ELSEVIER

Developing secure data warehouses with a UML extension

Eduardo Fernández-Medina[a,*], Juan Trujillo[b], Rodolfo Villarroel[c], Mario Piattini[a]

[a]*Departamento de Tecnologías y Sistemas de Información, Universidad de Castilla-La Mancha, Paseo de la Universidad, 4, 13071 Ciudad Real, Spain*
[b]*Departamento de Lenguajes y Sistemas Informáticos, Universidad de Alicante, C/San Vicente S/N. 03690, Alicante, Spain*
[c]*Departamento de Computación e Informática, Universidad Católica del Maule, Avenida San Miguel 3605, Talca, Chile*

Abstract

Data Warehouses (DWs), Multidimensional (MD) Databases, and On-Line Analytical Processing Applications are used as a very powerful mechanism for discovering crucial business information. Considering the extreme importance of the information managed by these kinds of applications, it is essential to specify security measures from the early stages of the DW design in the MD modeling process, and enforce them. In the past years, some proposals for representing main MD modeling properties at the conceptual level have been stated. Nevertheless, none of these proposals considers security issues as an important element in its model, so they do not allow us to specify confidentiality constraints to be enforced by the applications that will use these MD models. In this paper, we will discuss the specific confidentiality problems regarding DWs as well as present an extension of the Unified Modeling Language for specifying security constraints in the conceptual MD modeling, thereby allowing us to design secure DWs. One key advantage of our approach is that we accomplish the conceptual modeling of secure DWs independently of the target platform where the DW has to be implemented, allowing the implementation of the corresponding DWs on any secure commercial database management system. Finally, we will present a case study to show how a conceptual model designed with our approach can be directly implemented on top of Oracle 10g.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Data warehouses; UML extension; Multidimensional conceptual modeling; Confidentiality; Secure data warehouses

1. Introduction

Multidimensional (MD) modeling is the foundation of Data Warehouses (DWs), MD Databases, and On-Line Analytical Processing (OLAP) applications. These systems are used as a very powerful mechanism for discovering crucial business information in strategic decision-making processes. Considering the extreme importance of the information that a user can discover by using this kind of applications, it is crucial to specify confidentiality measures in the MD modeling process from the early stages of a DW project. Indeed, the very survival of the organizations depends on the correct management, security and confidentiality of information [1].

In fact, as some authors have remarked [2,3], security of information is a serious requirement

*Corresponding author. Tel.: +34 926 295300x3747; fax: +34 926 295354.

E-mail addresses: eduardo.fdezmedina@uclm.es (E. Fernández-Medina), jtrujillo@dlsi.ua.es (J. Trujillo), rvillarr@spock.ucm.cl (R. Villarroel), mario.piattini@uclm.es (M. Piattini).

which must be carefully considered, not as an isolated aspect, but as an element which turns up as an issue in all stages of the development lifecycle, from requirement analysis to implementation and maintenance.

In spite of the fact that different approaches for integrating security into the system development process have been proposed [4], they have only considered information security from a cryptographic point of view. On the other hand, the approach stated by Chung et al. [5] also insist on integrating security requirements into design, by providing designers with models specifying security aspects, but they do not deal with database and DW specific issues.

The goal of information confidentiality is to ensure that users can only access the information which they have privileges for. In the case of MD models, confidentiality is crucial, because business information is very sensitive and can be discovered by executing a simple query. Sometimes, MD models also store information regarding private or personal aspects of individuals, like identification data, medical data or even religious beliefs, ideologies, or sexual tendencies. In this case, confidentiality is redefined as privacy. Many governments are very concerned about the issue of privacy, passing laws to protect that of the individual, such as the European Union Directive 95/46/CE from the European Parliament and Council on people protection, regarding personal data management and free circulation of data [6] (and its national adaptation, such as for instance LOPD in Spain and BDSG in Germany), the European Union's Safe Harbour Law, the United States' HIPPA (Health Insurance Portability and Accountability Act), Gramm-Leach-Bliley Act, Sarbanes-Oxley Act, etc. According to these laws, organizations must provide exhaustive access control and complete audit trails for all access to personal data, and failure to comply with these laws tends to be very strictly sanctioned, by imposing severe penalties.

With regard to the modeling of DWs, we believe that the conceptual modeling phase has been widely recognized as an important step in the design of DWs, as the sooner we represent the main MD properties at the early stages of a DW project, the better the implemented DW will represent the requirements of the final user. In recent years, various approaches have been proposed for representing the main MD properties at the conceptual level (see Section 2). However, none of these approaches for conceptual MD modeling considers security as an important issue of its conceptual model, so they do not solve the problem of security within this kind of systems. Moreover, in most real world DW projects, security aspects are issues that usually rely on database management system (DBMS) administrators. We argue that the design of these security aspects should be considered alongside the conceptual modeling of DWs from the early stages of a DW project, allowing us to attach user security information to the basic structures of an MD model (e.g. dimensions, facts, attributes, and so on). In this way, we would be able to generate this information in a semi-automatic or automatic way, into a target platform. The final DW will thus suit the user's security requirements better.

In the last few years, we have made several proposals regarding this issue. In [7], we stated an OO conceptual MD modeling approach, for a powerful conceptual modeling of MD systems based on the Unified Modeling Language (UML). This proposal considers major relevant MD properties at the conceptual level in a way that is both elegant and easy. Furthermore, in [8] we applied the grouping mechanism called *package* provided by the UML. Thus, when modeling complex and large DW systems, we are not restricted to the use of flat UML class diagrams. Moreover, in [9] we presented a UML profile[1] for MD modeling based on our previously proposed approaches. To the best of our knowledge, in [10] we presented the first UML extension for the design of secure DWs that allows us to represent the main security information on data and their constraints in the MD modeling at the conceptual level. This approach is based on a combination of the multilevel security model [11], which allows us to classify both information and users into security classes, enforcing the mandatory access control (MAC), together with the role-based access control (RBAC) [12,13].

In this paper, we will extend our previous UML approach for designing secure DWs as follows: firstly, we will provide new stereotypes for the data types necessary to correctly represent security information in an MD model (i.e. facts, dimensions, attributes, etc.). Secondly, we will adapt the

---

[1]A *profile* is a set of improvements that extend an existing UML type of diagram for a different use. These improvements are specified by means of the extendibility mechanisms provided by the UML (stereotypes, tagged values and constraints) in order to be able to adapt it to a new method or model. In this paper, we will use the terms *UML extension* and *UML profile* indistinctly.

corresponding tagged values and well-formedness rules related to the new data types and stereotypes. Our conceptual modeling approach is completely independent of the target platform, which means that, we are able to implement secure MD models with any of the DBMS that can implement multi-level databases, such as Oracle10g Label Security [14] and DB2 Universal Database (UDB) [15]. Finally, another important issue provided in this paper is an in-depth detail on how to implement the security issues specified with our conceptual approach into a commercial platform such as Oracle 10g, thereby allowing us to enforce the security information under consideration in further design steps in a real world project.

The remainder of this paper is structured as follows: In Section 2, the main related work will be summarized while in Section 3 the conceptual approach for MD modeling which we based our work on will be briefly explained. The new UML extension for secure MD modeling will be proposed in Section 4. In Section 5, a case study will be presented and our UML extension for secure MD modeling will be applied. Section 6 will put forward an approach for implementing a conceptual model carried out with our UML extension in Oracle 10g. Finally, in Section 7, our main conclusions will be stated and our future work will be introduced.

2. Related work

In this section, we will organize the related work according to the three main research topics covered by this paper: (i) MD modeling, (ii) security integration into the design process, and (iii) security and access control models for DWs.

2.1. MD modeling

In recent times, several MD data models have been proposed. Some of them fall into the logical level (such as the well-known star-schema by Kimball [16]). Others may be considered as formal models, as they provide a formalism for considering the main MD properties. A review of the most relevant logical and formal models can be found in [17,18].

In this section, we will only briefly refer to the most relevant models that we consider "pure" conceptual MD models. These models provide a high level of abstraction for the main MD modeling properties at the conceptual level and are totally independent of implementation issues. An out-standing feature provided by these models is that they provide a set of graphical notations (such as the classical and well-known EER model) that facilitates their use and reading. On the one hand, there are approaches that extend the classical EER to adapt it to the MD modeling such as *The Multidimensional/ER (M/ER) Model* by Sapia et al. [19,20] and *The starER Model* by Tryfona et al. [21]. Others provide their own graphical notation such as *The Dimensional-Fact (DF) Model* by Golfarelli et al. [22,23] and the Model proposed by Hüseman et al. [24]. On the other hand, there are other proposals that use the object-oriented paradigm and are based on the UML such as *The Yet Another Multidimensional Model (YAM²)* by Abelló et al. [25], The Object-Oriented metacube proposed by Nguyen et al. [26,27] and The ADAPTed UML model proposed by Priebe et al. [28]. In this paper, we will take the UML approach for the MD conceptual modeling proposed by Trujillo et al. [7,9] as our base. This approach has lately been improved and formalized as an extension (*profile*) of the UML by Luján-Mora et al. [7,9] by using the standard extension mechanisms (stereotypes, tagged values and constraints) provided by the UML. We have chosen this latest approach for two main reasons: (i) it is a formal extension of the UML, as it uses the standard UML extension mechanisms, which makes it easier to be used by designers, modelers and administrators, and (ii) we consider it a very powerful approach since it allows us to model not only basic MD terms (such as facts, dimensions, classification hierarchies and so on) as in the rest of approaches, but also more complex MD features such as *nonstrict* and *complete* hierarchies, degenerate dimensions, degenerate facts, *many-to-many* relationships between facts and a particular dimension, and many more. We do not provide a more detailed comparison between all these approaches here as it is outside the scope of this paper.

However, none of these approaches for MD modeling considers security as an important issue of their conceptual models, so they do not solve the problem of security within this kind of systems (only The ADAPTed approach has been extended by Priebe and Pernul in [29] with some security constraints, as we will later summarize in Section 2.3).

2.2. Security integration into the design process

There are a few proposals that try to integrate security into conceptual modeling such as [30,31]. In

[30], Smith presents the first attempt to use a conceptual model to represent security semantic. He includes constructs for describing security constraints in the entity–relationship model. A more formalized extension to the entity–relationship model, with techniques for detecting conflicting constraints is presented by Pernul et al. in [31]. These approaches are very interesting, but they are focused on the database conceptual modeling, and they do not consider the peculiarities of DWs and MD modeling. These techniques are therefore not directly suited to the security problems regarding DWs. Many of their ideas and concepts, such as multilevel security, are considered in our approach, however. Moreover, [32] extends the Object Modeling Technique with multilevel security, to the modeling of security in information systems (IS) and databases. More recent proposals are UMLSec [33,34] and SecureUML [35] where UML is extended to develop secure systems. These approaches are very interesting, but, again, they only deal with IS in general, whilst conceptual database and DW design are not considered. Moreover, a methodology and a set of models have been proposed [36] to design secure databases to be implemented with Oracle 10g Label Security (OLS10g) [14]. This approach, based on the UML, is important because it considers security aspects at all stages of the development process, from requirement-gathering to implementation. Together with the previous methodology, the proposed Object Security Constraint Language (OSCL) [37], based on the Object Constraint Language (OCL) [38] of the UML, allows us to specify security constraints in the conceptual and logical database design process, and to implement these constraints into a specific DBMS, the OLS10g. Nevertheless, the previous methodology and models [36] do not consider the design of secure MD models for DWs, and therefore, are not appropriate for the representation of the peculiarities of DWs. In addition, the OSCL language is not completely valid for specifying security constraints of DWs.

2.3. Security and access control models for DWs

As described above, the peculiarity of DWs, along with the MD modeling and its terms (facts, dimensions, classification hierarchies and so on) used both for designing and for querying DWs, makes it necessary to deal with specific security models for DWs. These peculiarities of both DWs and the MD model make the classical database access control defined on tables, rows and so on, insufficient. Instead, we agree with Priebe and Pernul in [39,40] where it is stated that security constraints must be defined in terms of the MD modeling that final users work with when querying a DW (facts, dimensions, classification hierarchies, etc.).

In the relevant literature, we can find several initiatives for the inclusion of security in DWs. Many of them are focused on interesting aspects related to access control, multilevel security, the applications of these aspects to federated databases, applications using commercial tools and so on. For instance, in [41], authors present a prototype model for DW security based on metadata, which defines different user groups, and a different view of data for each user group. This initiative is interesting, but in a real situation, grouping users and defining a different view for each group is not enough, being necessary to combine groups and to specify complex confidentiality security constraints, which is performed by our proposal. On the other hand, in [42] an approach is presented which integrates security from the data sources, and propagates it to DW design. This approach is attractive, but we believe it is necessarily an explicit and independent phase which concerns itself to confidentiality problems in the conceptual MD modeling.

Additionally, there are some interesting proposals which define authorization models and security for DWs [39,40,43–45], but they only deal with OLAP operations (e.g. roll-up or drill-down) accomplished with OLAP tools. So these proposals are not conceived for their integration into MD modeling as part of the DW design process, and as a consequence, inconsistent security measures could be defined. We believe that we should consider basic security measures for business DWs with a conceptual model from the early stages of a DW project. Then, more specific security rules can be defined for particular groups of users in terms of data marts, views, OLAP tools, or any other analysis tools, but which are consistent with the main security rules defined for the DW.

Finally, in our opinion, the pieces of work carried out by Priebe and Pernul are the most interesting ones in the area of modeling secure DWs and OLAP applications. First of all, we agree with [39], where it is stated that some elements of the MD model such as a measure, a dimension level or a slice of a particular cube must be hidden from a certain group

of users. Thus, in [40], authors propose the definition of basic security measures for main OLAP operations when querying a DW. To this end, these authors extend the ADAPTed UML approach for specifying security measures on a UML class diagram which they use for the MD modeling of OLAP tools at the conceptual level. This is a very interesting proposal and has some similarities to ours, as the aforesaid authors provide a set of authorization rules (AURs) based on the roles of the different users and they also focus on the *read* operation (which is the most common operation for querying DWs). However, we extend their work in some sense, providing other features. Firstly, the conceptual modeling approach which we take as our base allows us to model complex DW scenarios and at the same time considers nonstrict and complete hierarchies, degenerate dimensions, degenerate facts or *many-to-many* relationships between facts and a particular dimension (with the corresponding attributes). Hence, we also try to provide a wider approach for modeling a business DW, rather than an approach aimed at modeling small data marts. Finally, we cover some relevant situations such as providing role hierarchies and compartment groups for users, and not just the typical security levels of multilevel security. This allows us to cover more real DWs, as in real world projects, since we have found that users normally are classified by different criteria. Furthermore, we provide a constraint language that allows us to specify security constraints, taking into account the above-mentioned user classification criteria. This constraint language, based on the OCL [29] is independent of any target commercial tool, which guarantees the independency of our proposal from any implementation detail. As we will show at the end of the Case Study (Section 6), our constraint language can be easily translated into any commercial DBMS supporting multilevel security access.

To the best of our knowledge, only our previous work [10] sets the basis for providing a conceptual model for the design of secure DWs. In [10], we presented a preliminary version of our extension of the UML (profile) that allows us to represent main the security information on the data and their constraints in the MD modeling at the conceptual level. The proposed extension is based on the profile presented in [9] for the conceptual MD modeling. This last profile allows us to consider the main MD modeling properties and at the same time it is based on the UML (designers can avoid learning a new specific notation or language). In our approach for the conceptual modeling of DWs, we consider the multilevel security model [11], but we focus on considering aspects regarding the *read* operations because this is the most common operation in DW/ OLAP applications. We should take into account that if we add security measures to an MD model, we are basically adding security measures to a model that has been built to satisfy final users; and final users will use the *read* operation when querying a DW with an OLAP tool. Other operations such as that of inserting or updating will be considered in future work in the framework of ETL—Extraction, Transformation and Loading processes. However, this is for further research, as security measures both in transactional databases and DWs must be considered. This is a hard task to perform since the security measures defined in these databases are very different one from another and hence there are many conflicts that need to be solved.

Our model, therefore, allows us to classify both information and users into security classes, and enforce the mandatory and RBAC [11]. In this paper, we will refine this approach with the main aspects mentioned in the introduction, thereby obtaining a more powerful conceptual model for designing secure DWs.

3. Object-oriented MD modeling

In this section, we will outline the UML we use for DW conceptual modeling [7,9]. This approach has been specified by means of a UML profile which contains the necessary stereotypes for carrying out the MD modeling at the conceptual level successfully [46]. The main features of MD modeling considered here are the relationships many-to-many between facts and one specific dimension, degenerated dimensions, multiple classification and alternative path hierarchies, and nonstrict and complete hierarchies. In this approach, structural properties of MD modeling are represented by means of a UML class diagram in which the information is clearly organized into facts (items of interest for a given business) and dimensions (context in which facts have to be analyzed).

Facts and dimensions are represented by means of fact classes (stereotype Fact ▦), and dimension classes (stereotype Dimension ▫), respectively. Fact classes are defined as composite classes in shared aggregation relationships of *n* dimension classes. The minimum multiplicity in the role of the

dimension classes is 1 (all facts are always related to all dimensions). Relations many-to-many between a fact and a specific dimension are specified by the multiplicity 1.* in the role of the corresponding dimension class.

Let us introduce the case study that we will use throughout the paper to illustrate our approach. In this case study, we are interested in finding out the profitability of a patient through the different admissions, in terms of the diagnosis that was carried out, the patient on whom it was made, the ward the patient was assigned to, and the date the diagnosis was made. In the MD model (see Fig. 1) of this example, the fact is represented by the Admission fact class and the dimensions by the Diagnosis, Patient, Ward and Time dimension classes. In Fig. 1 we can also see how the Admission fact class has a many-to-one relationship with all dimension classes (diagnosis, patient, ward and time).

A fact is composed of measures or fact attributes. By default, all measures within the fact class are considered to be additive. For nonadditive measures, additive rules are defined as constraints and are included in the fact class. Furthermore, derived measures can be also explicitly represented (indicated by /) and their derivation rules are placed between square brackets near the fact class. See the benefit attribute and its corresponding derived rule in the Admission fact class in Fig. 1.

With this approach, we can also define identifying attributes in the fact class (stereotype OID). Thus, degenerated dimensions can be considered [16], thereby representing other fact features in addition to the measures for analysis. For instance, we could store the bill number (bill_number) as a degenerate dimension (see Admission fact in Fig. 1); allowing us to consider another interesting fact feature which is different from the usual measures.

Fig. 1. Multidimensional modeling using the UML.

With respect to dimensions, each level of a classification hierarchy is specified by a base class (stereotype Base ▣). An association of base classes specifies the relationship between two levels of a classification hierarchy. The only prerequisite is that these classes must define a Directed Acyclic Graph (DAG) rooted in the dimension class (DAG constraint is defined in the stereotype Dimension). The DAG structure can represent both multiple and alternative path hierarchies. Apart from being considered internally, one can easily find out the correct DAG structure by making all the roles of the different classification hierarchy levels visible. For example, we have made some roles for different dimensions to clarify the navigability between classification hierarchy levels visible. For example, *Ward* can be aggregated/rolled-up to *Hospital* and *Hospital* into *Hospital_Type*. It is the same classification hierarchy path, but we have explicitly shown the roles of the association relation between *Hospital* and *Hospital_Type* to make it clear that Hospital rolls-up to (roll $+r$ in the association relation) *Hospital_Type*. Conversely, $+d$ means that *Hospital_Type* can be drilled-down into *Hospital*. We can also see the same association roles between some classification hierarchy levels within the *Patient* dimension. From now on, we will not use these roles in order to avoid cluttered diagrams.

On the other hand, every base class must also contain an identifying attribute (OID) together with a descriptor attribute[2] (stereotype Descriptor). These attributes are necessary for an automatic generation process into commercial OLAP tools, as these tools store this information on their metadata.

Due to the flexibility of the UML, we can also consider nonstrict hierarchies (an object at a hierarchy's lower level belongs to more than one higher-level object) and complete hierarchies (all members belong to one higher-class object and that object consists of those members only). These features are specified by means of the multiplicity of the roles of the associations and by defining the constraint {completeness} in the target associated class role respectively. See Patient dimension in Fig. 1 for an example of all kinds of classification hierarchies. Lastly, the categorization of dimensions is considered by means of generalization/specialization relationships of UML.

---

[2] A descriptor attribute will be used as the default label in the data analysis in OLAP tools.

4. UML extension for secure MD modeling

As we have commented on previously, the goal of the UML extension proposed in this paper is to allow us to specify confidentiality constraints in the conceptual MD modeling. These confidentiality constraints will restrict the access to data, based on the sensitivity of data and the identity and properties of the users. That is to say, for each piece of data, we need to specify which users will have the necessary access privilege (e.g. a confidentiality constraint could be "*The personal information of patients can be read by administrative staff, but their medical information can be only read by doctors*"). To do so, we need each piece of data to refer to the user or the set of users that will have access privilege, and then we need to define a user classification criterion within every particular access privilege. We have considered three compatible and optional classification criteria that allow us to refer to user groups with a very high level of accuracy: (i) Security levels, that indicate the clearance level of the user; (ii) Security user roles, which are used by a company to organize users in a hierarchical role structure, according to the responsibilities of each type of work (each user can play more than one role); and (iii) Security user compartments, which are also used by organizations to classify users into a set of horizontal compartments or groups, such as geographical location, area of work, etc. (each user can belong to one or more compartments). Moreover, DW designers will be able to define a user profile class, specifying more user properties that can be used in confidentiality constraints.

Once the DW structure has been defined, we should perform these three steps in order to specify their confidentiality properties:

1. To define the organization of users that will have access to the MD system with precision. We can define a precise level of granularity considering three ways of organizing users: Security hierarchy levels (which indicate the clearance level of the user), user Compartments (which indicate a horizontal classification of users), and user Roles (which indicate a hierarchical organization of users according to their roles or responsibilities within the organization). For each user, a security level, one or more compartments and/or one or more user roles should be defined (e.g. Bob is a *Doctor*, and his security level is *TopSecret*)

2. To classify the information into the MD model. We can define for each element of the model (fact class, dimension class, fact attribute, etc.) its security information, specifying a sequence of security levels, a set of user compartments, and a set of user roles. We can also specify security constraints considering these security attributes. The security information and constraints indicate the security properties that users have to possess to be able to access the information.

3. To enforce the mandatory and RBAC. The typical operations that final users can execute in this type of systems are query operations. So, the MAC has to be enforced for *read* operations. The MAC rule for read operations depends on a dominance rule. A user can access data only if the user classification dominates the data classification. This happens only if, (a) the security level of the user is greater than, or equal, to the security level of the data, (b) all the user compartments that have been defined for the data are defined for the user, and, (c) at least one of the user roles (or a descendent in the user role hierarchy) that the data have defined, is played by the user. A trusted process is in charge of assigning the user classification for each user, and the data classification for each data. Fig. 2 shows an extension of the pattern that has been presented in [47], which clearly explains this access control model. The RBAC is also integrated into this combined model, since the user role hierarchy is involved in the dominance rule we have previously presented.

In this paper, we will only focus on the second stage by defining a UML extension that allows us to classify the security elements in a conceptual MD model as well as to specify security constraints. Furthermore, in Section 6, we will deal with a prominent work studying the third stage by generating the necessary structures in the target DBMS to consider all security aspects represented in the conceptual MD model. Finally, let us point

out that the first stage is concerned with security policies defined in the organization by managers, and it is outside the scope of this paper.

According to [48], an extension to the UML begins with a brief description and then lists and describes all the stereotypes, tagged values, and constraints of the extension. In addition to these elements, an extension contains a set of well-formedness rules. These rules are used to determine whether a model is semantically consistent with itself. According to this quote, we will define our UML extension for secure conceptual MD modeling following the schema composed of these elements:

- *description* (a brief description of the extension in natural language),
- *prerequisite extensions* (this element indicates whether the current extension needs the existence of previous extensions),
- *stereotypes/tagged values* (the definition of the stereotypes and/or tagged values),
- *well-formedness rules* (the static semantics of the metaclasses are defined both in natural language and as a set of invariants defined by means of OCL expressions), and
- *comments* (any additional comment, decision or example, usually written in natural language).

For the definition of stereotypes, we will follow the structure that is suggested in [46], which is composed of a name, the base metaclass, the description, the tagged values and a list of constraints defined by means of OCL. For the definition of tagged values, the type of tagged values, the multiplicity, the description, and the default value are defined.

4.1. Description

This UML extension re-uses a set of stereotypes previously defined in the UML extension proposed in [9] for MD modeling at the conceptual level, and
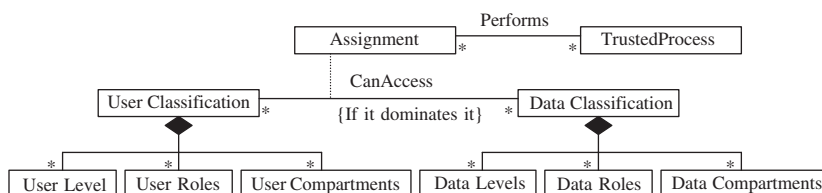

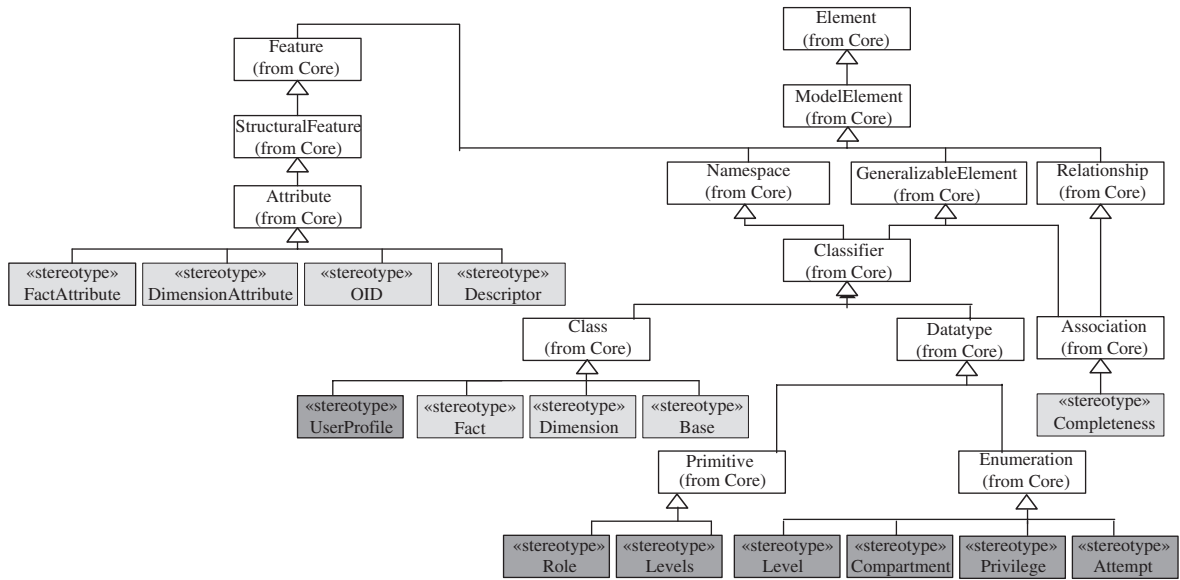Fig. 2. Class model for multilevel access control.

Fig. 3. Extension of the UML with stereotypes.

defines a set of tagged values, stereotypes, and constraints, which enables us to create secure MD models. This previous UML extension defined the *Fact, Dimension* and *Base* stereotypes by specializing the metaclass *Class*, the *FactAttribute, DimensionAttribute, OID* and *Descriptor* stereotypes by specializing the metaclass *Attribute*, and the *Completeness* stereotype as a specialization of the metaclass *Association*.

In our security extension, we have defined seven new stereotypes: one of them (*UserProfile*) specializes in the Class model element, two (*Role* and *Levels*) specialize in the Primitive model element and four (*Level, Compartment, Privilege* and *Attempt*) specialize in the Enumeration model element. The 20 tagged values we have defined (see Section 4.4.) are applied to certain components that are especially particular to MD modeling, allowing us to represent them in the same model and in the same diagrams that describe the rest of the system. These tagged values will represent the classification information of the different elements of the MD modeling (fact class, dimension class, base class, attributes, etc.), and they will allow us to specify security constraints depending on this security information and on the value of attributes of the model. The new stereotype will help us identify a special class that will define the profile of the users of the system. A set of inherent constraints are specified with the aim of defining well-formedness rules. The correct use of our extension is assured by the definition of constraints in both natural language and OCL [38].

In Fig. 3, we have represented portions of the UML metamodel[3] to show where our stereotypes fit. We have only represented the specialization hierarchies, as the most important fact about a stereotype is the base class that the stereotype specializes in. In that figure, new stereotypes are colored in dark gray, whereas stereotypes we re-use from our previous profile [33] are in light gray and classes from the UML metamodel remain white.

4.2. Prerequisite extensions

This UML profile re-uses stereotypes that were previously defined in another UML profile in [9]. This profile provided the necessary stereotypes, tagged values, and constraints to carry out with the MD modeling properly, allowing us to represent the main MD properties of DWs at the conceptual level. To facilitate the comprehension of the UML profile which we present and use in this paper, we will provide a summary of the specification of these stereotypes in Table 1.

---

[3]All metaclasses come from the *Core Package*, a subpackage of the *Foundation Package*. We have based our extension on the UML 1.5 as this is the current accepted standard. To the best of our knowledge, the current UML 2.0 is not yet the final accepted standard.

4.3. Datatypes

First of all, we need the definition of some new data types which are used in our tagged values definitions. In Fig. 3 we can see the base classes these new stereotypes are specialized from. In Table 2, we will provide the new data types definitions we have specified. All the information considered in these new stereotypes has to be defined for each specific MD model depending on its confidentiality properties, and on the number of users and complexity of the organization in which the MD model will be operative. Finally, we need some syntactic definitions that are not considered in OCL standard. Particularly, we need the new collection type *Tree* with its typical operations (see the defined well-formedness rules at the end of this section).

4.4. Tagged values

In this section, we will provide the definition of several tagged values specified at different levels of detail such as *model*, *classes*, *attributes*, *instances* and constraints. The reason to specify them as ''global'' tagged values within each corresponding level is to avoid specifying the same tagged value for each different stereotype defined in our approach (see Fig. 3). Moreover, this way, any stereotype or element can use them whenever convenient.

Table 3 shows the tagged values of all elements in this extension. Each tagged value can be used if necessary; otherwise the default value is automatically assigned to. Some of these tagged values must be used all together. For instance, if we need to specify the situation in which accesses to the information of a class have to be recorded in a log file for future audit, we should use *LogType* and *LogCond* tagged values together in that class (audit rules—ARs). On the other hand, if we need to specify a security constraint, we can use OCL and the *InvolvedClasses* tagged value to specify in which situation the constraint has to be enforced (sensitivity information assignment rules—SIARs). Finally, if we need to specify a special security constraint in which a user or a set of users (depending on a condition) can or cannot access to the corresponding class, independently of the security information of that class, we should use *exceptions* by considering together the following tagged values: *InvolvedClasses*, *ExceptSign*, *ExceptPrivilege* and *ExceptCond* (AURs). Unfortunately, the standard extension mechanisms that the UML provides do not allow us to formally group tagged values, and therefore, we will group them into UML notes.

4.5. Stereotypes

Having all these tagged values available, we can specify security constraints on an MD model, but with all of them depending on the value of attributes and specified tagged values. In this extension, we need to define a stereotype so as to specify other types of security constraints (see Table 4). The stereotype *UserProfile* is necessary to specify constraints depending on particular information of a user or a group of users, e.g., depending on the user citizenship, age, etc. Then, each of the previously defined data types and tagged values will be used in the *fact*, *dimension* and *base* stereotypes in order to consider other security aspects.

Table 1
Stereotype from the UML profile for conceptual MD modeling [9]

Name	Base Class	Description
Fact	Class	Classes of this stereotype represent facts in an MD model
Dimension	Class	Classes of this stereotype represent dimensions in an MD model
Base	Class	Classes of this stereotype represent dimension hierarchy levels in an MD model
OID	Attribute	Attributes of this stereotype represent OID attributes of Facts, Dimension or Base classes in an MD model
Fact-Attributes	Attribute	Attributes of this stereotype represent attributes of Fact classes in an MD model
Descriptor	Attribute	Attributes of this stereotype represent descriptor attributes of Dimension or Base classes in an MD model
Dimension-Atttribute	Attribute	Attributes of this stereotype represent attributes of Dimension or Base classes in an MD model
Completeness	Association	Associations of this stereotype represent the completeness of an association between a Dimension class and a Base class or between two Base classes

Table 2
Stereotypes of the new data types

Name	Level
Base class	Enumeration
Description	The type Level will be an ordered enumeration composed of all security levels that have been considered (these values, are typically unclassified, confidential, secret and top secret, but they could be different).
Attributes	None
Associations	None

Constraints	- The value of this type must be one of those that have been defined for the model (these values are defined in Table 3)
	    Context Model
	    Inv self.securityLevels->includes(Level)

Semantics	Level instances will be the values defined by the DW designer, but they could be as follows: {unclassified, confidential, secret, topSecret}
Notation	Fig. 4 (a)
Name	**Levels**
Base class	Primitive
Description	The type Levels will be an interval of levels composed by a lower level and a upper level

Attributes	lowerLevel
	    Type: UML::Datatypes::Enumeration::Level
	upperLevel
	    Type: UML::Datatypes::Enumeration::Level

Associations	None

Constraints	- LowerLevel and upperLevel form a correct interval of security levels from those that have been defined for the model (these values are defined in Table 3)
	    Context Model
	    inv self.securityLevels-> subSequence (lowerLevel, upperLevel)->notEmpty()

Semantics	Levels instances will be those adopted by the attributes lower level and upper level, that are instances of the Level stereotype
Notation	Fig. 4 (b)
Name	**Role**
Base class	Primitive
Description	The type role will represent the hierarchy of user roles that can be defined for the organization (for the sake of simplicity we consider that there is no multiple inheritance).
Attributes	RoleName Type: UML::Datatypes::Primitive::String
Associations	None

Constraints	- The value of this type must be one of those that have been defined within the organizational user roles hierarchy in the context of the model (that is defined in Table 3)
	    Context Model
	    Inv self.securityRoles->includes(Role)

Semantics	Role instances will be values of String
Notation	Fig. 4 (c)
Name	**Compartment**
Base class	Enumeration
Description	The type compartment is the enumeration composed by all user compartments that have been considered for the organization.
Attributes	None
Associations	None

Constraints	- The value of this type must be one of those that have been defined for the model (that are defined in Table 3)
	    Context Model
	    Inv self.securityCompartments->includes(Compartment)

Semantics	Compartment instances will be values of String
Notation	Fig. 4 (d)
Name	**Privilege**
Base class	Enumeration
Description	The type Privilege will be an ordered enumeration composed of all the different privileges that have been considered (these values, typically are read, insert, delete, update, all).
Attributes	None
Associations	None
Constraints	None
Semantics	Privilege instances will be the following values: { read, insert, delete, update, all, }
Notation	Fig. 4 (e)
Name	**Attempt**
Base class	Enumeration
Description	The type Attempt will be an ordered enumeration composed of all different access attempts that have been considered (these values, typically are none, all, frustratedAttempt, success-fullAccess, but they could be different).
Attributes	None
Associations	None
Constraints	None
Semantics	Attempt instances will be the following values: {all, frustratedAttempt, successfullAccess, none}
Notation	Fig. 4 (f)

The type of the arguments of *subSequence* collection is integer, but for the sake of readbility, we consider that the arguments can be elements of the subSequence. The correct expression should be subSequence(self.securityLevels->indexOf(c.securityLevels.lowerLevel), self.securityLevels ->indexOf(c.securityLevels.upperLevel). We consider this simplification in all uses of *subSequence* operation.

Table 3
Tagged values definition

Tagged Values of the Model				
Name	Type	M	Description	Default Value
classes	Set(OclType)	1	It specifies all classes of the model. This new tagged value is useful in order to navigate through all classes of the model	Empty set
securityLevels	Sequence (Levels)	1	It specifies all security levels (ordered from less to more restrictive) that can be used by the model elements	Empty sequence
securityRoles	Role	1	It specifies the hierarchical role structure that has been defined for the organization. This type will be managed as a tree	Empty tree
security-Compartments	Set (Compartment)	1	It specifies the set of compartments that have been defined for the organization	Empty set
Tagged Values of the Class				
Name	Type	M	Description	Default Value
SecurityLevels	Levels	1	It specifies the interval of possible security level values that an instance of this class can have. If the upper and lower security levels are the same, all instances will have the same security level. Otherwise, the specific instance security level will be defined according to a security constraint	The lower level (if we consider traditional levels), should be Unclassified)
SecurityRoles	Set(Role)	1	It specifies a set of user roles. Each role is the root of a subtree of the general user role hierarchy defined for the organization. All instances of this class can have the same user roles, or maybe subtrees of the roles that have been defined for the class. A security constraint can decide the user roles for each instance according to the value of some attributes of the instance	The set composed by one role that is the root of the role hierarchy defined for the model
Security-Compartments	Set (Compartment)	1	It specifies a set of compartments. All instances of this class can have the same user compartments, or a subset of them. A security constraint can decide the user compartments for each instance according to the value of some attribute of the instance	Empty set of compartments
LogType	Attempt	0..*	It specifies whether the access has to be recorded: none, all accesses, only frustrated accesses, or only successful accesses	None
LogCond	OCLExpression	0..*	It specifies whether the access has to be recorded	Empty
Involved-Classes	Set(OclType)	0..*	It specifies the classes that have to be involved in a query to be enforced in an exception	Empty
ExceptSign	{+,-}	0..*	It specifies if an exception permits (+) or denies (-) access to instances of this class to a user or a group of users	+
Except-Privilege	Set(Privilege)	0..*	It specifies the privileges the user can receive or remove	Read
ExceptCond	OCLExpression	0..*	It specifies the condition that users have to fulfill to be affected by this exception	Empty
Tagged Values of the Attribute				
Name	Type	M	Description	Default Values
SecurityLevels	Levels	1		The lower one
SecurityRoles	Set(Role)	1		The role hierarchy of the model
SecurityCompartments	Set (Compartment)	1		Empty set of compartments
Tagged Values of the Instance				
Name	Type	M	Description	Default Values
SecurityLevel	Level	1	It specifies the security level of an instance	The level of its class
SecurityRoles	Set(Role)	1	It specifies a set of user roles for this instance. Each role is a subtree of the user role hierarchy defined for the organization.	The user roles of its class
Security-Compartments	Set (Compartment)	1	It specifies the set compartments for an instance	The compartments of its class
Tagged Values of the Constraint				
Name	Type	M	Description	Default Values
Involved-Classes	Set(OCLType)	0..1	It specifies the classes, that are involved in a query, and which have to be enforced in the constraint	Empty

M stands for Multiplicity

Due to space constraints, we do not include the descriptions of the tagged values of attributes as they are similar to their counterpart tagged values of classes.

Table 4
Stereotype UserProfile of our extension

Name	**UserProfile**
Base class	Class
Description	Classes of this stereotype contain all the properties that the systems manage from users

Constraints	- This class has no associations to another classes
	  Self.AssociationsEnd.size()=0
	- There is no more than one class of this type
	  Context Model
	  Inv self.classes->forAll(oclisTypeOf(UserProfile))->size()<=1
	- The name of a class of this stereotype will be *UserProfile*
	  self.className=*UserProfile*

Tagged Values	None
Icon	

4.6. Well-formedness rules

Finally, we will identify and specify in both natural language and OCL some well-formedness rules needed for the correct use of the new UML elements specified in this extension. These rules are grouped in Table 5.

These well-formedness rules should be enforced when the designer is building the conceptual model. Therefore a CASE tool should be available to support the secure MD modeling and to check these well-formedness rules automatically and therefore to ensure the security elements have been correctly introduced into. We have developed a prototype that performs this activity.

4.7. Comments

Many of the previous constraints are very intuitive, but we have to ensure they are carried out; otherwise the system can be inconsistent.

In addition to the previous constraints, the designer can specify security constraints with OCL. If the security information of a class or an attribute depends on the value of an attribute of an instance, it can be expressed as an OCL expression (see Fig. 6). Normally, security constraints defined for stereotypes of classes (fact, dimension and base) will be defined by using a UML note attached to the corresponding class instance.[4] We do not impose any restriction on the content of these notes so as to allow the designer the greatest flexibility, except for

---

[4]The connection between a note and the element it applies to is shown by a dashed line without an arrowhead as this is not a dependency [49].

those restrictions imposed by the tagged values definitions (Fig. 4).

Several implication rules could be defined in our model, such as those defined for the ORION access control model [50], those defined for a model of content-based authorization in object-oriented databases [51], and more recently, the implication rules included in a grammar for the security policy specification language for web services [52]. Nevertheless, due to the high complexity of the definition of correct implication rules in our model, we prefer to define rules for solving conflicts. Some conflicts can appear between several AURs, several SIARs, and even between AURs and SIARs. We solve these conflicts according to the following rules: (i) If there is a conflict between a positive AUR and a SIAR, the set of users that will be able to access the information will be composed of users who fulfill the SIAR and users who do not fulfill the SIAR but who fulfill the subject condition of the AUR. (ii) If there is a conflict between a negative AUR and a SIAR, the set of users that will not be able to access the information will be composed of users who do not fulfill the SIAR and the users who fulfill the SIAR but who fulfill the subject condition of the AUR. (iii) If there is a conflict between two SIARs, the security information for each instance will be the result of applying the most restrictive SIAR. (iv) An AUR that refers to an individual user has a higher level of preference than any other AUR that refers to a set of users. (v) An AUR that refers to a particular user role *r* has greater preference than any other AUR that refers to an ascendant of *r* in the user-roles tree. (vi) Two AURs that refer to different sets of users (for instance to a compartment and a role) have the same preference, and (vii)

Table 5
Well-formedness constraints

| |
|---|
| **-   Security information of instances:** |
| The security level of the instance of a class has to be included in the ranking of security levels that has been defined for the class. The same rule is applicable to instances of attributes. |
| **context** Model
inv self.classes-> forAll(c \| c.allInstances -> forAll (i \| self.securityLevels-> subSequence(c.securityLevels.lowerLevel, c.securityLevels.upperLevel)-> includes(i.securityLevel))) |
| The user roles of an instance of a class have to be subtrees of the roles trees that have been defined for the class. The same rule is applicable to instances of attributes. |
| **context** Model
inv self.classes-> forAll(c \| c.allInstances -> forAll (i \| c.securityRoles-> includesAll(i.securityRoles))) |
| The user compartments of an instance of a class,have to be a subset of the compartments that have been defined for the class. The same rule is applicable to instances of attributes. |
| **context** Model
inv self.classes-> forAll(c \| c.allInstances -> forAll (i \| i.securityCompartments-> includesAll(i.securityCompartments))) |
| **-   Relationship between the security information of classes and its attributes:** |
| The security levels defined for an attribute have to be equal or more restrictive than the security levels defined for its class. The same rule is applicable to role hierarchies and user compartments. |
| **context** Model
inv self.classes-> forAll(c \| c.attributes-> forAll(a \| self.securityLevels-> subSequence(c.securityLevels.lowerLevel, a.securityLevels.upperLevel)-> includesAll(self.securityLevels-> subSequence(a.securityLevels.lowerLevel,a.securityLevels.upperLevel)))
inv self.classes-> forAll(c \| c.attributes-> forAll (a \| c.securityRoles-> includesAll(a.securityRoles)))
inv self.classes-> forAll (c \| c.attributes-> forAll (a \| c.securityCompartments-> includesAll(a.securityCompartments))) |
| **-   Categorization of dimensions:** |
| When a dimension class is specialized into several base classes, the security levels of the subclasses have to be equal to or more restrictive than the security levels of the superclass. The same rule is applicable to role hierarchies and user compartments. |
| **context** Model
inv self.classes-> forAll(c \| c.subClasses-> forAll(s \| self.securityLevels-> subSequence(c.securityLevels.lowerLevel, s.securityLevels.upperLevel)-> includesAll(self.securityLevels-> subSequence (s.securityLevels.lowerLevel,s.securityLevels.upperLevel))
inv self.classes-> forAll(c \| c.subClasses-> forAll (s \| c.securityRoles-> includesAll(s.securityRoles))
inv self.classes-> forAll (c \| c.subClasses-> forAll (s \| c.securityCompartments-> includesAll(s.securityCompartments))) |
| **-   Classification hierarchies.** As a general rule, we can consider that the more specific information is, the more restrictive its access is: |
| If class A has a 1..* association with class B, means that information of A groups information of B, so B is more specific than A. The security level defined for class B has to be more restrictive than the security level defined for the class A. This rule is also applicable to user roles and compartments. |
| **context** Model
inv self.classes-> forAll(c \| c.associationEnd-> forAll (a \| c.a.multiplicity>1 implies self.securityLevels-> subsequence(c.securityLevels.lowerLevel, a.securityLevels.upperLevel)-> includesAll(self.securityLevels-> subSequence(a.securityLevels.lowerLevel, a.securityLevels.upperLevel))
inv self.classes-> forAll(c \| c.associationEnd-> forAll (a \| c.a.multiplicity>1 implies c.securityRoles-> includesAll(a.securityRoles))
inv self.classes-> forAll(c \| c.associationEnd-> forAll (a \| c.a.multiplicity>1 implies c-securityCompartments-> includesAll (a.securityCompartments))) |
| If class A has a *..* association with class B, the designer has to decide which class contains the most specific information. This well-formedness rule cannot be specified because it depends on design decisions. |
| **-   Derived attributes:** |
| The security levels of a derived attribute have to be equal to or more restrictive than, the attributes which this attribute is based on. The same rule is applicable to user roles and compartments. By default, the derived attribute inherits the security information of the attribute it is based on. |
| **context** Model
inv self.classes-> forAll(c \| c.attributes -> forAll(a \| a.derived implies a.derivedFrom-> forAll (d \| self.securityLevels-> subsequence(a.securityLevels.lowerLevel, d.securityLevels.upperLevel)-> includesAll(self.securityLevels-> subSequence(d.securityLevels.lowerLevel, d.securityLevels.upperLevel))
inv self.classes-> forAll(c \| c.attributes -> forAll(a \| a.derived implies a.derivedFrom-> forAll (d \| d.securityRoles -> includesAll (a.securityRoles))
inv self.classes-> forAll(c \| c.attributes -> forAll(a \| a.derived implies a.derivedFrom-> forAll (d \| d.securityCompartments -> includesAll (a.securityCompartments)) |
| **-   Combination of dimensions:** |
| A query in the fact class has to consider the security information that has been defined for that class. |
| A query that involves the combination of a dimension class (or maybe a base class) and a fact class has to consider the combination of the security information in the dimension (or base) class and in the fact class. The security levels of the combination will be the most restrictive in the security levels of the dimension (or base) class and the fact class. The same rule is applicable to user roles and compartments. |
| A query that involves the combination of several dimensions class, and the fact class, has to consider the combination of the security information of all classes. The security levels of the combination will be the most restrictive in the security levels of all classes. The same rule is applicable to the user roles and compartments. |
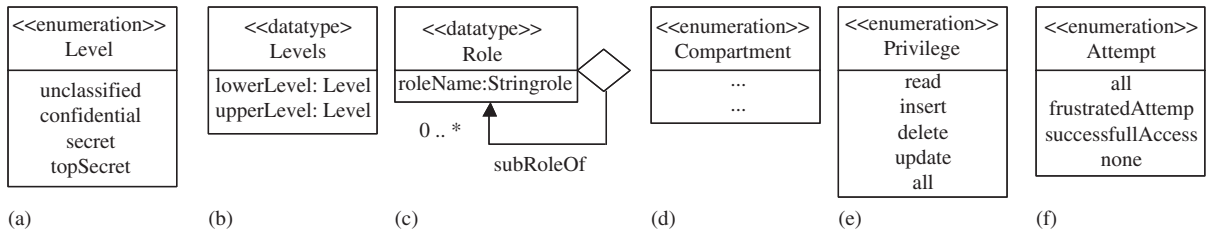
Fig. 4. New Data types.

If two AURs have the same preference, we will select the negative one. If they have the same sign there is no conflict.

5. A case study applying our extension for secure MD modeling

In this section, we will apply our extension for the conceptual design of a secure MD model within the context of a typical health-care system. Regarding the example of Fig. 1 (in Section 3), we have only considered an example in a shortened form (Fig. 6), so as to focus our attention on the main security specifications. In order to define both data and user classifications, we have defined a simplified hierarchy of user roles that is typical for a hospital (the most general is *hospitalEmployee*, which is then specialized into the roles *health* and *nonHealth*, and which are in turn specialized into the roles *doctor* and *nurse* in the former case, and into *maintenance* and *administrative*) in the latter. As security levels, we have considered in this case *confidential, secret* and *topSecret* (see Fig. 5). In this example, compartments have not been defined, as they depend on organization policies. Once we have defined the classification information (roles, levels and/or compartments), it is possible to enrich the MD conceptual model by integrating classification details and security and audit constraints by using the new elements (stereotypes and tagged values) we have defined in this extension.

Fig. 6 shows an MD model that includes a fact class (*Admission*), two dimensions (*Diagnosis* and *Patient*), two base classes (*Diagnosis_group* and *City*), and a class (*UserProfile*). *UserProfile* class (stereotype *UserProfile*) contains the information of all users who will have access to this MD model. *Admission* fact class—stereotype *Fact*—contains all individual admissions of patients in one or more hospitals. *Patient* dimension contains the information of hospital patients. *City* base class contains the information of cities, and it allows us to group
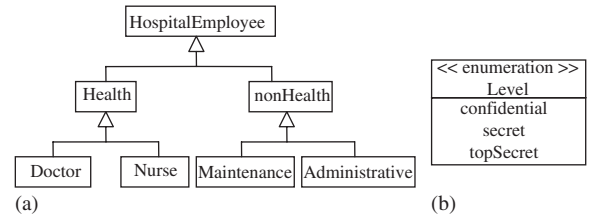

Fig. 5. (a) User roles hierarchy and (b) security levels.

patients by cities. *Diagnosis* dimension contains the information of each user diagnosis. Finally, *Diagnosis_group* contains a set of general groups of diagnosis. Each group can be related to several diagnoses, but a diagnosis will be always related to a group.

Once the MD model has been performed, the user profile has been defined, and the user classification information (levels, roles and/or compartments) has been identified, the designer, together with a security expert, should perform the assignment of user classification to the elements of the MD models; afterwards they should identify and specify possible security constraints (SIARs, AURs, and ARs).

We have identified some static user classification for the classes of the conceptual model (see tagged values defined inside the class box in Fig. 6):

- *Admission* fact class can be accessed by all users who have *secret* or *top secret* security levels (we use the tagged value *SecurityLevels (SL)* of *Admission* class), and play *health* or *administrative* roles (we use the tagged value *SecurityRoles (SR)* of *Admission* class). This means that user who have the *Maintenance* user role will not have access privileges over the admission information, and users who carry out *health* or *administrative* roles need to be classified at least into *secret* level to have access. The *cost* attribute can be only accessed by users who perform the *administrative*
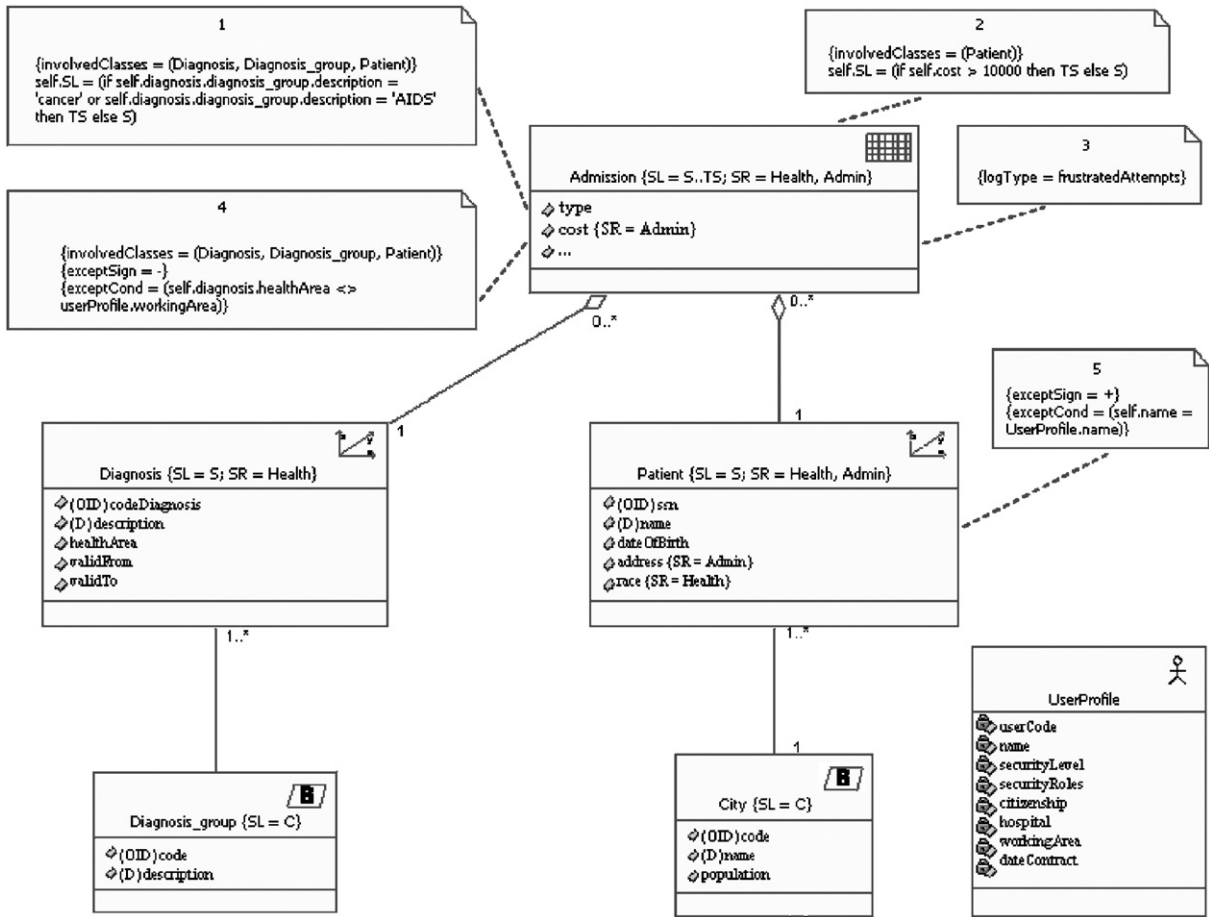
Fig. 6. Example of multidimensional model with security information and constraints.

role (tagged value *SR* of *cost* attribute), and therefore the access to this attribute will be forbidden for other users (*health* and *maintenance* employees).

- *Patient* dimension can be accessed by all users who have *secret* security level—tagged value *SL*-, and perform *health* or *administrative* roles—tagged value *SR*-. The *Address* attribute can be only accessed by users who have an *administrative* role—tagged value *SR* of attributes-, since this information is not necessary from a health point of view. Moreover, the *Race* attribute can be only accessed by *health* employees, since this information can be important in diagnosis, or to determine treatment. This information could be used by administrative staff *irregularly*, however, and it is not relevant for their administrative function.
- *City* base class can be accessed by all users who have *confidential* security level—tagged value

SL-. This information is not sensitive, so the classification is not rigorous.
- *Diagnosis* dimension can be accessed by users who have a *health* role—tagged value *SR*-, and have *secret* security level—tagged value *SL*-.
- Finally, *Diagnosis_group* can be accessed by all users who have *confidential* security level— tagged value SLs-.

We have identified two SIARs, two AURs and one AR for this straightforward example. These security constraints have been specified by using the previously defined constraints, stereotypes and tagged values[5]:

1. (SIAR) The security level of each instance of *Admission* is defined by a security constraint

---

[5]The number of each numbered paragraph corresponds to the number of each note in Fig. 6.

specified in the model. If the value of the *description* attribute of the *Diagnosis_group* to which the *diagnosis* that is related to the *Admission* is *cancer* or *AIDS*, the security level—tagged value *SL*- of this admission will be *top secret*, otherwise *secret*. We consider this situation sensitive, and we protect the data by specifying this security rule. This constraint is only applied if the user makes a query whose information comes from the *Diagnosis* dimension or *Diagnosis_group* base classes, together with the *Patient* dimension—*involvedClasses*, tagged value of constraints-. Therefore, a user who has *secret* security level could obtain the number of patients with *cancer* for each city, but never if information of the *Patient* dimension appears in the query. To specify this SIARs, we use and group two class tagged values, *involvedClasses*, which specify the classes that should be involved in a query for the rule to be applicable, and *SL*, which specifies the security level of each instance, depending on a condition that is specified by an OCL expression.

2. (SIAR) The security level—tagged value *SL*- of each instance of *Admission* can also depend on the value of the *cost* attribute (*topSecret* if *cost* is greater than 10.000, and *secret* in other cases), which indicates the price of the admission service. In this case, the constraint is only applicable to queries that contain information of the *Patient* dimension—tagged value *involvedClasses*-. If this rule was not specified, a user with *secret* security level could take a glance at the health life of famous and rich people.

3. (AR) We would like to record, for future audit, access attempts to the fact class, but which the system denies because of lack of permission. The tagged value *logType* has been defined for the *Admission* class, specifying the value *frustratedAttempts*. In this case it is not necessary to specify audit conditions, but it would be possible in more complex situations. In such cases, we should use the *logCond* tagged value grouped with *logType*.

4. (AUR) For reasons of confidentiality, we could deny access to admission information to users whose working area is different than the area of a particular admission instance, when the query involves *Diagnosis, Diagnosis_group* and *Patients*. This is an AUR which is specified as an exception in *Admission* fact class, considering tagged values *involvedClasses, exceptSign* (-, since we deny access) and *exceptCond* (an OCL expression that specifies the condition, based on the value of the attributes and on the profile of the users).

5. (AUR) Patients could be special users of the system. In this case, it could be possible for those who are being given health care to have access to their own information as patients (for instance, for querying their personal data). This situation can be specified as an AUR, composed of the *except Sign* and *exceptCond* tagged values in the *Patient* class.

The *privilege* that is considered in these exception is *read,* but we have not specified it in the model (the default value of *exceptPrivilege* tagged value is *Read*).

Note that, by using this extension, it is possible to specify a wide range of confidentiality constraints in the conceptual MD model which have been identified very early, and which can be semi-automatically implemented into the target DBMS.

In this simple example, we have used all new data types of our extension, excepting *Compartment*, since the example does not need a so high a level of user classification (the combination of user roles and security levels is enough). We have also used almost all stereotypes defined for the model, class, attribute and instance, but once more not including all those regarding compartments, *logCond* of classes and some of attributes and instances. Moreover, we use the stereotype *UserProfile* in this example, defining all properties of users. Regarding the well-formedness rules we have defined for this extension, it would be tedious to check whether they have been fulfilled here, and this should be done automatically.

6. Implementation

The properties represented in a conceptual model are not always supported by commercial implementation platforms. Therefore, an adaptation of concepts frequently needs to be performed, and even part of the semantics of the conceptual model cannot be represented in the target platform. In this section, we will state our analysis of Oracle technology, presenting a transformation approach to Oracle DBMS from our secure conceptual MD model. We have chosen Oracle DBMS since it supports security and audit facilities by means of some of its components, but other DBMS or OLAP

tools could be considered. We have also pointed out how the implementation could be tacked by Oracle OLAP tools, and how part of the design and implementation of secure DWs can be automated.

This section is organized into three subsections. The first subsection will present the most important characteristics of the components of Oracle 10g DBMS that allow us to implement our UML extension (OLS10g, Oracle Virtual Private Databases (VPD), and Oracle Fine-Grained Auditing (FGA)), and we will detail how a conceptual MD model generated by using the UML extension we present in this paper can be implemented into Oracle 10g. Section 6.2 will show some considerations regarding Oracle OLAP tools. Finally, in Section 6.3 we will present an overview of the prototype we have built for automating the design and implementation process of secure DWs.

6.1. Implementing secure DWs with oracle DBMS

Oracle is a very complete DBMS, which has different tools for implementing security. In this subsection we will introduce some of these tools, and we will present a guide to implement our UML-based secure DWs with Oracle DBMS. Finally, we will put forward some snapshots of Oracle DBMS with a set of details of our running example.

6.1.1. Oracle10g label security

OLS10g [14] is a component of version 10 of the Oracle DBMS which allows us to implement multi-level databases [53]. OLS10g defines labels that are assigned to the rows and users of the database. These labels contain confidentiality information for rows, and authorization information for users. OLS10g defines a combined access control mechanism, considering MAC by using the content of the labels, and Discretionary Access Control (DAC) which is based on privileges. An extended study of these access control techniques can be found in [11]. This combined access control imposes the rule that a user will be only entitled to have access to a particular row if authorized to do so by the DBMS, he/she has the necessary privileges, and the label of the user dominates the label of the row. Fig. 7 represents this combined access control mechanism by means of four steps: (1) the user performs a query, (2) the access control mechanism processes the privileges and classification data of the user, (3) the access control mechanism processes the result of the query and the corresponding classification data,

and compares it with the user classification. Finally, if the user has the necessary privileges, the access control mechanism filters all the data which classification class is dominated by the user classification class.

The security label for each row contains its security level, user compartments and hierarchical user groups, defining the confidentiality properties of the row, and therefore the properties the user has to possess to be able to access the row. Components of labels for users are as follows:

(a) *Security levels*: Four security levels have to be defined for each user: maximum level, which limits read accesses users will not be able to read rows with a higher security level than their maximum level); minimum level, which limits write accesses (users will not be able to insert, update or delete rows with a lower security level than its minimum level); default level, which is the level (between maximum and minimum) in which the user is connected to the DBMS; and row level, which is the level (between maximum and minimum) that is assigned to the row when it is created by this user, and is not always used.

(b) *User compartments*: Four compartment types must be defined for each user; compartments with read access permission; compartments with write access permission; compartments by default; and row compartments (not always used).

(c) *Hierarchical user groups*: Four group types have to be defined for each group; groups with read access permission; groups with write access permission; groups by default; and row groups (not always used).

Additionally, each user will have a set of discretionary privileges. These privileges allow the user to avoid some MAC constraints.

The access control rules that OLS10g enforces in the case of read access are:

(1) The default user security level has to be greater than, or equal to, the row security level.

(2) The user label has to include at least one of the hierarchical groups (or ascendant) that the row label contains.

(3) The user label has to include all the compartments that the row label contains.

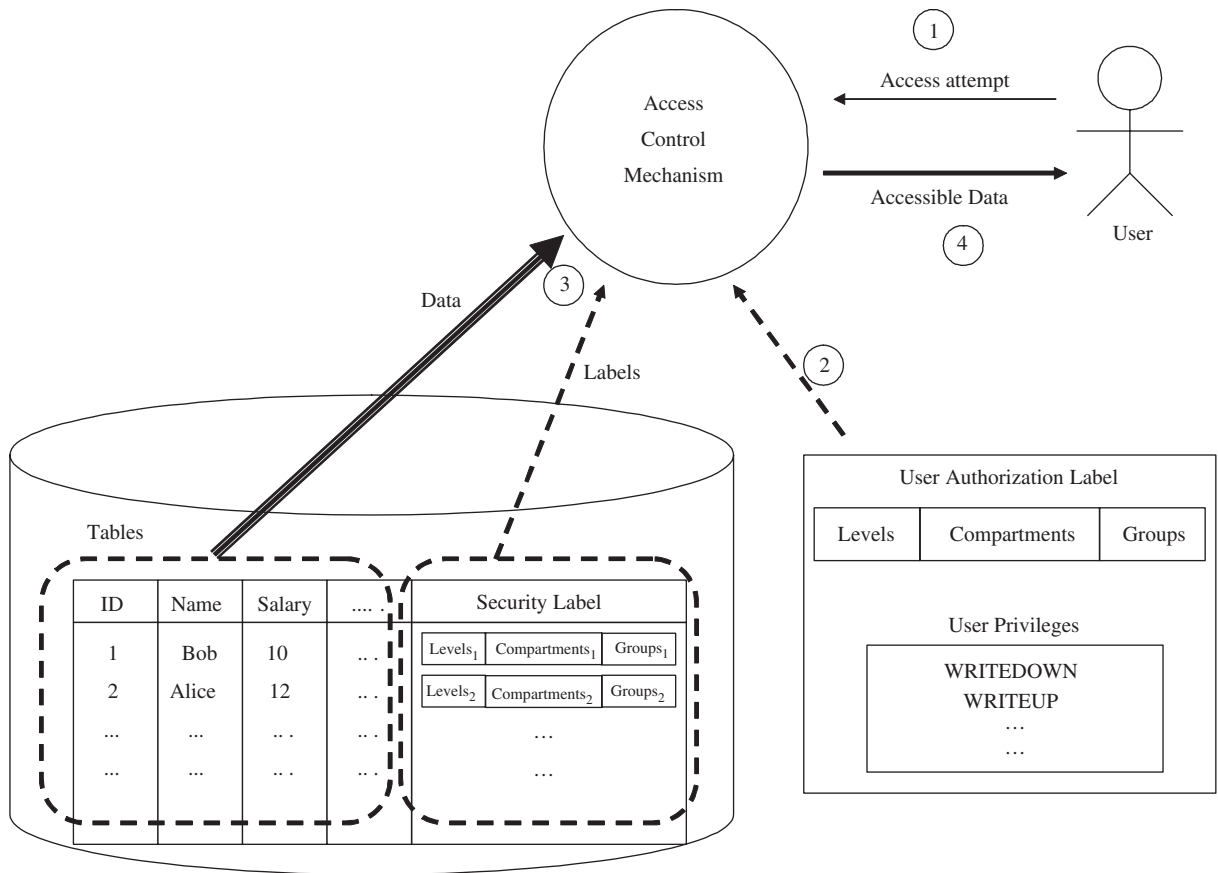All security information specified in OLS10g has to be performed in the context of a *security policy*.

Fig. 7. Access Control Mechanism.

```
(1)   CREATE_POLICY('MyPolicy', 'MyLabel', 'HIDE, READ_CONTROL')
(2)   CREATE_LEVEL('MyPolicy', 10, 'L', 'Low')
(3)   CREATE_LEVEL('MyPolicy', 20, 'H', 'High')
(4)   CREATE_GROUP('MyPolicy', 1, 'E', 'Europe')
(5)   CREATE_GROUP('MyPolicy', 2, 'N', 'North Europe', 'E')
(6)   CREATE_GROUP('MyPolicy', 3, 'S', 'South Europe', 'E')
(7)   CREATE_COMPARTMENT('MyPolicy', 1, 'ELEC', 'Electricity')
(8)   CREATE_COMPARTMENT('MyPolicy', 2, 'SOFT', 'Software')
```

Listing 1. Security Policy, Levels and Groups Definition.

When we create a security policy, we have to specify the name of the policy, the name of the column that will store the labels, and finally other options of the policy. Once this has been created, we must define the valid levels, compartments and hierarchical groups in the context of this security policy.

Listing 1 illustrates an example in which *MyPolicy* is created (1), indicating that the column that will store the labels will be *MyLabel*, and also specifying two options: *HIDE*, which means that *MyLabel* will be hidden for users; and *READ_CONTROL*, which specifies that the DBMS has to enforce the MAC for read operations. We define

Low and *High* as valid security levels (2, 3). When a new security level is defined, we have to specify the name of the policy, a number that indicates the order of the levels, a short name, and the name of the security level. So, three hierarchical groups are defined (4–6), considering *Europe* as the most general group, and *Northern Europe* and *Southern Europe* as descendents. When a new group is created, we have to specify the name of the policy, the number of the group, a short name, the name of the group, and the short name of its father in the hierarchy. Finally *Electricity* and *Software* are defined as two different business lines, and therefore as two compartments (7, 8).

Once a policy has been defined, it can be discretionally applied to one or more tables of the database. Moreover, we also have to define the mechanisms by which the rows will be labeled. OLS10g offers three different ways to label rows:

(1) Explicitly specifying the label row once it has been inserted into the database.
(2) Selecting the option *LABEL_DEFAULT* when the policy is created. This option ensures that each row inherits the row default security information from the label of the user who creates it.
(3) By using *labeling functions*, which are triggered when *INSERT* or *UPDATE* operations are executed. Labeling functions define the information of the security label according to the value of the columns of the row that is inserted or updated.

Listing 2 shows a labeling function (1) which creates the label to be assigned to a new row according to the value of the column *TypeBusiness*. If the business is *Electricity* then the security label will be composed of the security level *Low*, the compartment *Electricity* and the group *Europe*, and in other cases (if the business is *software*), then the label is composed of the security level *High*, the compartment *Software* and the group *Europe*. Later, the function is assigned to the table *EconomicOperations* (2).

The application of labeling functions is very useful for defining the security attributes of rows as well as for implementing security constraints. Nevertheless, sometimes labeling functions are not enough and it is necessary to specify more complex conditions. OLS10g provides the possibility of defining SQL predicates together with security policies. Both labeling functions and SQL predi-

cates will be especially important when implementing secure DWs.

6.1.2. Oracle VPD

Oracle VPD, which is also known as *fine-grained access control (FGAC)* and *row-level security*, provides an additional mechanism for enforcing complex access control rules in Oracle [54,55]. This technology restricts access to specific rows in a table. The result is, therefore, that any individual user sees a completely different set of data (a different view of the database), which consists only of the data that a particular user is authorized to see. VPD can process stored functions that use data of the application context (e.g. the name of the user) to generate strings that are dynamically added as a *where* clause predicate to each query that a user performs. This clause qualifies a particular set of rows within the table. Then, Oracle rewrites dynamically the query by appending these predicates. This process is composed of two steps:

- *Definition of a policy function*: We have to use PL/SQL to implement the function that enforces a FGAC rule. For instance, function in Listing 3 (1) enforces the access control rule "the owner of Table 1 in *Schema1* can access all information, but any other user can only access rows of that table in which the name of the user is included in the value of the column *AuthorizedUsers*".
- *Definition of a policy*: Policies are defined with the DBMS_RLS package, and this package specifies when and how functions that have been defined under VPD are applied to the user activities. According to Listing 1 (2), when a user executes a query, the function that has been defined in Listing 1 (1), will be triggered, and the string "user in AuthorizedUsers" (or null string if the user is the owner of the table) will be returned, and then automatically added to the initial query.

```
(1)   CREATE FUNCTION WhichBusiness (TypeBusiness: VarChar) Return
                      LBACSYS.LBAC_LABEL
      As MyLabel varchar2(80);
      Begin
        If TypeBusiness='Electricity' then MyLabel := 'L:ELEC:E';
                               else MyLabel := 'H:SOFT:E';
        end if;
        Return TO_LBAC_DATA_LABEL('MyPolicy', MyLabel);
      End;


(2)   APPLY_TABLE_POLICY ('MyPolicy', 'EconomicOperations',
                      'Scheme', , 'WhichBusiness')
```

Listing 2. Labeling Functions.

```
(1)   CREATE OR REPLACE FUNCTION SelectView
      {
        function_schema in varchar2,
        function_table in varchar2
      }
        return varchar2 as string_return varchar(2000);
      begin
        if (function_schema = user) then string_return := null;
        else
            string_return := user || 'in AuthorizedUsers';
        end if;
        return string_return,
      end;

(2)   begin
      dbms_rls.add_policy (
          object_schema      => 'Schema1',
          object_name        => 'Table1',
          policy_name        => 'SecurePolicy',
          function_schema    => 'SecurityManager'
          policy_function    => 'Select_View',
          statement_types    => 'Select',
      );
      end;
```

Listing 3. Virtual Private Database.

Therefore, when a user executes a query, one or several access control rules can be transparently applied. Similarly, other functions can be implemented to be executed when the user inserts, updates or deletes any data within the database. Moreover, a policy can be applied to multiple tables, and a single policy function can be used by any number of policies.

One or several VPD policies and functions can perfectly coexist with an OLS policy. OLS defines a mechanism for controlling access to the information, according to the value of security labels that have been defined for each row, and VPD can complement the access control mechanism by enforcing other access control rules that are not based on these labels.

6.1.3. Oracle FGA

Oracle provides a complex and complete set of auditing functionalities. Moreover, Oracle FGA enables us to monitor data access based on content. This fact allows us to be selective with the information to be audited, and therefore to save resources, only auditing the information that is really necessary.

FGA provides an interface for creating policies to audit select, update, insert and delete statements in tables and views according to an audit condition. If any row returned from a query (or the row that has been inserted, modified or deleted) matches the audit condition, then an audit event entry is inserted into the fine-grained audit trail. This entry includes information such us the session identification, a timestamp, the user identification, the object schema, the SQL text, etc. Listing 1 shows a very straightforward example of auditing policy in which an entry into the fine-grained audit trail will be recorded if a user inserts, updates or deletes a row in the table *PayRoll* with a salary greater than 100,000. (Listing 4).

6.1.4. Implementing secure DWs with oracle DBMS

In this subsection, we will sketch the most important issues regarding how to implement secure DWs modeled with our UML profile on Oracle 10g using OLS, VPD, and FGA. We have chosen these products because they are part of one of the most important DBMSs, which allow us to implement secure databases. Unfortunately, as our conceptual model for secure DWs is richer than the secure model of Oracle 10g, there is not a completely direct and automatic mapping between our conceptual model and Oracle. However, this is absolutely normal when translating a richer general model (PIM, Platform Independent Model, in Model-Driven Architecture (MDA) [56] terminology) into a more specific one (PSM, Platform-Specific Model). For instance, our general model considers security at the attribute level, and OLS10g only supports it at the row level (a coarser granularity access). However, for these secure aspects that do not have a direct mapping on Oracle 10g, we will use other mechanisms such as functions, proceedings and triggers to represent them. This way, we can assure that the main secure aspects considered in a conceptual MD modeling by using our UML approach have their corresponding implementation.

```
begin
    dbms_fga.add_policy(
        object_schema       => 'Employee',
        object_name         => 'PayRoll',
        policy_name         => 'MaxSalary',
        audit_condition     => 'SALARY > 100000'
        audit_column        => 'EMPLOYEE',
        statement_types     => 'insert, update, delete'
        enable              => true,
    );
end;
```

Listing 4. Fine-Grained Auditing.

With regard to the particularities of Oracle, the transformation between the conceptual DW model and this DBMS is as follows:

(1) *Definition of the DW schema*: The structure of the DW, composed by fact, dimension and base classes, including fact attributes, descriptors, dimension attributes, and aggregation, generalization and completeness associations, must be translated into a relational schema. This transformation is similar to the common transformation between conceptual and logical models (see, for example [57–59]), and it can be performed in a guided, semi-automatic way.

(2) *Adaptation of the new data types of the UML extension*: All new data types (Level, Levels, Role, Compartment, Privilege, and Attempt) are perfectly supported by Oracle.

(3) *Definition of the user profile*: The profile of each user will be stored in the table *UserProfile*, which must be defined according to class *UserProfile* that was designed in analysis-time.

(4) *Definition of the OLS security policy and its default options*: Different security policies can be defined, but we consider that an OLS database created using this methodology should have the options that are shown in Listing 5. The name of the column that stores the sensitive information in each table, which is associated with the security policy, is *SecurityLabel*. The option *HIDE* indicates that the column *SecurityLabel* will be hidden, so that users will not be able to see it in the tables. The option *CHECK_CONTROL* forces the system to check that the user has reading access when he or she introduces or modifies a row. The option *READ_CONTROL* causes the enforcement of the read access control algorithm for *SELECT*, *UPDATE* and *DELETE* operations. Finally, the option *WRITE_CONTROL* causes the enforcement of the write access control algorithm for *INSERT*, *DELETE* and *UPDATE* operations.

(5) *Specification of the valid security information in the security policy*: Listing 6 shows the definition of the security levels that will be valid for our running example (which was initially defined in Fig. 5(b)) and Listing 7 specifies the user roles tree that we defined for this database (which was initially defined in Fig. 5(b)). In this example there are no compartments, but these could be also defined at this point.

(6) *Creation of the authorized users and assigning their authorization information*: In Listing 8, the user *User1* is defined by the following information: Max security level *T*, default security level *S*, minimum security level *S*, read access groups *Health*, write access groups *Health*, and default groups *Health*. Although it is not usual to assign special privileges to users, we can see how this is possible.

(7) *Definition of the security information for tables through labeling functions*: The manner of

```
SET_LEVELS('SALAPolicy', 'User1', 'TS', 'S', 'S')
SET_GROUPS('SALAPolicy', 'User1', 'HE', 'HE', 'HE')
SET_USER_PRIVS('SALAPolicy', 'User1', 'FULL, WRITEUP,
                    WRITEDOWN, WRITEACROSS')
```

Listing 5. Security Policy Definition.

```
CREATE_LEVEL('HospitalPolicy', 1000, 'C', 'Confidential')
CREATE_LEVEL('HospitalPolicy', 2000, 'S', 'Secret')
CREATE_LEVEL('HospitalPolicy', 3000, 'TS', 'TopSecret')
```

Listing 6. User Roles Definition.

```
CREATE_GROUP('HospitalPolicy', 1, 'H', 'HospitalEmployee')
CREATE_GROUP('HospitalPolicy', 2, 'HE', 'Health', 'H')
CREATE_GROUP('HospitalPolicy', 3, 'D', 'Doctor', 'HE')
CREATE_GROUP('HospitalPolicy', 4, 'N', 'Nurse', 'HE')
CREATE_GROUP('HospitalPolicy', 5, 'NHE', 'NonHealth', 'H')
CREATE_GROUP('HospitalPolicy', 6, 'M', 'Maintenance', 'NHE')
CREATE_GROUP('HospitalPolicy', 7, 'A', 'Administrative', 'NHE')
```

Listing 7. User Roles Definition.

```
SET_LEVELS('SALAPolicy', 'User1', 'TS', 'S', 'S')
SET_GROUPS('SALAPolicy', 'User1', 'HE', 'HE', 'HE')
SET_USER_PRIVS('SALAPolicy', 'User1', 'FULL, WRITEUP,
               WRITEDOWN, WRITEACROSS')
```

Listing 8. User Definition.

```
CREATE FUNCTION Function1( ) Return LBACSYS.LBAC_LABEL
As  MyLabel varchar2(80);
Begin
   MyLabel := 'S::HE' ;
   Return TO_LBAC_DATA_LABEL('HospitalPolicy', MyLabel);
End;
APPLY_TABLE_POLICY ('HospitalPolicy', 'Patient', 'Scheme', 'Function1')
```

Listing 9. Labeling Functions Definition. Constant Security Information.

```
CREATE FUNCTION Function2 (Adm: Varchar2(20), Cost: Number)
                     Return LBACSYS.LBAC_LABEL
As MyLabel varchar2(80);
Begin
   Diag := select Diagnosis_Group from Diagnosis where AdmissionCode = Adm
   If (Diag = 'Cancer' or Diag = 'AID') or (Cost > 10000)  then MyLabel := 'TS::HE,A';
                              else MyLabel := 'S::HE,A';
   end if;
   Return TO_LBAC_DATA_LABEL('HospitalPolicy', MyLabel);
End;
APPLY_TABLE_POLICY ('HospitalPolity', 'Admission', 'Scheme', 'Function2')
```

Listing 10. Labeling Function Definition. Security Constraints.

assigning the security information (that has been defined for each class in the MD conceptual modeling) to each row, once it has been inserted, is through labeling functions. For each class CL_i in the MD model, if there are no security constraints associated with it (meaning that the security information for all rows will be the same), a labeling function has to be defined, always assigning the same security information to the row. Listing 9 shows a labeling function definition and the command by which it is assigned to a table, associated to the class *Patient* and as defined in Fig. 6. If we have a look at the class diagram, the security level that has been specified is *Secret* and the user group is *Health*. So *Function1* always creates the same security label (*S::HE*), which will be assigned to each instance of that table. We can see that these functions are easily re-usable for several tables with the same security classification. Although OLS allows us to define security labels for rows, it is not possible to consider security at attribute level. If there is security information

for attributes, therefore, this information will have to be discarded.

(8) *Implementation of the SIARs through labeling functions*: For each class that has been specified in the MD conceptual model and which has one or several SIARs (SIARs) assigned a labeling function that implements the constraint has to be defined. Listing 10 shows the implementation of the security constraints that have been specified in Fig. 6. *Function2* creates the security information, depending on the value of the column *Cost* and the type of *Diagnosis*. These functions are then associated with the table *Admission*.

(9) *Implementation of ARs*: We use FGA to specify these rules. The problem is that, once more, the matching between conceptual model and implementation is not perfect. FGA does not allow us to choose the audit type (all accesses, only frustrated accesses, or only successful accesses). Nevertheless, it is possible to specify an auditing condition. Listing 11 shows the implementation of the AR that was specified in Fig. 6. In this case, we do not exploit the functionality of FGA completely, since the AR

we have specified in the conceptual MD model is straightforward.

(10) *Implementation of ARs (AURs)*: We can use both VPD (if we aim at including a *where* clause) and OLS (if we aim at including a simple predicate in the application of the security policy to the table) to specify these rules. Listing 12 (1 and 2) shows the implementation of one of the AURs in Fig. 6. In this case we use VPD, and we define a function that is trivially derived from the specification of the AUR in the MD conceptual model. Then the function is associated with the corresponding table. Listing 12 (3) shows the other approach, adding a simple predicate (which implements the other AUR) when we define the OLS policy in the corresponding table.

```
Begin
    dbms_fga.add_policy(
        object_schema        => 'Hospital',
        object_name          => 'Admission',
        policy_name          => 'AllAdmissions',
        audit_column         => 'AdmissionCode',
        statement_types      => 'select',
        enable               => true
    );
end;
```

Listing 11. Audit Rules Implementation.

6.1.5. Snapshots of our prototype from oracle DBMS

In this subsection, we will provide some snapshots to show how Oracle 10g works with different secure rules that we have defined for our case study. All these snapshots are captured from the SQL Worksheet tool, a manager tool provided by Oracle to work with Oracle DBMS. Within this tool, in the upper window we introduce the SQL sentences to be executed, and in the lower window we can see the corresponding answer provided by the server.

First of all, we have created the database scheme. Then, the security policy, the security information (levels and groups), the users, the labeling functions, the predicates and the VPD functions have been defined by means of the Oracle Policy Manager, according to the listings shown in the previous subsection. Finally, in Table 6 we will show some inserted rows that will allow us to show the benefits of our approach.

Although the SecurityLabel column is *hide*, we have shown it for the Admission table, so that we can appreciate that the label for each row is correctly defined, according to the security information and constraints that have been specified in Fig. 6.

For the sake of simplicity, we have only defined three users: Bob, who has 'topSecret' security level,

```
(1)  CREATE OR REPLACE FUNCTION HealthArea
     {  function_schema in varchar2,
        function_table in varchar2  }
        return varchar2 as string_return varchar(2000);
     begin
        AreaUser := select WorkingArea from UserProfile where Name = user
        for adm_rec in   {
           select AdmissionCode from Admission, Diagnosis where
               Admission.AdmissionCode = Diagnosis.AdmissionCode and
               Diagnosis.HealthArea = AreaUser
        } loop
           string_return := string_return || ',' || adm_rec.AdmissionCode;
          end loop;
        string_record := ltrim(string_record, ',');
        string_record := 'AdmissionCode IN (' || string_record || ')';
         return string_return,
     end;
(2)  begin
     dbms_rls.add_policy (
         object_schema        => 'Schema',
         object_name          => 'Admission',
         policy_name          => 'HospitalPolicy',
         function_schema      => 'SecurityManager'
         policy_function      => 'HealthArea',
         statement_types      => 'Select');
     end;

(3)  APPLY_TABLE_POLICY ('HospitalPolicy', 'Patient', 'Scheme', , 'Function1', 'OR
     SYS_CONTEXT ('USERENV', 'SESSION_USER') = patient_name'
```

Listing 12. Authorization Rules.

Table 6
Rows inserted into the admission, patient, diagnosis and userprofile tables

ADMISSION				
TYPE	COST	SSN	CODE DIAGNOSIS	SECURITY LABEL
Primary	150000	12345678	S1.1	TS::HE, A
Secondary	180000	12345678	S1.2	TS::HE, A
Primary	8000	98765432	D1.1	S::HE, A
Primary	90000	98765432	C1.1	TS::HE, A
Primary	9000	12345678	D1.2	S::HE, A

PATIENT					
SSN	NAME	DATE OF BIRTH	ADDRESS	CITY NAME	CITY POPULATION
12345678	James Brooks	12/10/84	3956 North 46 Av.	Florida	15982378
98765432	Jane Ford	10/02/91	2005 Harrison Street	Florida	15982378

DIAGNOSIS					
CODE DIAGNOSIS	DESCRIPTION	HEALTH AREA	VALID FROM	VALID TO	DIAGNOSIS 0GROUP
C1.1	Skin Cancer	Dermatology	01/01/00	01/01/10	Cancer
C1.2	Cervical Cancer	Gynecology	12/10/04	12/10/14	Cancer
D1.1	Diabetes during pregnancy	Gynecology	07/11/03	01/11/13	Diabetes
D1.2	Other diabetes types	Endocrinology	12/12/00	12/12/10	Diabetes
S1.1	Symptomatic infection VIH	Internal medicine	10/11/00	10/11/10	AIDS
S1.2	AIDS related-complex	Internal medicine	11/11/01	11/11/11	AIDS

USERPROFILE					
USER CODE	NAME	CITIZENSHIP	HOSPITAL	WORKING AREA	DATE CONTRACT
P000100	James Brooks	Canadian	USA Medical Center		
H000001	Bob Harrison	United States	USA Medical Center	Gynecology	10/12/87
H000002	Alice Douglas	United States	USA Medical Center	Dermatology	11/11/80

and who plays 'HospitalEmployee' role; Alice, who has 'Secret' security level, and who plays 'Administrative' role; James, who is a special user because he is a patient (so he will be only able to access his own information and nothing else).

In order to illustrate how the security specifications that we have defined in the conceptual MD modeling (see Fig. 6) are enforced in Oracle, we have considered two different scenarios. In the first one we have not implemented the necessary functions to enforce the security rule that is defined in note 4 of Fig. 6. As it can be observed in Fig. 8, the first query, which is performed by Bob (as he has 'topSecret' security level and 'Health' role), shows all tuples in the database. On the other hand, the second query, which is performed by Alice (as she has 'Secret' security level and "Administrative" role), does not show information of patients whose diagnosis is Cancer or AIDS (specified in note 1 of Fig. 6) or whose cost is greater than 10.000 (specified in note 2 of Fig. 6).

In the second scenario, we have implemented the security rule that is defined in note 4 of Fig. 6. As we can observe in Fig. 9, the first query, which is performed by Bob, shows all rows in which note 4 of Fig. 6 is fulfilled (that is to say, when the health area of the patient is the same as the working area of

Bob). In the second query, which is performed by James, only his own information is shown (see note 5 of Fig. 6), hiding the patient information of any other person receiving health care within the same system (Fig. 9).

In conclusion, one of the key advantages of the overall approach presented in this paper is that general and important secure rules for DWs that are specified by using our conceptual modeling approach can be directly implemented into a commercial DBMS such as Oracle 10g. This way, instead of having partial secure solutions for certain and specific nonauthorized accesses, we deal with a complete comprehensive approach for designing secure DWs from the first stages of a DW project. Finally, as we carry out the corresponding transformations throughout all stages of the design, we can assure that the secure rules implemented into any DBMS correspond to the final user requirements captured in the conceptual modeling phase.

6.2. Implementing secure DWs with OLAP tools

One of the two key advantages of our approach is that (i) it is based on user roles (that are supported by most commercial OLAP tools), and (ii) all security measures on data are defined on a
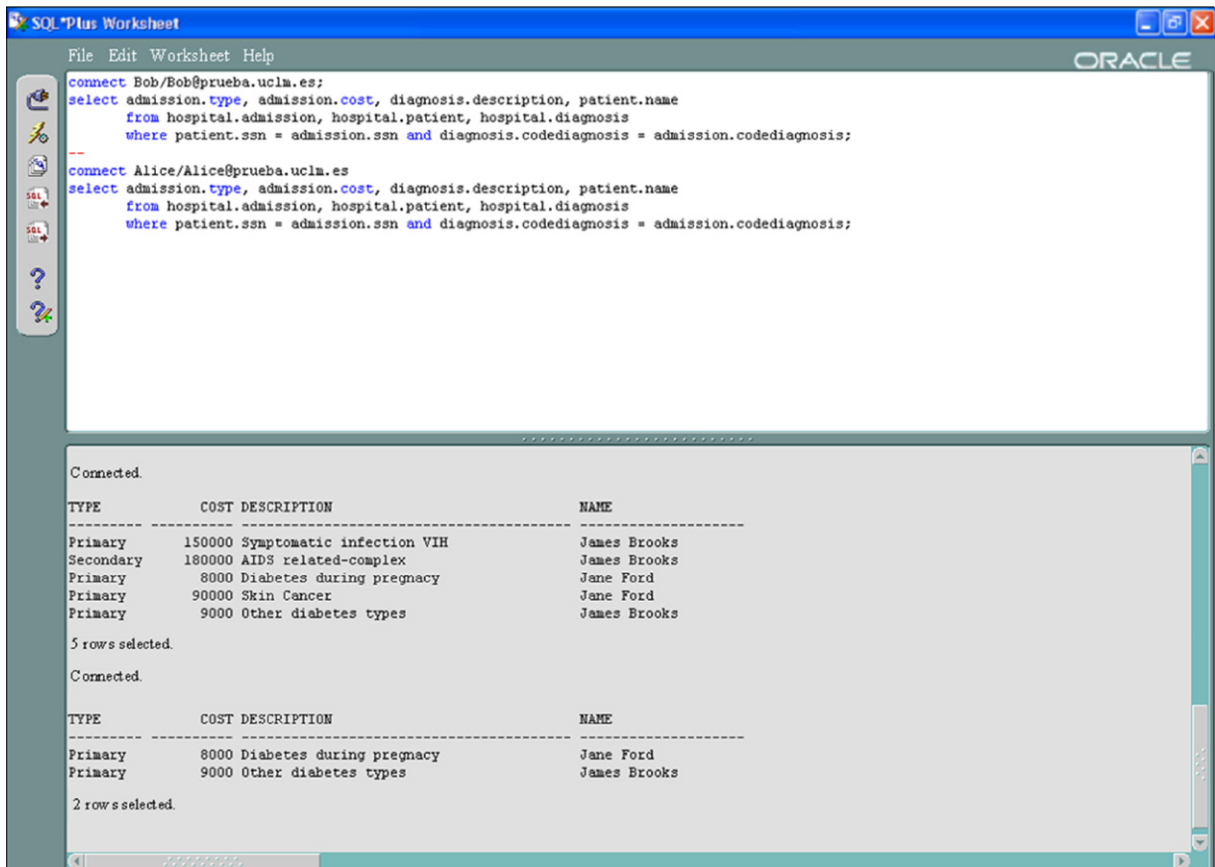
Fig. 8. First scenario.

conceptual MD model. This means that security rules are specified for facts, dimensions, hierarchy levels and so on. On the other hand, most OLAP tools allow us to specify security measures for different final users and reports (i.e. cubes or MD views). OLAP tools are usually based on user roles, to specify which user or group of users can access, read or execute a certain report, cube or MD view [39]. Once the different user roles have been defined, OLAP tools allow us to specify security measures based on the MD model in terms of hiding hierarchy levels, denying a drill-down on a certain level (for a specific user roles) and so on.

Therefore, most of our security measures and rules can be implemented by using an OLAP tool as both OLAP tools and our approach are based on the MD paradigm and user roles. As previously shown for Oracle DBMS, some security measures can be directly implemented on certain OLAP tools such as Microsoft SQL Analysis Server 2000, Microstrategy, Cognos Powerplay or Oracle Discoverer. The only issue we have to face up with is to

find out the specific mechanisms (views, reports, hybrid, etc.) which the specific OLAP tool implements.

For example, Oracle Discoverer allows us to specify which user roles can gain access to certain business areas (MD views or models), and then for every report (cube) we can specify certain rules for specific role of users by defining conditions on the users defined on the table ALLUSERS. In doing this, when a certain user or group of users specify a new cube, they can only use the dimensions, classification hierarchies and facts that they are allowed to by our security measures defined in our conceptual modeling. In Fig. 10, a snapshot from Oracle Discoverer shows that a user with the Administrative role can only include the Admission fact and the Patient dimension in his/her cube. This is because in the definition of our security measures in our case study (see Fig. 6), users with the Admin role cannot access to the Diagnosis dimension.

However, other security measures represented in our approach need further research as they do not
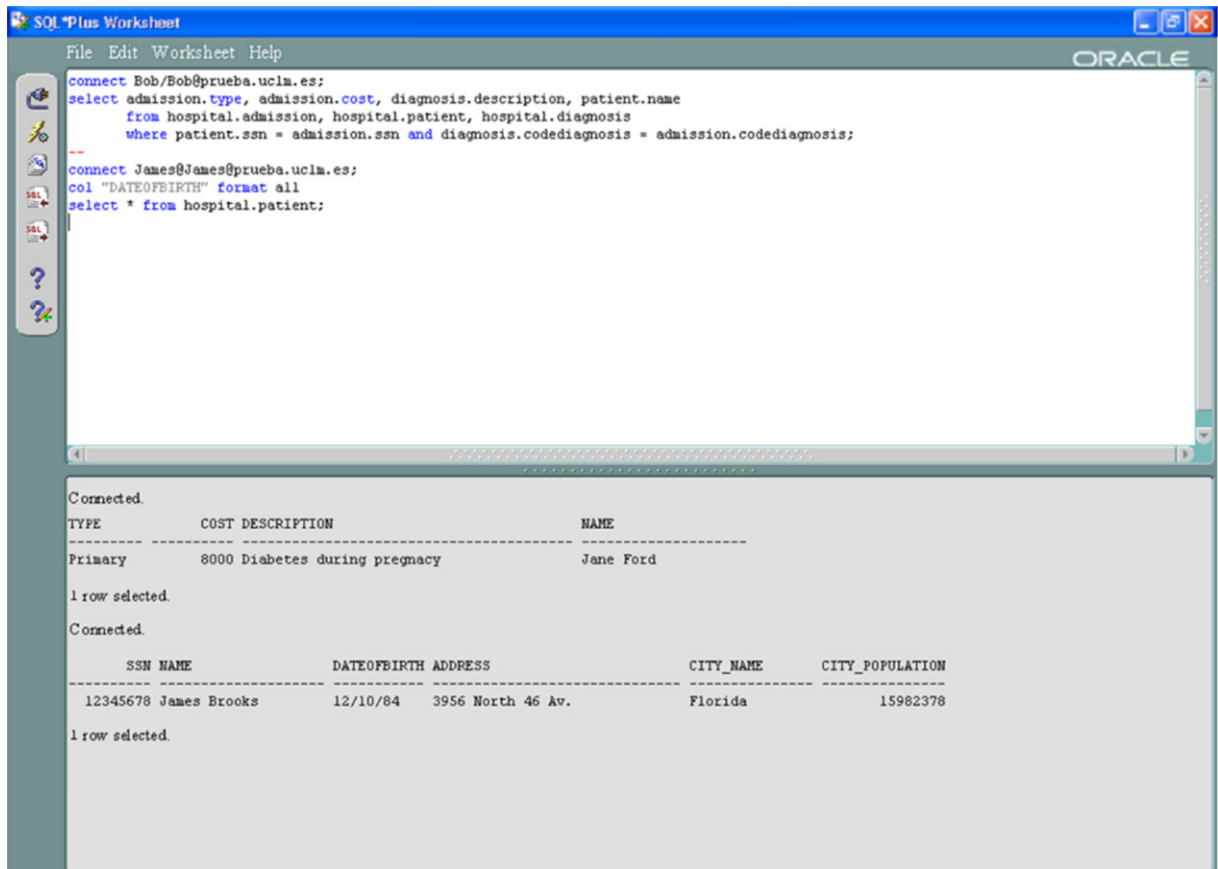
Fig. 9. Second scenario.

have a direct implementation in OLAP tools. This is mainly due to the higher power of expressiveness of the conceptual MD modeling approach which is our basis, since some properties such as degenerated dimensions or many-to-many relationships between facts and dimensions do not have the direct implementation in OLAP tools we have just mentioned, instead some transformations need to be carried out.

6.3. Automating the secure DW design and implementation process

A CASE prototype has been developed to support the conceptual design of secure DWs. This prototype has been integrated into Rational Rose, and it uses the Rose Extensibility Interface to define the specific Add-in for our proposal. An add-in is a collection of some combination of the following: main menu items, shortcut menu items, custom specifications, properties (UML tagged values),

data types, UML stereotypes, online help, context-sensitive help and event handling. In other words, any UML extension (profile) can be programmed to be used by Rational Rose.

We had previously programmed an Add-in that allowed us to specify DW by using our previous UML profile for DWs [8,9], extended to introduce the profile for designing secure DWs presented in this paper. In Fig. 11 we can see a screenshot from Rational Rose that shows the definition of a dimension from the running example used throughout the paper. Some comments have been added to the screenshot in order to highlight some important elements: new buttons added to Rational Rose, the stereotyped attributes or the stereotyped classes. We can also see the MD_Validate tool option in a dark shade; this option runs a Rose script that validates the correctness of the specified schema by checking all the constraints.

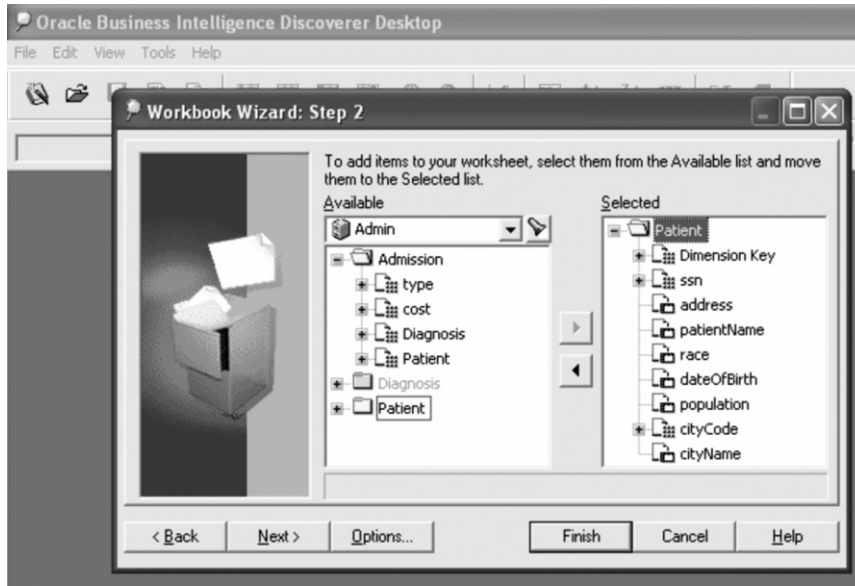Thus, we are able to integrate security measures into the MD conceptual model. Again, all the

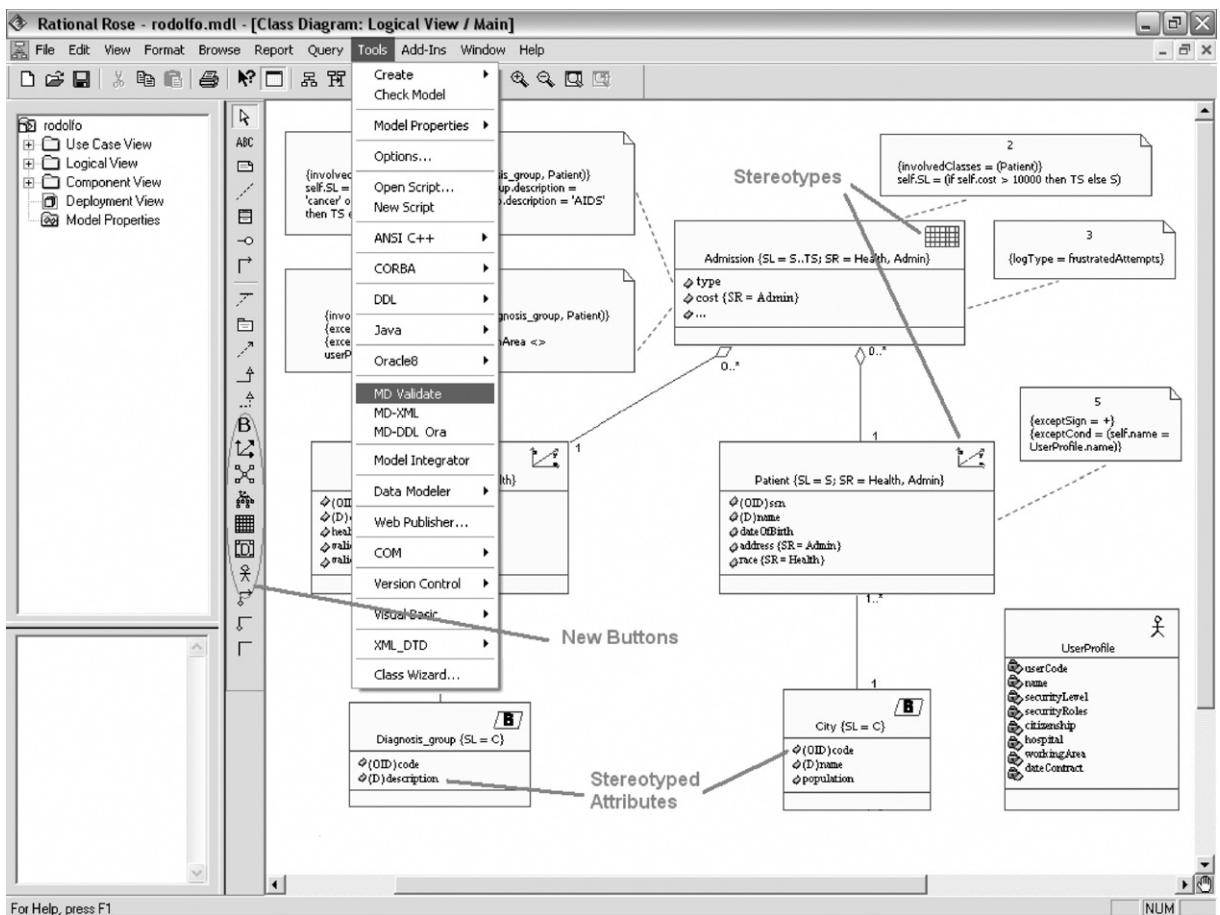Fig. 10. Oracle Discoverer scenario.

Fig. 11. Rational Rose Prototype.

required well-formedness rules are properly checked to validate the correctness of this model. This Add-in can generate the corresponding transformation from a conceptual model into a logical schema (e.g. star scheme), and then into a particular technology (e.g. Oracle). However, the automatic generation of the associated security measures has not been implemented yet in a totally automatic way.

An efficient semi-automatic transformation of almost all elements of the secure conceptual DW model is possible, generating Oracle code from the Rational metadata structures of the conceptual secure model, if we carry out the steps presented before. In particular,

(1) the generation of the database scheme specification has already been implemented, and it just needs some *design* decisions, and it can be performed automatically,

(2) new data types can be directly defined in Oracle,

(3) the definition of the user profile can be defined trivially, since it is a simple class that can be transformed into a table,

(4) the security policy has been defined by default, but it can be modified by the designer,

(5) the valid security information (security levels, user roles and compartments) can also be transformed automatically, since they are concepts that fit into the OLS structures perfectly,

(6) the definition of the data of each user according to the user profiles and the user classification criteria has to be performed manually,

(7) the definition of security information for tables can be performed automatically by building simple labeling functions, inserting metadata coming from the conceptual model,

(8) the implementation of SIARs can be performed automatically, since a straightforward OCL expression has to be transformed into a PL/SQL condition,

(9) the transformation of ARs needs to be performed manually, since there is no a direct equivalence in Oracle, and

(10) the implementation of AURs also has to be done manually, but some recommendations are offered above. A verification activity is performed by the prototype, checking the well formedness rules, but after the complete transformation, a test and validation activity should be carried out, checking the fulfillment of all security specifications in the secure conceptual MD model.

7. Conclusions and future work

In this paper, we have presented an extension of the UML that allows us to model secure DWs at the conceptual level. This UML extension contains the needed stereotypes, tagged values and constraints for a complete and powerful secure MD modeling. These new elements allow us to specify important security measures such as security levels on data, compartments and user roles on the main elements of an MD modeling such as facts, dimensions and classification hierarchies. We have used the OCL to specify the constraints attached to these new defined elements, thereby avoiding an arbitrary use of them. To the best of our knowledge, this is the first approach which integrates the modeling of security measures and the MD modeling of DWs together into the same approach. One of the best advantages of our proposal is that we model secure DWs from a conceptual perspective which is totally independent of implementation details. However, in order to show the benefits of our proposal, we have also sketched out how to implement a secure MD model designed with our approach into a commercial DBMS. The main relevant advantage of this approach is that it uses the UML, a widely accepted object-oriented modeling language, which saves developers from learning a new model and its corresponding notations for specific MD modeling. Furthermore, the UML allows us to represent some MD properties that are hardly considered by other conceptual MD proposals. Finally, we have also outlined the modeling prototype we use to model secure DWs when following our approach.

Our work for the immediate future is to extend the implementation issues presented in this paper, to allow us to use the security aspects considered when querying an MD model from OLAP tools, which includes an in-depth study of the different combinations of dimensions, measures, hierarchies and so on. Furthermore, one piece of work in the immediate future is to propose an automatic generation of conceptual schemas, considering security placed in logical schemas automatically within the framework of MDA, which will allow us to univocally specify the required set of transformations. Moreover, we also plan to extend the set of privileges considered in this paper (i.e. read) to make it possible to specify security aspects in the

crucial ETL processes for DWs, thereby considering other operations such as delete, insert and update. However, this future work will lead us towards solving conflicts between security rules defined both for data sources and for the DW itself.

Acknowledgements

References

[1] G. Dhillon, J. Backhouse, Information system security management in the new millennium, Commun. ACM 43 (7) (2000) 125–128.

[2] P. Devanbu, S. Stubblebine, Software engineering for security: a roadmap, in: A. Finkelstein (Ed.), The Future of Software Engineering, ACM Press, New York, 2000, pp. 227–239.

[3] E. Ferrari, B. Thuraisingham, Secure database systems, in: M. Piattini, O. Díaz (Eds.), Advanced Databases: Technology Design, Artech House, 2000.

[4] A. Hall, R. Chapman, Correctness by construction: developing a commercial secure system, IEEE Software 19 (1) (2002) 18–25.

[5] L. Chung, B. Nixon, E. Yu, J. Mylopoulos, Non-Functional Requirements in Software Engineering, Kluwer Academic Publishers, Boston/Dordrecht/London, 2000.

[6] Directive 95/46/CE of the European Parliament and Council, dated October 24th, about People protection regarding the personal data management and the free circulation of these data. DOCE no. L281, 23/11/1995, P.0031-0050. 1995.

[7] J. Trujillo, M. Palomar, J. Gómez, I.Y. Song, Designing data warehouses with OO conceptual models, IEEE Computer, special issue on Data Warehouses (34) (2001) 66–75.

[8] S. Luján-Mora, J. Trujillo, I.Y. Song, Multidimensional modeling with UML package diagrams, In Proceedings of International Conference on Conceptual Modeling—ER 2002, Springer. LNCS 2503, Tampere, Finland, 2002.

[9] S. Luján-Mora, J. Trujillo, I.Y. Song, Extending the UML for multidimensional modeling, In Proceedings of 5th International Conference on the Unified Modeling Language (UML 2002), Springer. LNCS 2460, Dresden, Germany, 2002. pp. 290–304.

[10] E. Fernández-Medina, J. Trujillo, R. Villarroel, M. Piattini, Extending the UML for designing secure data warehouses, In Proceedings of International Conference on Conceptual Modeling (ER 2004), Springer, Shangai, China, 2004.

[11] P. Samarati, S. De Capitani di Vimercati, Access control: policies, models, and mechanisms, in: R. Focardi, R. Gorrieri (Eds.), Foundations of Security Analysis and Design, Springer, Berlin, 2000, pp. 137–196.

[12] R. Sandhu, E. Coyne, H. Feinstein, C. Youman, Role-based access control models, IEEE Computer 29 (2) (1996) 38–47.

[13] R. Sandhu, D. Ferraiolo, R. Kuhn, The NIST model for role-based access control: towards a unified standard, Proceedings of fifth ACM Workshop on Role-Based Access Control, Berlin, Germany, 2000, pp. 47–63.

[14] J. Levinger, Oracle label security. Administrator's guide 10 g. Release 1 (10.1) http://www.csis.gvsu.edu/GeneralInfo/Oracle/network.920/a96578.pdf, 2003.

[15] S. Cota, For certain eyes only, DB2 Mag. 9 (1) (2004) 40–45.

[16] R. Kimball, M. Ross, The Data Warehousing Toolkit, Second ed, Wiley, New York, 2002.

[17] M. Blaschka, C. Sapia, G. Höfling, B. Dinter, Finding your way through Multidimensional Data Models, Proceedings of 9th Intl. Conference on Database and Expert Systems Applications (DEXA'98), Springer-Verlag, LNCS, 1460, Vienna, Austria, 1998. pp. 198–203.

[18] A. Abelló, J. Samos, F. Saltor, A Framework for the Classification and Description of Multidimensional Data Models, In Proceedings of 12th International Conference on Database and Expert Systems Applications (DEXA'01), Springer-Verlag LNCS 2113, Munich, Germany, 2001, pp. 668–677.

[19] C. Sapia, On modeling and predicting query behavior in OLAP systems, In Proceedings of International Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany, 1999. pp. 1–10.

[20] C. Sapia, M. Blaschka, G. Höfling, B. Dinter, Extending the E/R Model for the Multidimensional Paradigm, In Proceedings of 1st International Workshop on Data Warehouse and Data Mining (DWDM'98), Springer, 1552, Singapore, 1998, pp. 105–116.

[21] N. Tryfona, F. Busborg, J. Christiansen, starER: a conceptual model for data warehouse design, In Proceedings of ACM 2nd International Workshop on Data Warehousing and OLAP (DOLAP'99), ACM, Missouri, USA, 1999. pp. 3–8.

[22] M. Golfarelli, D. Maio, S. Rizzi, The dimensional fact model: a conceptual model for data warehouses, Int. J. Cooper. Inform. Syst. 7 (2–3) (1998) 215–247.

[23] M. Golfarelli, S. Rizzi, A methodological framework for data warehouse design, In Proceedings of 1st International Workshop on Data Warehousing and OLAP (DOLAP'98), Maryland, USA, 1998, pp. 3–9.

[24] B. Husemann, J. Lechtenborger, G. Vossen, Conceptual Data Warehouse Design, (Ed.), In Proceedings of the 2nd. International Workshop on Design and Management of Data Warehouses, Technical University of Aachen (RWTH), 2000. pp. 3–9.

[25] A. Abelló, J. Samos, F. Saltor, YAM2 (Yet another multidimensional model): an extension of UML, (Ed.), International

Database Engineering & Applications Symposium (IDEAS 2002), IEEE Computer Society, 2002, pp. 172–181.

[26] N. Binh, A. Tjoa, Conceptual multidimensional data model based on object-oriented metacube, Proceedings of ACM Simposion on Applied Computing, Las Vegas, Nevada, USA, 2001, pp. 295–300.

[27] N. Binh, A. Tjoa, R. Wagner, An object oriented multi-dimensional data model for OLAP, In Proceedings of First International Conference on Web-Age Information Management, Springer, Shanghai, China, 2000.

[28] T. Priebe, G. Pernul, Metadaten-gestüzer data-warehouse-entwurf mit ADAMTed UML, In Proceedings of 5th Internationale Tagung Wirtschaftsinformatik, Physica, Augsburg, 2001.

[29] OMG, Object management group. UML 2.0. OCL specification, OMG document ptc/03-10-14. 2004.

[30] G.W. Smith, Modeling security-relevant data semantics, IEEE Trans Software Eng. 17 (11) (1991) 1195–1203.

[31] G. Pernul, W. Winiwarter, A. Tjoa, The entity-relationship model for multilevel security, In Proceedings of 12th International Conference on the Entity-Relationship Approach (ER 1993), Arlington, TX, USA, 1993. pp. 166–177.

[32] D. Marks, P. Sell, B. Thuraisingham, MOMT: a multi-level object modeling technique for designing secure database applications, J. Object-Oriented Program. 9 (4) (1996) 22–29.

[33] J. Jürjens, Towards development of secure systems using UML, In Proceedings of International Conference on the Fundamental Approaches to Software Engineering (FASEi-TAPS), Springer, 2001.

[34] J. Jürjens, UMLsec: extending UML for secure systems development, in: J. Jézéquel, H. Hussmann, S. Cook (Eds.), UML 2002—The Unified Modeling Language, Model Engineering, Concepts and Tools, Springer, Berlin, 2002, pp. 412–425.

[35] T. Lodderstedt, D. Basin, J. Doser, SecureUML: A UML-based modeling language for model-driven security, In Proceedings of The Unified Modeling Language Conference, Springer. LNCS 2460, Dresden, Germany, 2002. pp. 426–441.

[36] E. Fernández-Medina, M. Piattini, Designing secure database for OLS, In Proceedings of Database and Expert Systems Applications: 14th International Conference (DEXA 2003), Springer. LNCS 2736, 2736, Prague, Czech Republic, 2003. pp. 886–895.

[37] E. Fernández-Medina, M. Piattini, Extending OCL for secure database design, In Proceedings of International Conference on the Unified Modeling Language (UML 2004), Springer, LNCS 3273, Lisbon, Portugal, 2004, pp. 380–394.

[38] J. Warmer, A. Kleppe, The Object Constraint Language, Second Ed., Getting Your Models Ready for MDA, Addison Wesley, 2003.

[39] T. Priebe, G. Pernul, Towards OLAP security design—survey and research issues, In Proceedings of 3rd ACM International Workshop on Data Warehousing and OLAP (DOLAP'00), Washington DC, USA, 2000, pp. 33–40.

[40] T. Priebe, G. Pernul, A pragmatic approach to conceptual modeling of OLAP security, In Proceedings of 20th Int. Conference on Conceptual Modeling, Springer, LNCS 2224, Yokohama, Japan, 2001, pp. 311–324.

[41] N. Katic, G. Quirchmayr, J. Schiefer, M. Stolba, A. Min Tjoa, A prototype model for data warehouse security based on metadata, In Proceedings of 9th International Workshop on Database and Expert Systems Applications (DEXA'98).

IEEE Computer Society, 8, Vienna, Austria, 1998. pp. 300–308.

[42] A. Rosenthal, E. Sciore, View security as the basic for data warehouse security, In Proceedings of 2nd International Workshop on Design and Management of Data Warehouse, 28, Sweden, 2000, pp. 8.1–8.8.

[43] R. Kirkgöze, N. Katic, M. Stolda, A. Min Tjoa, A security concept for OLAP, In Proceedings of 8th International Workshop on Database and Expert System Applications (DEXA'97), IEEE Computer Society, Toulouse, France, 1997, pp. 619–626.

[44] L. Wang, S. Jajodia, D. Wijesekera, Securing OLAP data cubes against privacy breaches, In Proceedings of IEEE Symposium on Security and Privacy, Berkeley, California, 2004. pp. 161–178.

[45] E. Weippl, O. Mangisengi, W. Essmayr, F. Lichtenberger, W. Winiwarter, An authorization model for data warehouses and OLAP, In Proceedings of Workshop on Security in Distributed Data Warehousing, New Orleans, Louisiana, USA, 2001.

[46] M. Gogolla, B. Henderson-Sellers, Analysis of UML Stereotypes within the UML Metamodel, In Proceedings of 5th International Conference on the Unified Modeling Language—The Language and its Applications, Springer, LNCS 2460, Dresden, Germany, 2002. pp. 84–99.

[47] E.B. Fernandez, R.Y. Pan, A pattern language for security models, In Proceedings of 8th Conference on Patterns Languages of Programs (PLOP 2001), Illinois, USA, 2001.

[48] J. Conallen, Building Web Applications with UML, Object Technology Series., Addison-Wesley, Reading, MA, 2000.

[49] OMG, Object Management group: unified modeling language specification 1.5. 2004.

[50] A. Baraani-Dastjerdi, J. Pieprzyk, R. Safavi-Naini, Security in Databases: A Survey Study. Department of Computer Science, The University of Wollongong, 1996.

[51] A. Baraani-Dastjerdi, R. Safavi-Naini, J. Pieprzyk, J. Getta, A Model of Authorization for Object-Oriented Databases Based on Object Views, In Proceedings of International Conference on Deductive and Object-Oriented Databases, Springer, LNCS 1013, Singapore, 1995. pp. 503–520.

[52] E. Gün, K. Wang, An access control language for web services, In Proceedings of Seventh ACM Symposium on Access Control Models and Technologies, ACM Press, Monterey, California, USA, 2002, pp. 23–30.

[53] R. Sandhu, F. Chen, The multilevel relational data model, ACM Trans. Inform. Syst. Security (TISSEC) 1 (1) (1998) 93–132.

[54] A. Nanda, Keeping Information Private with VPD, Oracle XVIII (2) (2004) 81–84.

[55] A. Nanda, D. Burleson, Oracle privacy security auditing, RAMPANT, 2003.

[56] A. Kleppe, J. Warmer, W. Bast, MDA Explained; The Model Driven Architecture: Practice and Promise, Addison-Wesley, Reading, MA, 2003.

[57] P. Atzeni, S. Ceri, S. Paraboschi, R. Torlone, Database Systems. Concepts Language and Architectures, McGraw-Hill, New York, 1999.

[58] M. Blaha, W. Premerlani, Object-Oriented Modeling and Design for Database Applications, Prentice-Hall, Englewood Cliffs, NJ, 1998.

[59] R. Muller, Database Design for Smarties. Using UML for Data Modeling, Morgan Kaufmann Publisher, inc, San Francisco, California, 1999.