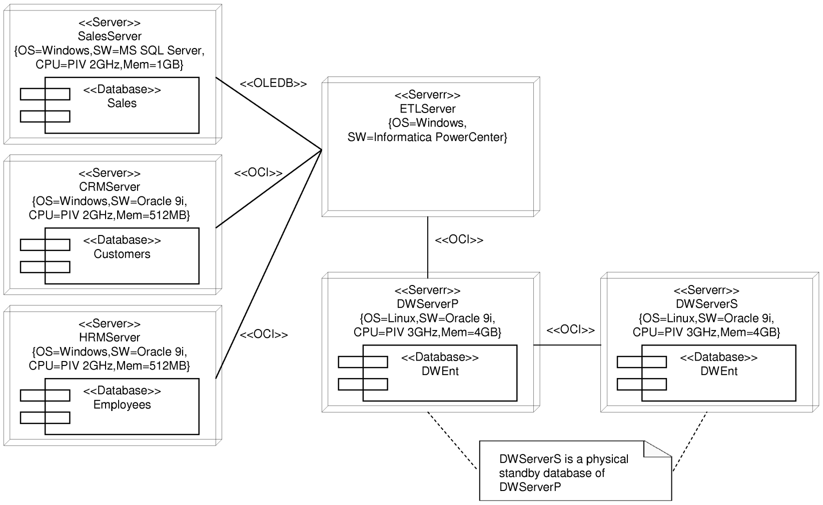


Figure 30: Integration Transportation Diagram: deployment diagram

in Figure 20; on the right hand side, the DWServerP, previously defined in Figure 29, is shown. Moreover, the DWServerS, a physical standby database[12] is also included in the design.

In Figure 30, 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[13] and OCI (*Oracle Call Interface*) to communicate with CRMServer, HRMServer, and DWServer because they use Oracle.

Finally, 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.

For example, in Figure 31, the CTD of our running example is shown. This diagram is based on the list of needs that users defined during the requirements workflow of our DWEP (see Section sec:cs-Requirements). On the left hand side of this diagram, part of the DWPS, which has been previously defined in Figure 29, 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

[12] 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.

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

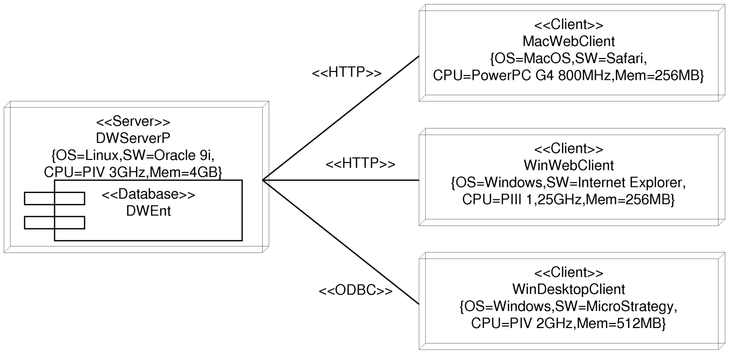


Figure 31: Customization Transportation Diagram: deployment diagram

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.

5.5 Implementation in Oracle

In Oracle, creating a database consists basically of the following steps:

- Creating the database's datafiles[14], its control files[15] and its redo log files[16].
- Creating the system tablespaces.
- Creating the data dictionary (tables, views, etc.).

5.5.1 Creating the Database

Before proceeding, we should point out that the information about the database repository to be created (Figure 27) 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

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

[15] 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.

[16] 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).

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.

5.5.2 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 27, 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 27). In the following, we can see the SQL statements for defining the two 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,
    'E:\ORACLE\ORADATA\DWENT\DIM-PART2.ORA' SIZE 20M
    EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT  AUTO
```

5.5.3 Creating the Data Dictionary

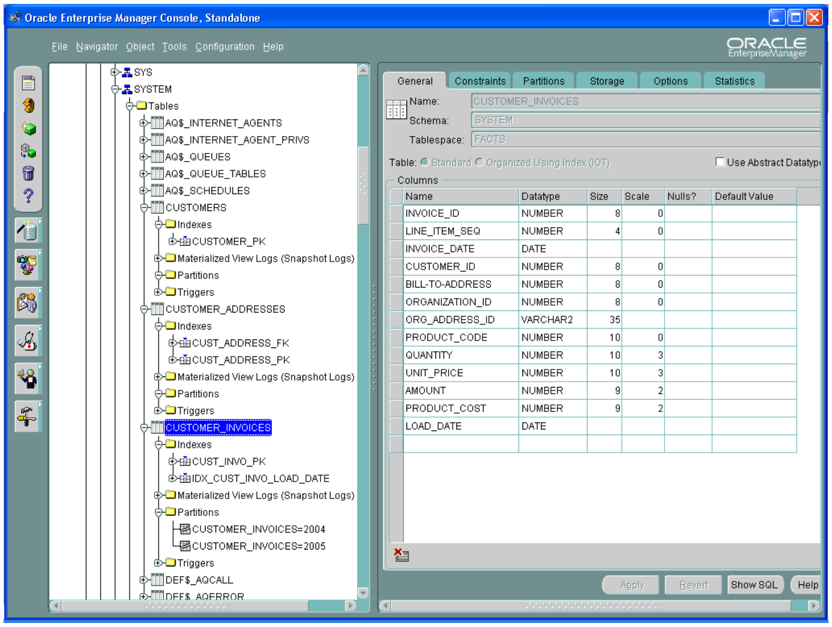
Once both tablespaces and datafiles have been created within a database, the next step is to define fact and dimension tables. First of all, in the following SQL sentences, we can see how we have specified the *product dimension* table. Apart from the columns, we should point out the fact that the table 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 28. 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'))
    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 32 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



```
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 Datatyp

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 32: Defintion of customer_invoices table in Oracle

is intended to locate data for every year in each different partition (Figure 28), and therefore, the column in which we base our partition is load_date. Instead of the previous dimension tables, this fact table is 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.

```
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 subsection 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.

6 Conclusions and Future Work

In this chapter, we have presented our *Data Warehouse Engineering Process* (DWEP), a DW development process based on the UML and the UP. To the best of our knowledge, our method is the only one that addresses the full life cycle of DW development, from the operational data sources to the multidimensional model to the final implementation of the DW.

UML is an standard of the OMG [44] that unifies many years of effort in object oriented analysis and design. UML is the visual modeling language of choice for building object-oriented and component-based systems. On the other hand, UP [26] is a generic and stable process that we have instantiated to cover the development of data warehouses. We have based our proposal in UML and UP for three main reasons: (i) UML is a well-known standard modeling language known by most database designers, thereby designers can avoid learning a new notation, (ii) UML can be easily extended so that it can be tailored for a specific domain with concrete peculiarities such as the multidimensional modeling for data warehouses, and (iii) UML is highly scalable as a new tailored UML profile can further be enriched by adding new properties to adapt it to new situations.

Our main contribution is the definition of several diagrams (techniques) and UML profiles [34, 35, 54, 36, 31] in order to model data warehouses more properly. Whereas the different diagrams provide different views or perspectives of a data warehouse, the engineering process specifies how to develop a data warehouse and ties up all the diagrams together. The main advantages of our approach are:

- The use of a development process, the UP, which is the outcome of more than 20 years of experience.
- The use of the UML, a widely accepted visual modeling language, for designing the different data warehouse diagrams and the corresponding transformations.
- The use of the UML as the modeling language provides much better tool support than using an own modeling language.
- The proposal of a data warehouse development process that addresses both the back-end and the front-end of data warehouses in an integrated manner.

7 Acknowledgement

This work has been partially supported by the METASIGN project (TIN2004-00779) from the Spanish Ministry of Education and Science, by the DADAS-MECA project (GV05/220) from the Valencia Government, and by the DADS (PBC-05-012-2) project from the Regional Science and Technology Ministry of Castilla-La Mancha (Spain).

References

[1] A. Abelló, J. Samos, and F. Saltor. 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)*, volume 2113 of *Lecture Notes in Computer Science*, pages 668–677, Munich, Germany, September 3 - 7 2001. Springer-Verlag.

[2] A. Abelló, J. Samos, and F. Saltor. YAM2 (Yet Another Multidimensional Model): An Extension of UML. In *International Database Engineering & Applications Symposium (IDEAS'02)*, pages 172–181, Edmonton, Canada, July 17 - 19 2002. IEEE Computer Society.

[3] L. Agosta. Market Overview Update: ETL. Technical Report RPA-032002-00021, Giga Information Group, March 2002.

[4] J. Arlow and I. Neustadt. *UML and the Unified Process. Practical Object-Oriented Analysis & Design.* Object Technology Series. Addison-Wesley, 2002.

[5] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language: User Guide.* Object Technology Series. Addison-Wesley, 1999.

[6] R.M. Bruckner, B. List, and J. Schiefer. Developing Requirements for Data Warehouse Systems with Use Cases. In *Proceedings of the 7th Americas Conference on Information Systems (AMCIS'01)*, pages 329–335, Boston, USA, August 3 - 5 2001.

[7] L. Cabibbo and R. Torlone. A Logical Approach to Multidimensional Databases. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98)*, volume 1377 of *Lecture Notes in Computer Science*, pages 183–197, Valencia, Spain, March 23 - 27 1998. Springer-Verlag.

[8] L. Carneiro and A. Brayner. X-META: A Methodology for Data Warehouse Design with Metadata Management. In *Proceedings of 4th International Workshop on the Design and Management of Data Warehouses (DMDW'02)*, pages 13–22, Toronto, Canada, May 27 2002.

[9] J.M. Cavero, M. Piattini, and E. Marcos. MIDEA: A Multidimensional Data Warehouse Methodology. In *Proceedings of the 3rd International Conference on Enterprise Information Systems (ICEIS'01)*, pages 138–144, Setubal, Portugal, July 7 - 10 2001. ICEIS Press.

[10] A. Cockburn. *Writing Effective Use Cases.* Addison-Wesley Professional, 2000.

[11] J. Conallen. *Building Web Applications with UML.* Object Technology Series. Addison-Wesley, 2000. (Last edition: 2nd edition, Addison-Wesley, 2003).

[12] M. Corey, M. Abbey, I. Abramson, and B. Taub. *Oracle8i Data Warehousing.* Oracle Press. Osborne/McGraw-Hill, 2001.

[13] N.T. Debevoise. *The Data Warehouse Method.* Prentice-Hall, New Jersey, USA, 1999.

[14] W. Eckerson. Four ways to build a data warehouse. *Application Development Trends*, May, 2002.

[15] M. Fowler. *UML Distilled. Applying the Standard Object Modeling Language.* Object Technology Series. Addison-Wesley, 1998.

[16] T. Friedman. ETL Magic Quadrant Update: Market Pressure Increases. Technical Report M-19-1108, Gartner, January 2003.

[17] S.R. Gardner. Building the Data Warehouse. *Communications of the ACM*, 41(9):52–60, September 1998.

[18] W. Giovinazzo. *Object-Oriented Data Warehouse Design. Building a star schema*. Prentice-Hall, New Jersey, USA, 2000.

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

[20] M. Golfarelli and S. Rizzi. A Methodological Framework for Data Warehouse Design. In *Proceedings of the ACM 1st International Workshop on Data Warehousing and OLAP (DOLAP'98)*, pages 3–9, Bethesda, USA, November 7 1998. ACM.

[21] D. Hackney. Data Warehouse Delivery: Who Are You? Part I. *DM Review Magazine*, February, 1998.

[22] B. Hüsemann, J. Lechtenbörger, and G. Vossen. Conceptual Data Warehouse Modeling. In *Proceedings of the 2nd International Workshop on Design and Management of Data Warehouses (DMDW'00)*, pages 6.1–6.11, Stockholm, Sweden, June 5 - 6 2000.

[23] IBM. IBM Rational Unified Process (RUP). Internet: http://www.rational.com/products/rup/index.jsp, 2003.

[24] W.H. Inmon. *Building the Data Warehouse*. QED Press/John Wiley, 1992. (Last edition: 3rd edition, John Wiley & Sons, 2002).

[25] J. Poole. Model Driven Data Warehousing (MDDW). Internet: http://www.cwmforum.org/POOLEIntegrate2003.pdf, 2003.

[26] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Object Technology Series. Addison-Wesley, 1999.

[27] M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis. *Fundamentals of Data Warehouses*. Springer-Verlag, 2 edition, 2003.

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

[29] L. Greenfield. Data Extraction, Transforming, Loading (ETL) Tools. The Data Warehousing Information Center. Internet: http://www.dwinfocenter.org/clean.html, 2003.

[30] S. Luján-Mora and J. Trujillo. A Data Warehouse Engineering Process. In *Proceedings of the 3rd Biennial International Conference on Advances in Information Systems (ADVIS'04)*, volume 3261 of *Lecture Notes in Computer Science*, pages 14–23, Izmir, Turkey, October 20 - 22 2004. Springer-Verlag.

[31] S. Luján-Mora and J. Trujillo. Modeling the Physical Design of Data Warehouses from a UML Specification. In *Proceedings of the 8th IASTED International Conference on Software Engineering and Applications (SEA 2004)*, pages 7–12, Cambridge, USA, November 9 - 11 2004.

[32] S. Luján-Mora and J. Trujillo. Physical Modeling of Data Warehouses using UML. In *Proceedings of the ACM Seventh International Workshop on Data Warehousing and OLAP (DOLAP 2004)*, pages 48–57, Washington D.C., USA, November 12 - 13 2004. ACM.

[33] S. Luján-Mora and J. Trujillo. Physical Modeling of Data Warehouses Using UML Component and Deployment Diagrams: Design and Implementation Issues. *Journal of Database Management*, 17(2):12–42, April-June 2006.

[34] S. Luján-Mora, J. Trujillo, and I. Song. Extending UML for Multidimensional Modeling. In *Proceedings of the 5th International Conference on the Unified Modeling Language (UML'02)*, volume 2460 of *Lecture Notes in Computer Science*, pages 290–304, Dresden, Germany, September 30 - October 4 2002. Springer-Verlag.

[35] S. Luján-Mora, J. Trujillo, and I. Song. Multidimensional Modeling with UML Package Diagrams. In *Proceedings of the 21st International Conference on Conceptual Modeling (ER'02)*, volume 2503 of *Lecture Notes in Computer Science*, pages 199–213, Tampere, Finland, October 7 - 11 2002. Springer-Verlag.

[36] S. Luján-Mora, P. Vassiliadis, and J. Trujillo. Data Mapping Diagrams for Data Warehouse Design with UML. In *Proceedings of the 23rd International Conference on Conceptual Modeling (ER'04)*, volume 3288 of *Lecture Notes in Computer Science*, pages 191–204, Shanghai, China, November 8 - 12 2004. Springer-Verlag.

[37] R.A. Maksimchuk and E.J. Naiburg. Entity Relationship Modeling with UML. *DM Direct Newsletter*, January, 2003.

[38] E. Marcos, B. Vela, and J.M. Cavero. Extending UML for Object-Relational Database Design. In *Proceedings of the 4th International Conference on the Unified Modeling Language (UML'01)*, volume 2185 of *Lecture Notes in Computer Science*, pages 225–239, Toronto, Canada, October 1 - 5 2001. Springer-Verlag.

[39] E. Medina and J. Trujillo. A Standard for Representing Multidimensional Properties: the Common Warehouse Metamodel (CWM). In *Proceedings of*

6th East-European Conference on Advances in Databases and Information Systems (ADBIS 2002), volume 2435 of *Lecture Notes in Computer Science*, pages 232–247, Bratislava, Slovakia, September 8 - 11 2002. Springer-Verlag.

[40] D.L. Moody and M.A.R. Kortink. From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart Design. In *Proceedings of the 2nd International Workshop on Design and Management of Data Warehouses (DMDW'01)*, pages 5.1–5.12, Stockholm, Sweden, June 5 - 6 2000.

[41] E.J. Naiburg and R.A. Maksimchuk. *UML for Database Design*. Object Technology Series. Addison-Wesley, 2001.

[42] National Technical University of Athens (Greece). Knowledge and Database Systems Laboratory. Internet: http://www.dblab.ntua.gr/, 2003.

[43] Object Management Group (OMG). Common Warehouse Metamodel (CWM) Specification 1.0. Internet: http://www.omg.org/cgi-bin/-doc?ad/2001-02-01, February 2001.

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

[45] Object Management Group (OMG). Model Driven Architecture (MDA). Internet: http://www.omg.org/mda/, 2004.

[46] Rational Software Corporation. The UML and Data Modeling. Internet: http://www.rational.com/media/whitepapers/Tp180.PDF, 2000.

[47] Ronin International. Enterprise Unified Process (EUP). Internet: http://www.enterpriseunifiedprocess.info/, 2003.

[48] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Object Technology Series. Addison-Wesley, 1999.

[49] K. Siau and Q. Cao. Unified Modeling Language (UML) - A Complexity Analysis. *Journal of Database Management*, 12(1):26–34, 2001.

[50] K. Siau, J. Erickson, and L. Lee. Theoretical versus Practical Complexity: The Case of UML. *Journal of Database Management*, 16(3):40–57, 2005.

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

[52] SQL Power Group. How do I ensure the success of my DW? Internet: http://www.sqlpower.ca/page/dw_best_practices, 2002.

[53] K. Strange. ETL Was the Key to this Data Warehouse's Success. Technical Report CS-15-3143, Gartner, March 2002.

[54] J. Trujillo and S. Luján-Mora. A UML Based Approach for Modeling ETL Processes in Data Warehouses. In *Proceedings of the 22nd International Conference on Conceptual Modeling (ER'03)*, volume 2813 of *Lecture Notes in Computer Science*, pages 307–320, Chicago, USA, October 13 - 16 2003. Springer-Verlag.

[55] J. Trujillo, M. Palomar, J. Gómez, and I. Song. Designing Data Warehouses with OO Conceptual Models. *IEEE Computer, special issue on Data Warehouses*, 34(12):66–75, December 2001.

[56] N. Tryfona, F. Busborg, and J.G. Christiansen. starER: A Conceptual Model for Data Warehouse Design. In *Proceedings of the ACM 2nd International Workshop on Data Warehousing and OLAP (DOLAP'99)*, pages 3–8, Kansas City, USA, November 6 1999. ACM.

[57] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual Modeling for ETL Processes. In *Proceedings of the ACM 5th International Workshop on Data Warehousing and OLAP (DOLAP 2002)*, pages 14–21, McLean, USA, November 8 2002. ACM.

[58] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Modeling ETL Activities as Graphs. In *Proceedings of 4th International Workshop on the Design and Management of Data Warehouses (DMDW'02)*, pages 52–61, Toronto, Canada, May 27 2002.

[59] P. Vassiliadis, Z. Vagena, S. Skiadopoulos, N. Karayannidis, and T. Sellis. ARKTOS: towards the modeling, design, control and execution of ETL processes. *Information Systems*, 26(8):537–561, December 2001.

[60] J. Vowler. Data warehouse design from top to bottom. ComputerWeekly.com. Internet: http://www.computerweekly.com/-Article110431.htm, March 2002.