Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information and Software Technology 47 (2005) 463–477
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Designing secure databases

Eduardo Fernández-Medina*, Mario Piattini

Escuela Superior de Informática, Alarcos Research Group, University of Castilla-La Mancha, Paseo de la Universidad 4, 13071 Ciudad Real, Spain

Received 21 May 2004; revised 28 September 2004; accepted 29 September 2004
Available online 14 November 2004

Abstract

Security is an important issue that must be considered as a fundamental requirement in information systems development, and particularly in database design. Therefore security, as a further quality property of software, must be tackled at all stages of the development. The most extended secure database model is the multilevel model, which permits the classification of information according to its confidentiality, and considers mandatory access control. Nevertheless, the problem is that no database design methodologies that consider security (and therefore secure database models) across the entire life cycle, particularly at the earliest stages currently exist. Therefore it is not possible to design secure databases appropriately. Our aim is to solve this problem by proposing a methodology for the design of secure databases. In addition to this methodology, we have defined some models that allow us to include security information in the database model, and a constraint language to define security constraints. As a result, we can specify a fine-grained classification of the information, defining with a high degree of accuracy which properties each user has to own in order to be able to access each piece of information. The methodology consists of four stages: requirements gathering; database analysis; multilevel relational logical design; and specific logical design. The first three stages define activities to analyze and design a secure database, thus producing a general secure database model. The last stage is made up of activities that adapt the general secure data model to one of the most popular secure database management systems: Oracle*9i* Label Security. This methodology has been used in a genuine case by the Data Processing Center of Provincial Government. In order to support the methodology, we have implemented an extension of Rational Rose, including and managing security information and constraints in the first stages of the methodology.

Keywords: Secure databases; Database design; Unified Modeling Language; Object Constraint Language

1. Introduction

Modern society forces business to evolve, and to manage information correctly in order to achieve their objectives and survive in the digital era. Organizations increasingly depend on information systems (IS), which rely upon large databases, and these databases therefore need increasingly more quality and security [8]. Indeed, the very survival of organizations depends on the correct management, security and confidentiality of this information [14,15].

Consequently, protecting information that is stored in databases is important for companies, but at times, it is also important for individuals. This is because databases also frequently store information regarding private or personal aspects of individuals, such as identification data, medical data or even religious beliefs, ideologies, or sexual tendencies. As a result, there are laws to protect the individual's privacy, such as the European Union Directive 95/46/CE of the European Parliament and Council, which deals with the protection of personal data and its free circulation [16]. These laws tend to be very strict, imposing severe penalties for failure to comply with them. This information should then be protected against non-authorized access, thus fulfilling the existing data protection laws.

Some authors note that database protection is a serious requirement that must be carefully considered, not as an isolated aspect, but as an element present in all stages of the database life cycle [13,19,21]. Even the Information Systems Audit and Control Foundation affirms that managers have to ensure that security is considered as an integral part of the systems development life cycle process

* Corresponding author. Tel.: +34 926 29 53 00; fax: +34 926 295 354.
E-mail addresses: eduardo.fdezmedina@uclm.es (E. Fernández-Medina), mario.piattini@uclm.es (M. Piattini).
URL: http://alarcos.inf-cr.uclm.es/english/.

0950-5849/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2004.09.013

and is explicitly addressed during each phase of the process [24].

In this article, we propose a methodology to build multilevel databases, taking into consideration aspects of security (with regard to confidentiality) from the earliest stages to the end of the development process. We believe that a new methodology for designing secure databases should be an extension of a widely accepted modeling language, in order to save developers from learning a new model and its corresponding notation. Therefore, the methodology we propose extends some well-known models, such as different unified modeling language (UML) models [7], the unified process (UP) [25], and the object constraint language (OCL) [38]. This methodology allows us to create conceptual and logical models of multilevel databases, and implement them by using Oracle*9i* Label Security (OLS*9i*) [29].

The rest of the article is organized as follows: in Section 2 we present related work. Section 3 presents a summary of OLS*9i*. In Section 4 we provide an overview of the case study. A concise revision of the secure database design methodology, including subsections with details of each stage and the models and languages that have been defined is presented in Section 5. An overview of the CASE tool developed is shown in Section 6. Section 7 collects the lessons learned when applying the methodology to the case study. Finally, in Section 8, we mention some conclusions that have been drawn and future work to be carried out.

2. Related work

In spite of the fact that there is a vast amount of work related to security and databases, not much work of this integrates security into the database development process. We can classify related work as follows:

- *Database design*. Traditional database design methodologies [4,11] do not consider security. Therefore, these methodologies are not useful in developing secure databases.
- *Security design*. Security methods can be organized into three main generations: checklists, engineering, and logic transformation methods [3]. These methods are based on concepts that have appeared in parallel in the ISs methodology generations, but integration between software development and security is not achieved. Castano et al. [9] present an interesting methodological approach for designing security in databases, but they do not consider security integration within the database development process. Therefore, these security techniques are not useful in developing secure databases.
- *Integration of security in the software development process*. Many initiatives have been tested in order to attain this objective, but none of the solutions is completely satisfactory. Smith [36] proposed the Semantic Data Model to conceptually design secure databases, and Marks et al. [30] proposed the multilevel object modeling technique, which is an extension of the object modeling technique [33], in order to design multilevel databases. These two initiatives offer very interesting ideas, but they have not been completely developed. Most recently, Chung et al. [10] also insist on integrating security requirements in the design, by providing the designers with models specifying security aspects, but they do not deal with specific database issues. Once again, Hall and Chapman [22] propose different ideas for integrating security into the system development process, but they only consider database security from a cryptographic point of view. A very serious and complete proposal is [26] where UML is extended to develop secure systems. This approach is very interesting, but again, it only deals with IS in general, whilst conceptual and logical database design, and secure database implementation are not considered.

3. Oracle*9i* label security

OLS*9i* [29] is a component of version 9 of Oracle database management system (DBMS) which allows us to implement multilevel databases [35]. OLS*9i* defines labels that are assigned to the rows and users of the database. These labels contain confidentiality information for the rows, and authorization information for users. OLS*9i* defines a combined access control mechanism, considering mandatory access control (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 [34]. This combined access control imposes the rule that a user will only be entitled to access a particular row if that user is 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. 1 represents this combined access control mechanism.

Each security label contains security levels, user compartments and hierarchical user groups, but these components are different depending upon whether the label is assigned to a row or to a user. Components of labels for rows are as follows:

(a) Security levels: these denote the level of sensitivity of the information. The higher the level of security, the more sensitive is the information. Traditional security levels (originally associated with military environments) are *Unclassified*, *Confidential*, *Secret*, and *Top Secret*.
(b) User compartments: these identify sectors which will be entitled to access the row. User compartments allow us to form a horizontal classification of members of

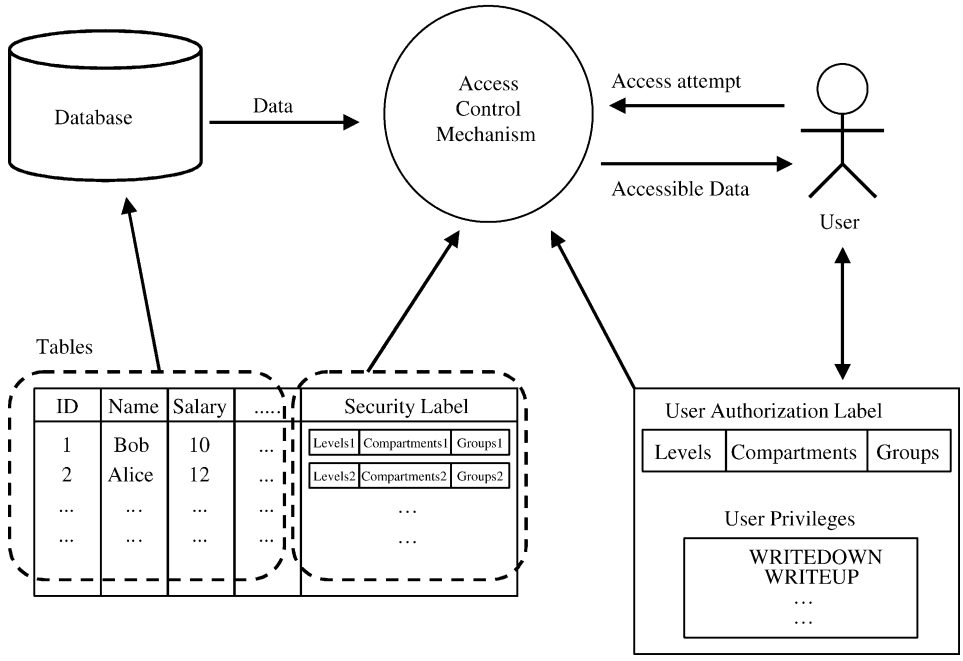

Fig. 1. Access control mechanism.

organizations. Members of a department or sector, or workers on a project may belong to the same compartment.

(c) Hierarchical user groups: these specify a set of user roles that users have to play in order to be entitled to access the row. Each organization can define a set of hierarchical roles, assign responsibilities to each role, and define who will be a member of each role. This organization is useful in the prevention of data dissemination and organizational changes.

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 (the user will not be able to read rows with a higher security level than its maximum level); minimum level, which limits write accesses (the user will not be able to insert, update or delete rows with a lower security level that its minimum level); default level, which is the level (between the maximum and minimum) in which the user is connected to the DBMS; and row level, which is the level (between the 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 mandatory access control constraints.

Access control is enforced by OLS*9i* according to the access type (read or write). The access control rules that OLS*9i* enforces in the case of read access are as follows:

(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.

The access control rules that OLS*9i* enforces in the case of write access are as follows, and in this order:

(1) The row security level has to be greater than or equal to the minimum user security level, and less than or equal to the default user security.
(2) If the row label contains some hierarchical groups, the user label has to contain at least one of these (or an ascendant), with write access permission. Moreover, the user label has to contain all the compartments that the row label contains.
(3) If the row label does not contain any hierarchical groups, the user label has to contain all the compartments that the row label contains, but with write access permission.

```
(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')
```

Fig. 2. Security policy, levels and groups definition.

All security information in OLS*9i* is in the context of a *security policy*. 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 it has been defined, we must define the valid levels, compartments and hierarchical groups in the context of this security policy.

Fig. 2 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 mandatory access control 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. Then 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 and 8).

Once a policy has been defined, it can be discretionally applied to one or more tables of the database. We also have to define the mechanisms by which the rows will be labeled. OLS*9i* 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 *Labeling Functions*, which are triggered when operations *INSERT* or *UPDATE* 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.

Fig. 3 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

```
(1) CREATE FUNCTION WhichBusiness (TypeBusiness: VarChar) Return
                    LBACSYS.LBAC_LABEL
    As MyLabel varchar2(80);
    Begin
      IfTypeBusiness='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')
```

Fig. 3. Labeling functions.

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).

Labels and privileges are manually assigned to the users.

4. Case study

In order to develop our methodology, we have used the *Action Research* method [2], applying the methodology to the redesign of a Spanish Provincial Government's database. Those involved in this process were mainly Provincial Government managers and researchers.

The database that we considered in this case study was used by an application, called System for the Accounting of the Local Administration (SALA), which had various confidentiality problems which were solved by creating a new secure design of the database. The general aim of this application was to control the budget and accounts of the Provincial Government. This application not only manages economic information about companies and individuals, but also *personal information about individuals*. If minimal security measures are taken into account, it is possible to illegally explore economic and personal information, by collecting addresses, account numbers, telephone numbers, information about economic transactions, etc. Therefore, it would be easy to discover the habits of companies and individuals, and consequently to form a profile of them. In this case, the public organization would be faced with legal responsibilities.

The system was built in 1988, and was developed by a team of 12 developers. It needed one and half years to be completely operative. The DBMS that supports the system is Ingres. The system is organized into 10 main subsystems (expense, income, financed expenses, non-budgetary operations, other resources, treasury, tax collector control, general accounting, third parties, and system administration). The database has 74 tables with an average of nine columns for each table.

In Section 5, we introduce our methodology, considering some models and specifications that have been extracted from this case study.

5. Methodology overview

As we have mentioned previously, there is no satisfactory solution to the problem of integrating security into the database development process. Therefore, our main objective is to build a complete methodology (with the necessary techniques) in order to develop secure databases. Considering this main objective, we have defined the following set of partial sub-objectives with regard to the methodology: (1) it has to be easy to learn; (2) it has to be flexible; (3) it has to be implementation-independent; (4) it has to be made specific to a particular DBMS; and (5) it has to be supported by CASE tools.

The methodology is based on the UP and on traditional database design methodologies, which are well-known in the software development community. Moreover, the methodology uses models and techniques that are based on the UML and on the OCL, so sub-objective (1) is satisfied. By using this methodology, we can build a general secure database model that can be used to implement the secure database with any DBMS used for implementing secure databases. Thus the methodology is flexible and implementation-independent. The methodology allows the implementation of secure databases in OLS9*i*, so sub-objective (4) is successful. Finally, a CASE tool has been developed to support the methodology. This tool has been integrated in Rational Rose as an add-in, so this collaborates in satisfying sub-objectives (1) and (5).

The general structure of the methodology can be observed in Fig. 4. Stages *Requirements Gathering*, *Database Analysis*, and *Multilevel Relational Logical Design*, allow us to build a general model of the secure database, and finally the stage *Specific Logical Design* adapts this general model to the particularities of OLS9*i*. The methodology is open to the integration of another *Specific Logical Design* after the *Multilevel Relational Logical Design* in order to implement the secure database using another DBMS (e.g. DB2).

In the following subsections, we present each stage in more detail, illustrating them with a portion of the case study. Additionally, we present the most important techniques and models involved in each stage of the methodology. Finally, we introduce some considerations of verification and validation.

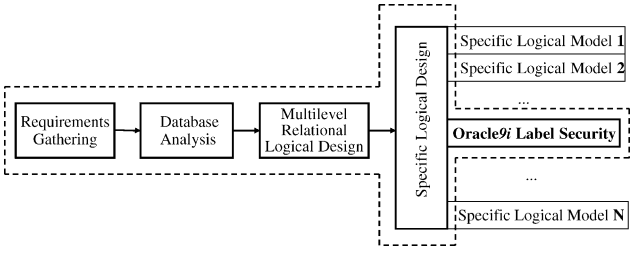

Fig. 4. Methodology stages.

5.1. Requirements gathering

As in any other development methodology, the objective of this stage is to elicit and model requirements, with the particularity that we must also consider security requirements. Several artifacts are involved in this stage (requirements catalog, business model, system glossary, actors table, roles tree, use case table, persistent information elements table, secure use case model and interface prototypes), but the most important is that of the *secure use case model*, which is an extension of the use case model, and which allows us to indicate special security characteristics of actors and use cases through two stereotypes (*secure use case* and *authorized actor*). A secure use case is a use case that has confidentiality requirements. Secure use cases are those in which *sensitive* information is read or written. Information that is protected by personal data protection laws, and information that is important for companies (business, strategy, economy, etc.), and which they wish to be kept secret, or with certain security measures, can be considered *sensitive*. An authorized actor is an actor who must have special authorization in order to execute a particular use case. Fig. 5 shows a very straightforward example of the secure use case model.

Most of the activities of the *Requirements Gathering* stage have been inherited from the UP, and adapted to the database development context, so these activities are not described here. The activities in this stage are: gathering initial requirements, creating the business model and the system glossary, searching for actors, searching for use cases, searching for persistent elements, describing use cases, analyzing security in actors and in use cases, defining priorities in use cases, structuring the use case model, searching for relationships between use cases, and reviewing use cases.

The most relevant (and new) activity from the point of view of security is the *analysis of security in actors and use cases*. Once use cases and actors have been identified, this activity is in charge of analyzing all use cases, determining which of them have confidentiality requirements and which actors will need special authorization to execute secure use cases. We then include the necessary stereotypes in the secure use case model.

It is important to mention that at this stage, confidentiality requirements (like other requirements) are not very specific, and that we only identify use cases with *confidentiality requirements* (specifying the corresponding stereotype in the use case diagram), leaving the task of representing the requirement in the conceptual model of the database until the next stage.

The resulting artifacts of this stage (mainly the secure use case model) are crucial for the success of the complete methodology since, as with the UP, this methodology is guided by use cases.

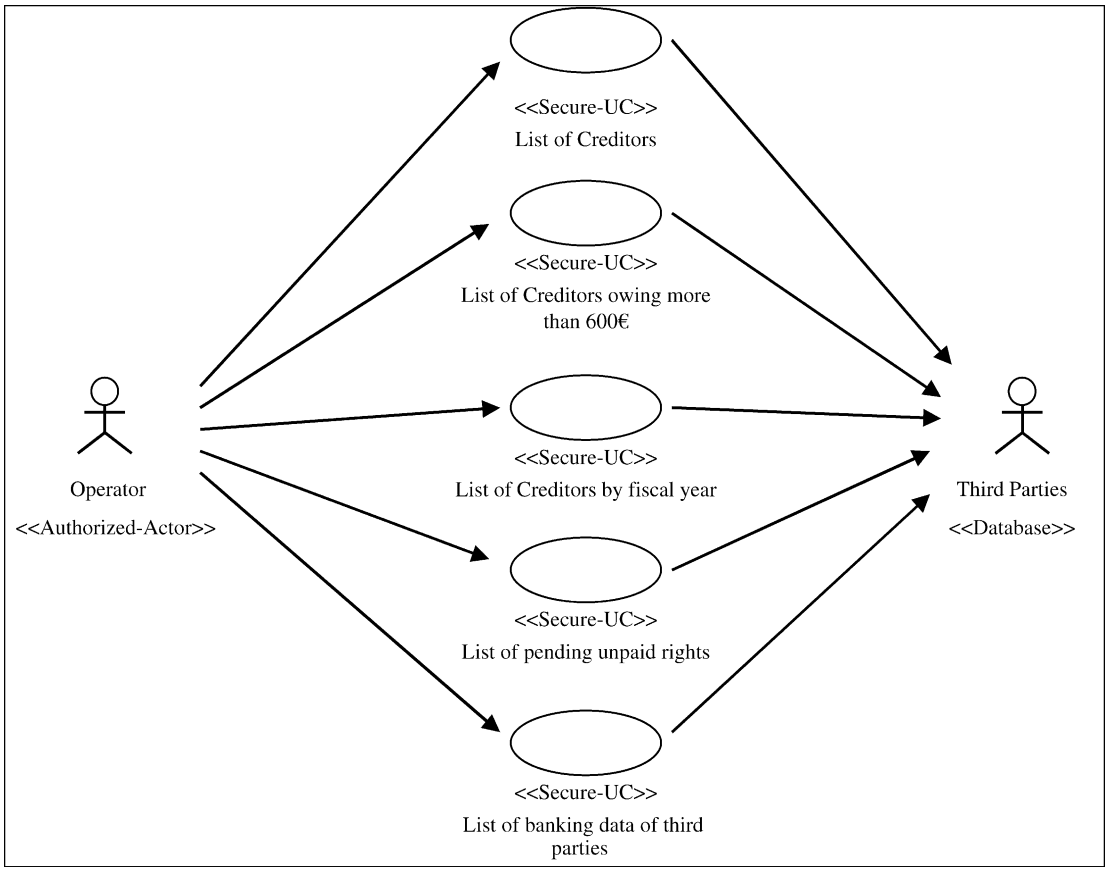

Fig. 5. Fragment of the third parties secure use case diagram.

5.2. Database analysis

The aim of this stage is to build the database conceptual model, considering all the requirements that have been elicited and modeled in the previous activities. The conceptual model will be composed of several artifacts, but the most representative from the point of view of security is the *secure class model*, and a set of *security constraints*, which will represent the origin of the resulting multilevel database.

The secure class model makes it possible to specify security information in classes, attributes and associations, indicating the conditions that the users will have to fulfill in order to access them, with regard to *security levels* and *roles of authorized users* (*user roles* in our methodology are equivalent to hierarchical groups in OLS9*i*). Traditional security levels that have been considered in the context of multilevel databases are *Unclassified*, *Confidential*, *Secret* and *Top Secret*, but our methodology allows the analyst to define the number and name of security levels, according to the confidentiality properties of the database. On the other hand, user roles represent an organizational classification of users, where each role represents an activity or responsibility within the company. These user roles will be represented as *accredited-actors* in the secure use case model.

In this methodology, we deal with secure or multilevel databases, which are based on mandatory policies. Moreover, in addition to security levels, user roles are also integrated as confidentiality information, so MAC and role based access control (in their basic models) are supported in this methodology. Nevertheless, in spite of the fact that discretionary access control (DAC) is one of the most traditional and relevant access control techniques, it has not been considered in the methodology, mainly due to its important security vulnerabilities: DAC can be bypassed by Trojan Horses embedded in programs, and it does not enforce any control on the flow of information once this information is acquired by a process [34].

The language that we use to specify security constraints is the Object Security Constraint Language (OSCL) [32], which is an extension of the OCL [38]. The OSCL language allows the specification of the security constraints that define the information about the security of classes, attributes or associations, depending on a particular condition. For instance, Fig. 6 specifies a security constraint with the following semantics: the security level of the objects belonging to the class *CreditorOfTheExpenseBudget* will be *Unclassified* if *refunds* are less than or equal to 3.000, *Secret* if *refunds* are less than or equal to 10.000 and greater than 3.000, and *Top Secret* in other cases.

```
Context CreditorOfTheExpenseBudgetinv:
    self.SL=if Refunds <= 3000 then U
                    else if Refunds <= 10000 then S
                                        else T
                        endif
        endif
```

Fig. 6. Example of OSCL constraint.

Table 1
Valid security levels

Unclassified	U
Secret	S
Top secret	T

Once again, almost all the activities in this stage (architectural analysis, use case analysis, class analysis and package analysis) are *inherited* from the UP and are not described in the article. The new activity to be included in this stage is *Security Analysis*, which is composed of the following tasks:

(1) Specification of the valid security levels for the database, according to their properties of confidentiality. The left-hand column of Table 1 represents the valid security levels that we have specified for this case study, and in the right-hand column, the abbreviations used in secure class models are specified.
(2) Definition of the user roles hierarchy, considering the type of users that will access the database. Part of the roles hierarchy that has been defined for the case study database is shown in Fig. 7. For each role, a short name, which is used in the diagrams, is also defined.
(3) Assignation of security levels to classes, attributes and associations, taking into account the security properties of the information, and the *inherent constraints* (see Section 5.5) of the secure class model. The secure class model has different inherent constraints that must be fulfilled, such as 'the security level of the attributes should be equal to or more restricted than the security level of the class they belong to'. Fig. 8 illustrates an example of a secure class model, and we can observe that in the diagram, security levels are represented as tagged values, but that in practice, only classes are classified (attributes and associations are not). We are able to assign an individual security level to an element, which means that all instances of that element will be classified exactly in this security level. For example, all instances of the class *CconfidentialInformation* will be *Top Secret*. We can also assign two security levels to an element: maximum and minimum. For example, all instances of the class *BankingData* will be classified between *Unclassified* and *Secret*, and this classification will depend on a security constraint.
(4) Classification of classes, attributes and associations into different authorized user roles. Fig. 8 shows how user roles are represented by means of tagged values, and that again, in practice, only classes are classified. For instance, the class *EconomicalData* has been defined as security level *T* and role *OAC*. This means that the information of the instances that belong to that class will only be accessible to users who have the security level *Top Secret* and who play the role *OAL* (*Account Area Operator*). If there is no security information associated with a class, then it is accessible to all users.
(5) Specification of security constraints, which define the security information of different model elements, using OSCL. In Fig. 8, we can also observe three OSCL constraints, which define the security level of the objects, depending on the value of their different attributes. For instance, the security constraint associated with the class *BankingData* indicates that the security level of its objects will be *Secret* if the value of the attribute *BankDescription* is equal to *Non-Spanish Bank*, and *Unclassified* in any other case.
(6) Analysis of other kinds of security constraints. This methodology is focused on confidentiality problems, but this activity has been defined in order to take into account other security problems in analysis time. In this activity we should try to identify additional security

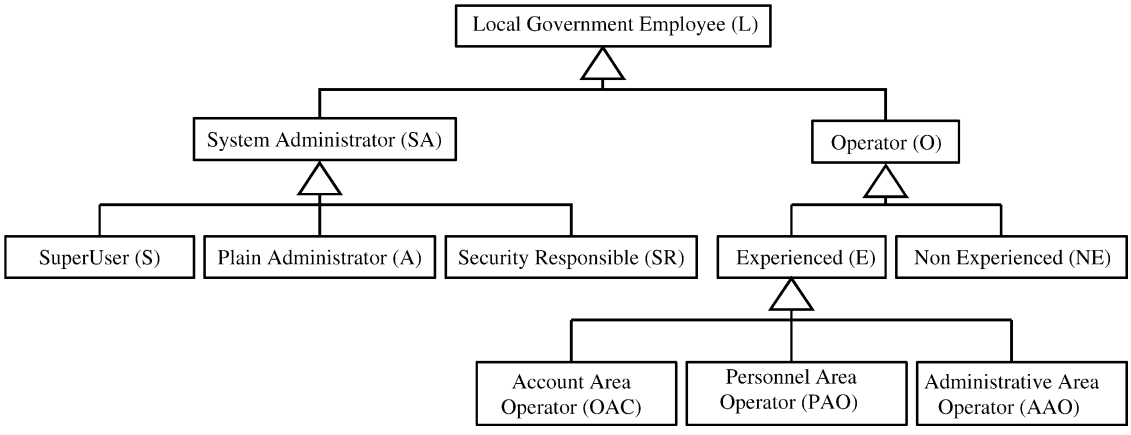

Fig. 7. Roles hierarchy.

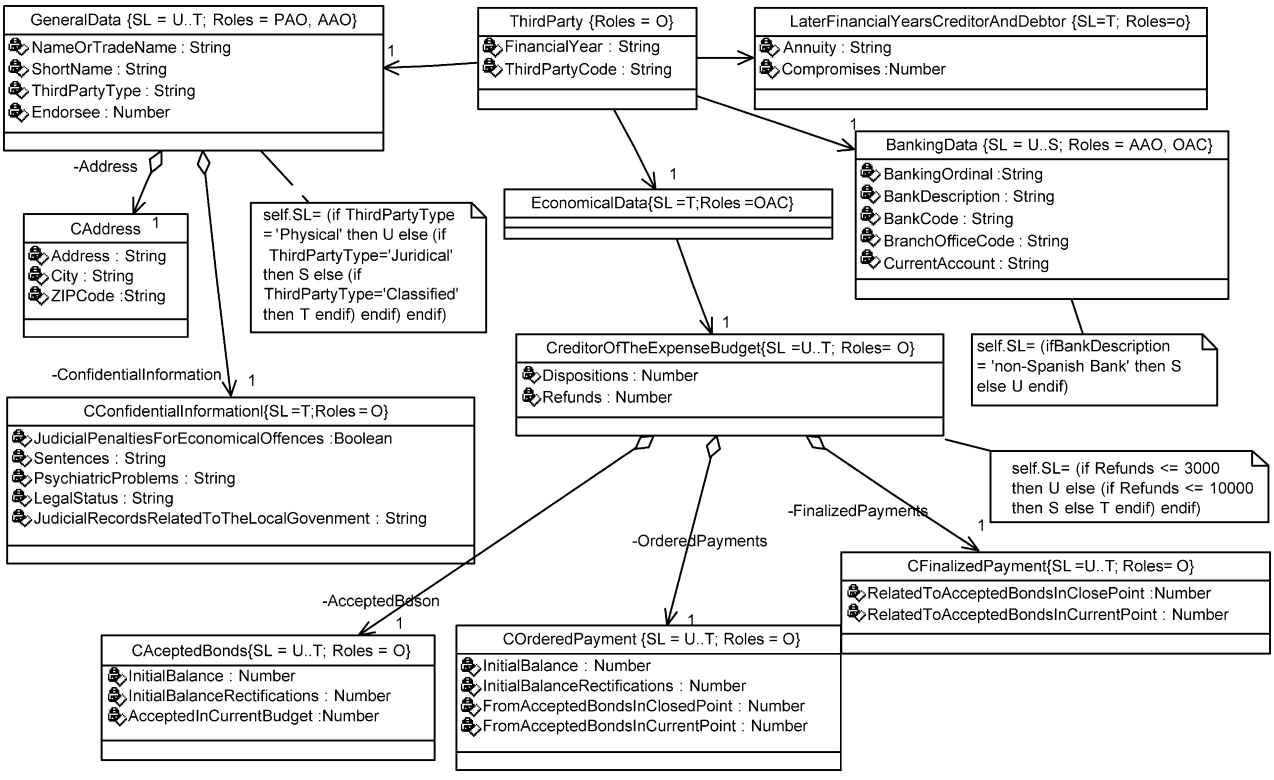

Fig. 8. Third parties class diagram.

requirements, which can be related to confidentiality, integrity or availability.

(7) Definition of the user authorization information, which is composed of the security level and the roles that users play. This activity allows us to identify the users of the database (that will be instances of actors) and to define their authorization information.

(8) Security revision. The aim of this activity is to check the coherence between all the aspects of security that have been defined, both for class model elements and for users.

5.3. Multilevel relational logical design

This stage is the bridge between the database conceptual model and the database specific logical design model. We believe that it is very important to maintain the independence between a *general logical model* and the different specific logical models, because multilevel and other authorization issues vary considerably from one product to another. Up until now we have only considered a general relational security database model because relational databases are the most widely used at present [28], but it could easily be possible to adapt the methodology to develop secure object-relational or object oriented databases.

As we can see, in this methodology there is a 'gap' between this stage and the previous one. In fact, in the conceptual model we use an object oriented paradigm, and in the logical model, a relational paradigm. This problem is solved by defining transformation rules that adapt concepts from the conceptual to the logical model, as in other methodologies such as [6,31].

If we extended the methodology to the object oriented database paradigm, we would not have this gap between the two different paradigms and the transformation of the conceptual to the logical model would be more straightforward.

The three components of the general multilevel relational model are:

Database relational model. This component includes the definition of each relation of the database, considering the attributes necessary for representing the confidentiality information. Using the structure shown in Table 2 (a), we define this component as a set of n relations, where REL_i is the name of the relation i, RL_i and RR_i are attributes that will, respectively, contain the security level and role of each row of relation i, A_{ij} is the name of the attribute j,

Table 2
Multilevel relational model

(a)	$REL_1(RL_1, RR_1, A_{11}, L_{11}, R_{11},\ldots A_{1m}, L_{1m}, R_{1m})$
	…
	$REL_n(RL_n, RR_n, A_{n1}, L_{n1}, R_{n1},\ldots A_{nm}, L_{nm}, R_{nm})$
(b)	$\langle REL_1, IL_1, SR_1, A_{11}, DT_{11}, VL_{11}, VR_{11},\ldots A_{1m}, DT_{1m}, VL_{1m}, VR_{1m}\rangle$
	…
	$\langle REL_n, IL_n, SR_n, A_{n1}, DT_{n1}, VL_{n1}, VR_{n1}, \ldots A_{nm}, DT_{nm}, VL_{nm}, VR_{nm}\rangle$
(c)	Context REL_k inv: SecurityAttribute = OSCLExpression

and L_{ij} and R_{ij} are attributes that will, respectively, contain the security level and role of the attribute A_{ij}.

Metainformation of the model. Each relation has a metainformation tuple associated with it, which includes the datatype of the attributes, and the valid values of the attributes related to security information of the tuples and attributes. We define this component as a set of tuples, as is shown in Table 2 (b), where REL_i is the name of the relation i, IL_i and SR_i are the interval of security levels and the set of user roles that has been respectively defined for this relation i, A_{ij} is the name of the attribute j, DT_{ij} is the datatype of A_{ij}, and VL_{ij} and VR_{ij} are the interval of security levels and the set of user roles that has been respectively defined for A_{ij}.

Security constraints. All the security constraints defined in the conceptual model are specified in this model without loss or modification of their semantics. Each security constraint has the format shown in Table 2 (c), where REL_k is the name of the associated relation, *SecurityAttribute* is the name or the attribute that the security constraint defines (it can be RL_k, RR_k, RL_k, L_{ki} or R_{ki}), and the *OSCLExpression* is the condition that determines the value of the *SecurityAttribute*.

Although security constraints are also metainformation, we have separated them from the rest of the metainformation and consider a separate element in the model where all security constraints are grouped.

The activities in this stage deal with the transformation of all the elements from the secure class diagram into the multilevel relational model. This transformation is similar to the common transformation between conceptual and logical models (see, for example [1,6,31]), differing only in the security information. Security levels and user roles, which have been included in the conceptual model as UML tagged values, and OSCL security constraints, should be appropriately transformed into concepts in the multilevel relational model. We can summarize this transformation as follows:

- Each class, attribute, association, and generalization hierarchy must be transformed into relations and attributes of these relations according to traditional transformation rules [1,6,17,31]. The relational database model will be partially completed with the name of relations (REL_i) and attributes that have been derived from the class diagram (A_{ij}). Also, the metainformation of the model is partially completed with the name of the relations (REL_i) and attributes (A_{ij}), and the datatypes described in the class diagram (DT_{ij}).
- For each tagged value of a class, containing the security level of the class, we must include some information in the multilevel relational model:
  - If the value is an interval this means that the security level of a row may be different to the security level of another row. Thus, in the database relational model we have to define a new attribute for the corresponding relation in order for it to contain the security level of each row (RL_i). Otherwise, the security level would be the same for all rows, and it would therefore not be necessary to define an attribute.
  - In association with the corresponding relation, the value of the security level (IL_i) has to be stored in the metainformation of the model.
- We must include information in the multilevel relational model, considering the two previous rules (defined as L_{ij} and VL_{ij}) for each tagged value of an attribute, containing the security level of the class.
- In this model, we must include the following information for each tagged value of a class which contains the security roles of the class:
  - If the class has an associated security constraint, which makes the set of authorized roles for each row vary, a new attribute must be included in the database relational model (RR_i). If not, the security roles will be the same for all rows, and it is not necessary to define an attribute for them.
  - In association with the corresponding relation, the value of the user roles (SR_i) has to be stored in the metainformation of the model.
- For each tagged value of an attribute containing the security roles of the class, we have to include information in the multilevel relational model, considering the two previous rules (defined as R_{ij} and VR_{ij}).
- Each OSCL constraint has to be adapted to this model. The name of the relation (REL_i) is derived from the name of a class or an association of the class diagram, and the *SecurityAttribute* has been previously defined in the database relational model, and is derived from a tagged value of the class diagram. The *OCLExpression* has to be adapted to the new relation and attributes.

A fragment of the multilevel relational model in our example can be seen in this subsection, but for reasons of space, we have only selected three classes of the secure class diagram to illustrate the results of this stage: *BankingData*, *LaterFinantialYearsCreditorAndDebtor*, and *CreditorOfTheExpenseBudget*.

Fig. 9 shows the database relational model, which contains the name of relations and their attributes, and also the specification of primary (underlined attributes) and foreign (shown with arrows to other relations) keys. There is

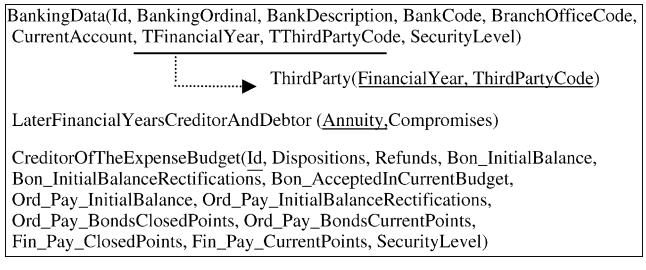

Fig. 9. Database relational model (multilevel relational model).

a direct translation of classes, attributes and associations into relations and attributes, but there is no such direct translation of security information. Security information through tagged values, which are associated with classes, attributes or associations, will be translated into attributes in the database relational model only if this security information is variable. For instance, in Fig. 8 class *BankingData* has the tagged value *SL*=*U*…*S*, which means that not all its instances will be classified at the same security level. In this case we must include a new attribute of the relation *BankingData* which stores the security level of each tuple. On the other hand, class *LaterFinantialYearsCreditorAndDebtor* has the tagged value *SL*=*T*, which means that all its instances will be classified at the level *Top Secret*, so therefore it is not necessary to consider a new attribute that stores the same information for each tuple. A similar mechanism is considered for translating the security information of attributes (which can be translated as additional security attributes associated with the derived attributes of the relational model), and associations (which can be translated as additional security attributes associated with the foreign keys derived from these associations in the relational model).

Fig. 10 contains the metainformation associated with these three classes. This element of the multilevel relational model includes data types of the attributes, and also the security information of each relation and attribute. This is important information which cannot be specified in the database relational model. For instance, in Fig. 9, we considered the additional attribute *SecurityLevel* in relation *BankingData* to store the security level of each tuple of that relation, but we must also register the constraint that only security levels between *Unclassified* and *Secret* will be valid for each of these tuples. In Fig. 10 we can see the security levels *U*…*S* and the user roles *AAO* and *OAC* of the tuple corresponding to the *BankingData* relation. This information comes from the security information that has been specified for the class *BankingData* in Fig. 8. In spite of the fact that security information that appears in Fig. 10 is only related to relations, it can also be specified for attributes.

Fig. 11 shows the security constraints of the multilevel relational model. We can see that the security constraints are very similar to those specified in the secure class diagram. The difference is that here, we work with relations and attributes, and not with classes, associations and attributes. Usually these constraints will be the same as those that have been specified in the secure class diagram, but it is possible that there may be differences due to the translation between conceptual and logical concepts.

```
<BankingData, U..S, AAO, OAC, Id: VarChar; BankingOrdinal: VarChar; Bank-
Description: VarChar; BankCode : VarChar; BranchOfficeCode:VarChar; Curren-
tAccount: VarChar; TfinancialYear: VarChar; TThirdPartyCode : VarChar; Securi-
tyLevel : SL>

<LaterFinancialYearsCreditorAndDebtor,T, O, Annuity: VarChar; Compromises:
Real>

<CreditorOfTheExpenseBudget,U..T, O, Id :VarChar; Dispositions: Real; Refunds
:Real; Bon_InitialBalance: Real; Bon_InitialBalanceRectifications: Real;
Bon_AcceptedInCurrentBudget : Real; Ord_Pay_InitialBalance :Real;
Ord_Pay_InitialBalanceRectifications: Real; Ord_Pay_BondsClosedPoints : Real;
Ord_Pay_BondsCurrentPoints Real; Fin_Pay_ClosedPoints : Real;
Fin_Pay_CurrentPoints : Real; SecurityLevel : SL>
```

Fig. 10. Metainformation (multilevel relational model).

```
Context BankingData inv:
        Self.SecurityLevel = (if self.BankDescription = "Non-Spanish bank"
                              then S
                              else U
                              endif)
Context CreditorOfTheExpenseBudget inv:
        Self.SecurityLevel = (if Refunds <= 3000
                              then U
                              else (if refunds <= 10000
                                    then S
                                    else T
                                    endif)
                              endif)
```

Fig. 11. Security constraints (multilevel relational model).

Traditional multilevel models classify information into different levels of security. Typically, in these models it is possible to classify a tuple into any security level, and this will depend on the security level of the user who creates it. Moreover, polyinstantiation is commonly used to avoid disclosure of sensitive information [34,35].

Polyinstantiation combines the primary key of each relation with security level attributes in order to create a new primary key, allowing the existence of several rows in which the primary key differs only in the security level. Thus, if a user creates a row that already exists but with a higher security level (so that this row is not accessible to this user), the system does not show an error message, as this would be a disclosure of information, but creates the row. The problem polyinstantiation tries to solve is provoked by the property of traditional multilevel models in which the security level of a new row is inherited from the security level of the user who creates the row. However, in our model the level of the information does not depend on the level of the user who creates it, but on the security properties of the information itself, so therefore, polyinstantiation is less important. At analysis-time we decide on the classification level of the information, so that rows receive the previously defined security information when they are created.

5.4. Specific logical design

In this stage we specify the secure database in a particular logical model: OLS*9i*. We have chosen this model because it is part of one of the most important DBMSs, which allows the implementation of label-based databases. Nevertheless, the match between the multilevel relational logical model and OLS*9i* is not perfect, but this is the price of translating a general model (PIM, Platform Independent Model, in Model Driven Architecture [27] terminology) into a more specific one (PSM, Platform Specific Model). For instance, our

general model considers security at attribute level, and OLS*9i* only supports it at row level (a coarser granularity access). On the other hand, OLS*9i* permits the classification of information in security levels, compartment and user groups, and this methodology only considers security levels and user groups.

According to the particularities of OLS*9i*, the activities that transform the multilevel relational model into this specific relational model are as follows:

(1) Definition of the database model. All relations that have been defined in the database relational model have to be created in OLS*9i*. Considering the definition of the multilevel relational model (Table 2), for each REL_i that has been identified in the database relational model, we have to define a CREATE TABLE sentence, specifying the attributes of each relation (A_{ij}) with the corresponding datatype specified in the metainformation of the model (DT_{ij}). It is important to mention that in this case, all security attributes that could be defined in this model for each relation (RL_i and RR_i), have to be discarded here because they are not necessary, since information security in OLS*9i* is stored in an automatically defined label. Moreover, all security attributes that are defined for each attribute (L_i and R_i) also have to be discarded because OLS*9i* does not support security for attributes (only for rows). This is a limitation of OLS*9i* that has a complex solution, so if it is important to also have security for attributes, another secure DBMS should be chosen.

(2) Definition of the security policy and its default options. Different security policies can be defined, but we consider that an OLS*9i* database created using this methodology should have the options that are shown in Fig. 12. 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.

(3) Specification of the valid security information in the security policy. Fig. 13 shows the definition of the security levels that will be valid for this database (which was initially defined in Table 1), and Fig. 14 specifies the user roles tree that we defined for this database (which was initially defined in Fig. 7). This information has been defined in activities 1 and 2 of the database analysis.

(4) Creation of the authorized users and assigning their authorization information. In Fig. 15 the user *User1* is defined with the following information: Max security level *T*, default security level *S*, minimum security level *S*, read access groups *Operator*, write access groups *Operator*, and default groups *Operator*. Since it is not usual to assign special privileges to users, we show how this is possible. This information has been defined in activity 7 of the database analysis.

(5) Definition of the security information for tables through labeling functions. The manner of assigning security information (which is specified in the metainformation of the multilevel relational logical model) to the row, once it has been inserted, is through labeling functions. For each relation REL_i in the multilevel relational 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 (previously defined in the metainformation of the model by IL_i and SR_i). Fig. 16 shows a labeling function definition and the command by which it is assigned to database tables, which are associated with the relations that have been defined in Fig. 8. If we observe the metainformation of the table *LaterFinancialYearsCreditorAndDebtor* in Fig. 10, the security level that has been specified is *Top Secret* and the user group is *Operator*. So *Function1* always creates the same security label (*T::O*), which will be assigned to each instance of that table. We can see that these functions are easily reusable for several tables with the same security classification. Once again, if there is

```
CREATE_POLICY('SALAPolicy','SecurityLabel', 'HIDE,
                CHECK_CONTROL, READ_CONTROL,
                WRITE_CONTROL')
```

Fig. 12. Security policy definition.

```
CREATE_LEVEL('SALAPolicy', 1000, 'U','Unclassified')
CREATE_LEVEL('SALAPolicy', 2000,'S','Secret')
CREATE_LEVEL('SALAPolicy', 3000, 'T', 'Top Secret')
```

Fig. 13. Security levels definition.

```
CREATE_GROUP('SALAPolicy', 1, 'L', 'LocalGovermmentEmployee')
CREATE_GROUP('SALAPolicy', 2, 'O','Operator', 'L')
CREATE_GROUP('SALAPolicy', 3, 'SY', 'SystemAdministrator','L')
CREATE_GROUP('SALAPolicy', 4, 'S','SuperUser','SA')
CREATE_GROUP('SALAPolicy', 5, 'A','NormalAdministrator', 'SA')
CREATE_GROUP('SALAPolicy', 6, 'SR', 'SecurityResponsible', 'SA')
CREATE_GROUP('SALAPolicy', 7, 'E', 'Experienced','O')
CREATE_GROUP('SALAPolicy', 8, 'NE', 'Non-Experienced', 'O')
CREATE_GROUP('SALAPolicy', 9, 'AAO', 'AccountAreaOperator', 'E')
CREATE_GROUP('SALAPolicy', 10, 'PAO', 'PersonnelAreaOperator', 'E')
CREATE_GROUP('SALAPolicy', 11, 'OAC', 'AdministrativeAreaOperator', 'E')
```

Fig. 14. User roles definition.

```
SET_LEVELS('SALAPolicy', 'User1', 'T', 'S', 'S')
SET_GROUPS('SALAPolicy','User1', 'O', 'O','O')
SET_USER_PRIVS('SALAPolicy', 'User1', 'FULL, WRITEUP,
                    WRITEDOWN, WRITEACROSS')
```

Fig. 15. User definition.

```
CREATE FUNCTION Function1( ) ReturnLBACSYS.LBAC_LABEL
As   MyLabel varchar2(80);
Begin
  MyLabel := 'T::O' ;
  Return TO_LBAC_DATA_LABEL('SALAPolicy', MyLabel);
End;
APPLY_TABLE_POLICY ('SALAPolicy', 'LaterFinancialYearsCreditorAndDebtor',
                    'Scheme', , 'Function1')
```

Fig. 16. Labeling functions definition. Constant security information.

security information for attributes (VL_{ij} or VR_{ij}), this information has to be discarded because OLS*9i* does not support security for attributes.

(6) Implementation of the security constraints through labeling functions. For each security constraint that has been specified in the multilevel relational model, a labeling function that implements the constraint has to be defined. Fig. 17 shows the implementation of the security constraints that have been specified in Fig. 11. *Function2* creates the security information, depending on the value of the column *BankDescription*, and *Function3* depending on the value of the column *Refunds*. These functions are then, respectively, associated with the tables *BankingData* and *CreditorOfTheExpenseBudget*.

(7) Implementation, if necessary, of the operations and control of their security. In this case study, this has not been necessary.

5.5. Considerations of verification and validation

Verification and validation (V & V), including testing, are important activities in all software development methodologies, so in this subsection we introduce some considerations regarding V & V [5,17,23].

The V & V activities that have been considered in the methodology presented in this article are classified in three non-disjoint groups: V & V activities of UP [25], V & V that are commonly applied to databases [17], and V & V activities that are necessary in multilevel security [35].

UP V & V activities: like UP, our methodology is iterative and incremental. This allows us a better risk control, and particularly permits us to reduce the risk of building a database that does not fulfill user requirements. In fact, this philosophy helps in building the database step by step, and in defining many checkpoints that allow us to prove the coherence between the stages of the methodology, and the development of the right product. Additionally, as in other methodologies, we have defined some activities of inspection and review in each stage: reviewing use cases in the requirements gathering stage; reviewing classes, reviewing attributes and associations, and reviewing security in the database analysis stage, and reviewing the model in the multilevel relational logical design stage. On the other hand, we have adapted the UP test stage to be included in this methodology. This stage is more important in the last iterations of the methodology, when part of the product is built. In our methodology we have considered both the UP artifacts (test case, test procedure, test component, test model, test plan, defect, and test evaluation) and workflows (plan and design test, implement test, and execute and evaluate test).

```
CREATE FUNCTION Function2 (BankDescription: VarChar)
                                    Return LBACSYS.LBAC_LABEL
As MyLabel varchar2(80);
Begin
  If BankDescription='Non-Spanish bank' then MyLabel := 'S::AAO,OAC';
                                        else MyLabel := 'U::OAA,OAC';
  end if;
  Return TO_LBAC_DATA_LABEL('SALAPolicy', MyLabel);
End;
CREATE FUNCTION Function3 (Refunds: Real) Return LBACSYS.LBAC_LABEL
As MyLabel varchar2(80);
Begin
  If Refunds <=3000 the MyLabel := 'U::O';
    else if Refunds <= 10000 then MyLabel := 'S::O';  elseMyLabel := 'T::O'; end if;
  end if;
  Return TO_LBAC_DATA_LABEL('SALAPolicy', MyLabel);
End;

APPLY_TABLE_POLICY ('SALAPolicy','BankingData', 'Scheme', , 'Function2')
APPLY_TABLE_POLICY ('SALAPolicy','CreditorOfTheExpenseBudget',
                                    'Scheme', ,'Function3')
```

Fig. 17. Labeling function definition. Security constraints.

Database V & V activities: V & V activities are quite usual in the context of software engineering, but they are not so common in the database development. These activities are usually related to database integrity. Therefore, in the test stage, we have included activities in order to check database integrity constraints, such as with regard to domains, keys, dependencies, etc. Moreover, typical database tests [17], such as data charge, and database performance test have also been included.

Security V & V activities. These activities are focused on the fulfilment of the security requirements that have been elicited and modelled in requirements gathering and database analysis stages, respectively, and on the correct integration of the multilevel specifications and constraints into the class model. All inspection and review activities commented on before (in UP V & V) include tasks to verify all security decisions. Moreover, the UP test approach allows us to plan, design, implement, execute and evaluate tests, also with regard to confidentiality requirements. On the other hand, all the security information that the extended class model may contain, must respect some inherent model constraints (formally specified with OSCL in [18]):

- The security levels defined for each class of the model, for each attribute, and for each association, must belong to the sequence of security levels that has been defined for the model.
- The set of user roles defined for each class, attribute, and association of the model has to be a subtree of the roles tree that has been defined for the model.
- The security level of the instance of a class must be included in the interval of security levels that have been defined for this class. The same rule is applicable to the instances of attributes and links.
- The user roles of an instance of a class must be subtrees of the roles trees that have been defined for the class. The same rule is applicable to the instance of attributes and links.
- The security levels defined for an attribute have to be equal to or more restricted than the security levels defined for its class. The same rule is applicable to the role hierarchies.
- The security levels defined for an association between two classes have to be equal to or more restricted than the security level of these classes. The same rule is applicable to the role hierarchies.
- In generalization hierarchies, the security level of the subclasses has to be equal to or more restrictive than the security level of the superclass. This rule is applicable to user roles.

The manual verification of these constraints is tedious, and not very reliable. Therefore, the CASE tool that we introduce in the next section automatically verifies the fulfillment of all the class model security inherent constraints.

6. Case tool

A CASE tool that extends Rational Rose has been developed in order to automate the *requirements gathering* and *database analysis* stages of the secure database design methodology. The main functions of this tool can be grouped as follows:

- *System Security Information Definition*. For each database, it is possible to define and manage the valid values of security levels and the valid user role hierarchy.
- *Use Cases Security Information Definition*. This functionality helps to model use case diagrams, integrating the new stereotypes that have been defined for the extended use case diagrams (*secure use case* and *authorized actor*).
- *Class Diagrams Security Information Definition*. This functionality allows us to model class diagrams, but this is done by integrating the new tagged values that contain the security levels and user roles which can be defined for each element of the class model (classes, attributes and associations). The CASE tool does not only allow us to introduce security information in the models, but it automatically checks the inherent constraints of the extended class diagram.
- *Security Constraint Specification*. OSCL security constraints can be introduced in the class diagram by using this functionality. Once the security constraint has been introduced, an automatic lexical and syntactical analysis is performed.

This CASE tool has been implemented as a dynamic link library, in Visual Basic 6. We have used the Rose Extensibility Interface, which allows us to program using a direct interface of the UML diagrams, and modify and extend the basic UML stereotypes. Therefore, it has been possible to extend the use case and class diagrams, and to implement a CASE tool for managing these diagrams and the security information. Finally, the tool has been added to Rational Rose (in both its 98 version and that of 2000), using its add-in manager, by means of a simple installation.

This CASE tool is being extended in order to support the complete methodology. Therefore, in addition to the functionality previously commented, the new CASE tool will semi-automatically generate the multilevel relational logical model starting from the conceptual model. As a concrete application, the multilevel relational logical model will be processed to semi-automatically generate the OLS*9i* code which defines the database structure, implements the necessary labeling functions, and defines the security policy according to the information gathered throughout the application of the methodology.

7. Lessons learned

The development of this methodology, together with its application for the design of a secure database for a Provincial Government in Spain during a period of almost 2 years, and with continuous revisions and feedback has contributed to many advantages and lessons learned. This activity has been positive both for the Provincial Government and for the results of our research.

The positive aspects for the Provincial Government have been the following

- Considering that Society's concerns for security aspects in information technology is not great, the application of this methodology, together with the extension of personal data protection laws, have contributed to raising the Provincial Government staff's awareness of the importance of confidentiality and privacy.
- Security flaws have been detected and corrected from the first stages of the application of the methodology. Therefore, the application of a methodological approach in the development of this secure database (less expenses and development effort) has reaped considerable benefits.
- The personal data protection laws are fulfilled, and therefore, the privacy of citizens and the confidentiality of companies can be respected.
- The staff of the Provincial Government that collaborated with us has been motivated, and has participated in the design of the secure database, and in the improvement of the methodology.

From the point of view of the research, this collaboration has helped us in several aspects:

- The application of the action research method has allowed us to define the models, languages and stages of the methodology, and to improve them thanks to continuous feedback and tests.
- It has been possible to define the requirements of the CASE tool we have developed, and to test and use it with this case study.
- We have realized the high complexity of applying this methodology together with a DBMS that does not support multilevel security (e.g. Ingres).
- We have also realized that V & V activities in the methodology have to be improved. This fact confirms the result of [20] in which authors mention that V & V activities that are considered in the standard IEEE 1012 [23] are not completely supported by the UP.
- We have realized that many security requirements are repeated. Therefore an approach in which requirements can be reused could be integrated within the methodology.
- Some details regarding the granularity of information classification have been discovered. The methodology has to be general enough, allowing a high degree of granularity in the classification, but this case study demonstrates that: It is sometimes necessary to specify security levels for attributes that are different from the security level of the class they belong to; it is infrequently necessary to specify security roles for attributes that are different from the roles of the class they belong to; it is infrequently necessary to specify security constraints (both for levels and roles) for attributes; it is frequently necessary to define security levels and roles for classes; and sometimes it is necessary to specify security constraints, but always with regard to security levels.

Moreover, this collaboration has generated some interesting new ideas and future considerations, such as the extension of the methodology to develop other database paradigms, for example object-relational or XML databases, which are being gradually introduced in companies and public institutions. Another important challenge is to extend the methodology to be applied with DBMS that does not support multilevel security. In such a case, access control and security constraints have to be enforced by implementing complex triggers, and security information must be stored in hidden columns.

8. Conclusions and future work

The critical nature of IS and especially of databases for modern business, together with new requirements of laws and governments, make more sophisticated approaches necessary to ensure database security.

Traditionally, information security deals with different research topics, such as access control techniques, secure architectures, cryptographic methods, etc. Although all these topics are very important, we believe it is fundamental to use a methodological approach, where security is taken into consideration at all stages of the database development process and especially from the earliest stages of the life cycle. In this article we have summarized a methodology for designing secure databases, which extends the most widely accepted modeling languages, process models, constraint languages and security models in the industrial and research community. The methodology has been refined and tested by designing a secure database for a Spanish local government, thus solving its problems of confidentiality. We have also developed a CASE tool to automatically support the management of secure use case and secure class diagrams and OSCL security constraints.

We are extending the methodology presented in this article in several directions: it may be interesting to improve and extend languages and techniques, for example, to consider new kinds of security constraints, such as temporal, integrity or availability constraints [12]. We are also improving the requirements gathering stage in order to

reuse legal requirements [37]. Other interesting future lines are to extend and adapt this methodology to be applied to the design of multimedia information (digital libraries, multimedia databases, XML-based multimedia documents, etc.), and issues related to the World Wide Web (web databases, web portals, intranets, etc.).

Acknowledgements

This research is part of the CALIPO project, supported by the Dirección General de Investigación of the Ministerio de Ciencia y Tecnología (TIC2003-07804-C05-03), and the MESSENGER project, supported by the Consejería de Ciencia y Tecnlogía of the Junta de Comunidades de Castilla-La Mancha (PCC-03-003-1). We would like to thank the rest of the Alarcos Research Group members, and especially the reviewers of this journal for their valuable comments, and Antonio Martínez, director of the SALA project.

References

[1] P. Atzeni, S. Ceri, S. Paraboschi, R. Torlone, Database Systems. Concepts, Languages and Architectures, McGraw-Hill, New York, 1999.
[2] D. Avison, F. Lau, M. Myers, A. Nielsen, Action research, Communications of the ACM 42 (1) (1999) 94–97.
[3] R. Baskerville, Information systems security design methods: implications for information systems development, ACM Computing Surveys 25 (4) (1993) 375–415.
[4] C. Batini, S. Ceri, S. Navathe, Conceptual database design. An entity-relationship approach, Addison-Wesley, New York, 1991.
[5] R.V. Binder, Testing Object-Oriented Systems—Models, Patterns, and Tools, Addison-Wesley, Reading, MA, 2000.
[6] M. Blaha, W. Premerlani, Object-Oriented Modeling and Design for Database Applications, Prentice-Hall, Englewood Cliffs, NJ, 1998.
[7] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language, User Guide, Addison-Wesley, Redwood city, CA, 1999.
[8] D. Brinkley, R. Schell, What Is There to Worry About? An Introduction to the Computer Security Problem in: M. Abrams, S. Jajodia, H. Podell (Eds.),, Information Security, An Integrated Collection of Essays, IEEE Computer Society, Silver Spring, MD, 1995.
[9] S. Castano, M. Fugini, G. Martella, P. Samarati, Database Security, Addison-Wesley, Reading, MA, 1994.
[10] L. Chung, B. Nixon, E. Yu, J. Mylopoulos, Non-functional requirements in software engineering, Kluwer Academic Publishers, Dordrecht, 2000.
[11] T. Connolly, C. Begg, Database systems. A practical approach to design, implementation, and management, Addison Wesley, Reading, MA, 2002.
[12] C. Conrad, K. Turowski, Temporal OCL: Meeting Specification Demands for Business Components in: K. Siau, T. Halpin (Eds.), Unified modeling language: Systems analysis, design and development issues, IDEA Group Publishing, 2001, pp. 151–165.
[13] 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.
[14] G. Dhillon, Information Security Management: Global challenges in the New Millennium, Idea Group Publishing, 2001.
[15] G. Dhillon, J. Backhouse, Information system security management in the new millennium, Communications of the ACM 43 (7) (2000) 125–128.
[16] Directive, 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 L281, 23/11/1995 (95/46/CE) 0031-0050.
[17] R. Elmasri, S. Navathe, Fundamentals of Database Systems, Addison-Wesley, Reading, MA, 2002.
[18] E. Fernández-Medina, M. Piattini, Extending OCL for Secure Database Development (accepted), Proceedings of The Unified Modeling Language Conference, Lisbon (Portugal), 2004.
[19] E. Ferrari, B. Thuraisingham, Secure Database Systems in: M. Piattini, O. Díaz (Eds.), Advanced Databases: Technology Design, Artech House, 2000.
[20] C. Fuhrman, F. Djlive, E. Palza, Software Verification and Validation within (Rational) Unified Process, Proceedings of 28th Annual NASA Goddard Software Engineering Workshop (SEW'03). IEEE, Greenbelt, Maryland, 2003, pp. 216–221.
[21] A. Ghosh, C. Howell, J. Whittaker, Building software securely from the ground up, IEEE Software 19 (1) (2002) 14–17.
[22] A. Hall, R. Chapman, Correctness by construction developing a commercial secure system, IEEE Software 19 (1) (2002) 18–25.
[23] IEEE, IEEE Standard for software verification and validation, (Ed.), IEEE Std 1012-19981998.
[24] ISACF, Information Security Governance. Guidance for Boards of Directors and Executive Management, Information Systems Audit and Control Foundation, USA, 2001.
[25] I. Jacobson, G. Booch, J. Rumbaugh, The unified software development process, Addison-Wesley, Reading, MA, 1999.
[26] 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.
[27] A. Kleppe, J. Warmer, W. Bast, MDA Explained, The Model Driven Architecture: Practice and Promise, Addison-Wesley, Reading, MA, 2003.
[28] N. Leavitt, Whatever happened to Object-Oriented Databases?, Industry Trends. IEEE Computer Society 33 (8) (2000) 16–19.
[29] J. Levinger, Oracle label security, Administrator's guide, Release 2 (9.2), http://www.csis.gvsu.edu/GeneralInfo/Oracle/network.920/a96578.pdf., 2002.
[30] D. Marks, P. Sell, B. Thuraisingham, MOMT: a multi-level object modeling technique for designing secure database applications, Journal of Object-Oriented Programming 9 (4) (1996) 22–29.
[31] R. Muller, Database Design for Smarties, Using UML for Data Modeling, Morgan Kaufmann Publisher, San Francisco, CA, 1999.
[32] M. Piattini, Fernández-Medina, Specification of security constraints in UML, Proceedings of 35th Annual 2001 IEEE International Carnahan Conference on Security Technology, London (UK), 2001 pp. 163–171.
[33] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Object-Oriented Modeling and Design, Prentice-Hall, Englewood Cliffs, NJ, 1991.
[34] 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.
[35] R. Sandhu, F. Chen, The Multilevel Relational Data Model, ACM Transactions on Information and Systems Security 1 (1) (1998).
[36] G.W. Smith, Modeling security-relevant data semantics, IEEE Transactions on Software Engineering 17 (11) (1991) 1195–1203.
[37] A. Toval, A. Olmos, M. Piattini, Legal Requirements Reuse: A Critical Success Factor for Requirements Quality and Personal Data Protection, Proceedings of IEEE Joint International Requirements Engineering Conference (RE'02), IEEE Computer Society, Silver Spring, MD, 2002. pp. 95–103.
[38] J. Warmer, A. Kleppe, The object constraint language, Addison-Wesley, Reading, MA, 1998.