# VIEWS AS SECURITY MECHANISM IN OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEMS

Michael Dobrovnik, Elke Hochmüller

Institut für Informatik-Systeme
Universität Klagenfurt
Klagenfurt, AUSTRIA
Email: {michi,elke}@ifi.uni-klu.ac.at

## ABSTRACT

The semantically rich data models of object-oriented database management systems (OODBMSs) are much more a challenge to achieve security than the traditional (relational) models. Issues to be considered include inheritance relationships and complex object aggregations as well as method executions.

In relational systems, views have their place as one important building block of the security mechanism. They allow to derive and re-structure the schema according to security requirements and also to implement content based restrictions on database access.

The construction of derived schema elements and schema restructuring in order to design subschemas as external models are also main goals of approaches for view mechanisms in OODBMSs. While security is not the main focus of those proposals, it seems equally promising to use them as implementation means to achieve security like in the relational systems.

One of the newer and more powerful proposals for views in OODBMSs is our approach named eXoT/C. It supports the derivation of external schemas from the conceptual schema by means of a mapping specification. All security requirements can be captured in the mapping itself. For the users of external schemas this mechanism is transparent and there are no security induced effects on the conceptual schema which would distort its semantics.

Through an example which takes into account all the important categories of security constraints in OODBMSs we will investigate the suitability and assess the usefulness of eXoT/C with respect to security issues.

## 1 MOTIVATION

Approaches to achieve security in database systems can be classified into two categories, namely discretionary and mandatory access control.

**Discretionary access control.** The main idea of discretionary security mechanisms (also called authorization models), is that of ownership of information in combination with delegation of access rights. An owner of a piece of information can apply any legal operation to it. The owner can further grant a subset of his rights to other users at his complete discretion. Grantees can also be permitted to pass a subset of their rights to other users.

While this approach is very flexible and adaptive, there is no central security policy that can be enforced by a subsystem of the database. The security of the data heavily depends on the discipline and wellbehavedness of the individual holders of access rights. Semantically rich object-oriented data models which support inheritance, associations, composite data structures, navigation and methods pose additional challenges to discretionary access control (Lunt and Fernandez, 1990).

**Mandatory access control.** Mechanisms for mandatory access control require to label all objects in a database based on their sensitivity (*classification*). The active processes requesting access to database objects (e.g. users, application programs) are called *subjects* and are assigned to a *clearance* level. Classifications and clearances are partially ordered. There is a strictly enforced central security policy (the Bell-LaPadula model) which has two aspects. A subject is allowed to read a data item if the subject's clearance is greater than or equal to the data item's classification (*read down*). A subject is allowed to write data classified equal to or higher than the subjects clearance (*write up*). This ensures unidirectional information flows from subjects at a lower clearance to subjects at a higher one and leads to a multilevel database which appears to be different to subjects at different clearances (Pernul, 1995).

The rigidity of this policy implies high inflexibility and has severe impact on schema design. In most approaches, a conceptual class whose attributes are attached to different classifications (multilevel classes) requires a realization via a set of interconnected classes, each at a single security level. Additional inheritance relationships and associations based solely on security requirements are thereby introduced to the schema (cf. Millen and Lunt, 1992). The resulting schemas have poor quality in the areas of minimality, expressiveness, readability, self-explanation and extensibility (cf. Batini et al. 1992).

In addition to the two traditional kinds of access control mechanisms, the ANSI/SPARC three-level architecture could serve as a basis for the realization of a security subsystem. The external schemas can be regarded as the sole part of the database a user or application is permitted to access. This perspective suggests a closer investigation of the applicability and usefulness of external schemas as building block for access control.

The challenge is to provide a mapping mechanism between the conceptual schema and the external schemas which is powerful enough to express all common kinds of security constraints. Semantically rich object oriented data models with their support of static and dynamic constructs seem to be ideally suited as an implementation framework.

**Object-oriented views.** As in relational systems, views in OODBMSs allow one to construct derived schema elements and subschemas. Although most proposals for OO views do not claim to be security models in the first place they provide a schema mapping facility which can be utilized as means to realize security support in the object-oriented context.

The applicability of object-oriented views for the realization of security constraints was already examined by Hochmüller (1996), with eXoT/C (say: exotic) as representative of an object-oriented view approach. In the comparison, the problem of inheritance anomalies did arise especially in the case of mandatory access control of multilevel real world entities realized by single level objects even for minuscule examples, while the object-oriented view approach was free of this semantic dilemma.

This result inspired us to further investigate the principal suitability of object-oriented views for access control. The central idea of this paper is not a complete presentation of the area of object-oriented views (for a comparison of different proposals cf. Motschnig-Pitrik, 1996 and Dobrovnik, 1997), but to discuss the realization of access control with eXoT/C through an example which features all categories of security constraints (cf. Pernul, 1994; Castano, 1995).

The rest of the paper is structured as follows. In section 2 the security constraint categories are presented in the context of the example which will be used throughout the paper. Section 3 gives a brief introduction to the basic features of eXoT/C illustrated by the implementation of the conceptual schema of the example. The realization of the schema mapping together with its discussion by constraint category is dealt with in section 4. An assessment of the object-oriented view approach in contrast to the two traditional security approaches can be found in section 5. The security-specific contributions of our approach are summarized in the last section.

## 2 SECURITY CONSTRAINTS

In order to establish a common basis for the explanation of the various security constraint categories as well as for the discussion of the realization of security constraints with object-oriented views, we will next introduce a simple example schema. Afterwards, the security constraints proposed so far in the literature will be briefly discussed .Then, we will constrain the example schema according to security requirements. The applicability of object-oriented views for the realization of these security constraints will be demonstrated in section 4.

### 2.1 Example schema

For the sake of uniformity, the OMT notation of Rumbaugh et al. (1991) will be used (with some extensions) throughout the paper to represent the static object schemas of the examples. As required by OMT, inherited attributes and methods will not be denoted again.

Fig. 1 contains the OMT object schema of a conceptual model in a simplified enterprise domain. It consists of three object types: `Employee`, `Manager`, and `Project`. Object type `Employee` is characterized by the attributes `ssn`, `name`, `salary`, `birthday`, `sex` and the method `age()`. The object type `Project` has `title`, `budget`, and `category` as attributes. Employees can be assigned to projects (many-to-many relationship). Object type `Manager` is defined as subtype of `Employee`and inherits all attributes, methods and relationships of `Employee`. Each `Project` is lead by one `Manager`. `Bonus` is an additional attribute of `Manager`. The method `total_budget()`returns the sum of budgets of those projects which are lead by the manager.

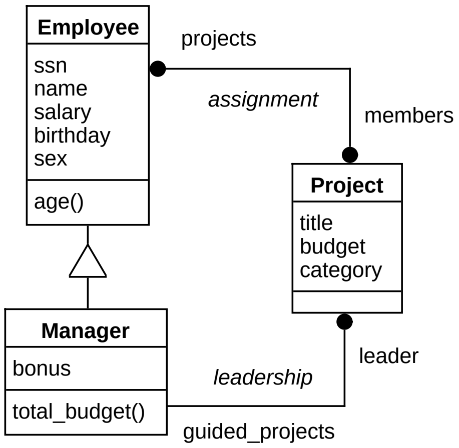

**Fig. 1 Conceptual object schema**

### 2.2 Security constraint categories

In the sequel, we will talk of *security objects* when we refer to objects in the database as well as when we consider query results. Security constraints are already discussed and classified in literature (cf. Castano et al., 1995; Pernul, 1994; Pernul, 1995). In our discussion of security constraints, we follow the taxonomy as presented by Pernul (1994). This classification distinguishes between two different types of application-dependent security constraints:

- constraints that classify characteristic properties (attributes, methods) of security objects (2.2.1 to 2.2.4)
- constraints that classify retrieval results (2.2.5 to 2.2.7)

In the following, all these kinds of security constraints will be shortly described and illustrated by example constraints on the elements of the object schema represented in fig. 1.

#### 2.2.1 Simple constraints

Simple constraints classify certain properties of one security object to be at a higher security level than others of the same object.

**Example.** The budget of each project is confidential. This requires to classify the attribute `budget` of object type `Project` at a secure level.

### 2.2.2 Content constraints

Content constraints classify characteristic properties of one security object based on their particular values or on the values of other properties of the same object.

**Example.**The birthday of women is confidential. Hence, the security level of `Employee's` attribute `birthday` depends on the value of attribute `sex`.

### 2.2.3 Complex constraints

Complex constraints classify characteristic properties of one security object based on the values of properties of another associated object.

**Example.**All information about critical projects is confidential. This implies that `assignments` of `Employees` to `Projects` with `category = `critical`` has to be classified at a secure level.

### 2.2.4 Level-based constraints

Level-based constraints classify characteristic properties of one security object based on the classification of another property of the same object.

**Example.**The method `age()` of `Employee` must always be at the same classification level as the attribute `birthday`.

### 2.2.5 Association-based constraints

Association-based constraints classify the combination of the value of certain characteristic properties of one object with the identifying property of this object at a higher level of classification as the values of the unrelated properties themselves. This constraint type is also called *context constraint* (Castano, 1995).

**Example.** The salary of each individual employee is confidential. However, the values of salary without any information about which employee gets what salary are unclassified. Thus, the relationship between the values of the attribute `salary` and the corresponding `Employee` objects are classified at a secure level.

### 2.2.6 Inference constraints

Inference constraints restrict from being able to infer classified information by using unclassified data and hidden paths which may also involve personal background knowledge.

**Example.** The total budget of a manager can be considered as unclassified if the amount of these projects is high enough. This would require to allow the computation of the total budget (message `total_budget()`) only if a proper condition (e.g. cardinality of property `guided_projects` should be greater than 3) holds.

### 2.2.7 Aggregation constraints

Aggregation constraints classify several instances of the same security object at a higher security level than single instances of the same object. As single instances usually can be aggregated offline, an aggregation constraint might not be very useful on its own. Moreover, such an aggregation constraint will require to restrict access to the single instances (through other constraints of the categories presented above) in order to inhibit offline aggregation.

**Example.** Let us consider the assignment of projects to employees as unclassified. However, the aggregation of all employee assignments to a certain project should be at a secure level. This should protect from inferring which team is responsible for what project (cf. Pernul, 1994). However, enough trials of querying single assignments will help to elude any aggregation constraint.

## 2.3 Example schema with security constraints

Fig. 2 is an extension of fig. 1. It contains some security constraints which will subsequently be discussed within the application context. For reasons of simplicity we use two classification levels: unclassified (U) and secure (S). Only secure elements will be explicitly denoted as such ones in the object schema of fig. 2. We use this simple notation for the sake of a clear presentation. For a somewhat more complete notation the reader is referred to MOMT (Marks et al., 1996).

The attribute `budget` of a `Project` is secure. Projects themselves are either unclassified or secure. A Project is secure if its `category` is `'critical'`. All information about secure projects is secure (assignments, leaderships, and exclusion from computation of manager's budget sum).

The values of attribute `salary` are unclassified. Their association to instances of object type `Employee`, however, is secure. The `birthday` of women is secure. This requires also the computation of women's age to be secure (corresponds to a kind of 'dynamic' level-based constraint).

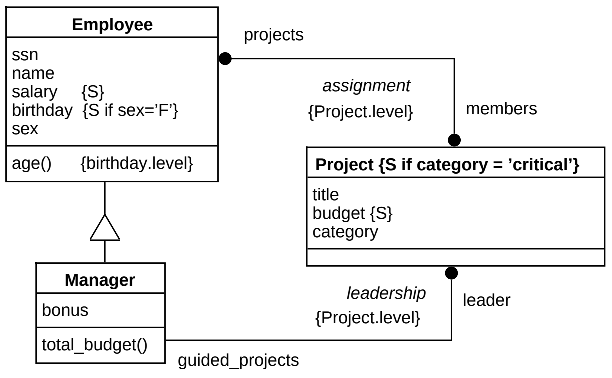

**Fig. 2 Object schema with security constraints**

## 3 OBJECT-ORIENTED VIEWS WITH eXoT/C

### 3.1 General Architecture

The main principle of the eXoT/C proposal as developed by Dobrovnik (1995) is the clear and clean separation of the conceptual schema from the external schemas. An external schema can be regarded as a kind of interface specification through which the database (the conceptual schema) can be accessed. The designer of an external schema implements it as a derived schema via a mapping specification.

There are special type constructors which allow to derive external schema elements (*types*, *containers*) from conceptual ones. Object generating and object preserving semantics are supported. We do not only take into account structural aspects (*type and schema restructuring*) but also deal quite extensively with dynamic aspects as *inter schema method resolution* and *method steadiness* (cf. Dobrovnik and Eder, 1996). Since we are not in the position to present an in-depth treatment of eXoT/C here due to space limitations, we refer to Dobrovnik and Eder, 1994 and to Dobrovnik, 1995.

In the sequel, we will present the data model of eXoT/C which can essentially be regarded as being a subset of the ODMG proposal (Catell et al. 1997). The data model is illustrated by the conceptual schema implementation of the running example from fig. 1. The constructs for the derivation of external schemas will be explained in section 4.

### 3.2 Data Model

We distinguish between extensional and intensional concepts, so a schema in our data model consists of a set of types and a set of containers. The types describe the structural and behavioral aspects of the objects and values.

We provide some atomic value types (boolean, integer, string, ...) and an atomic root object type. The type constructors *set* and *tuple* can be orthogonally applied to types to build set valued and tuple valued structured value types.

Object types (also commonly referred to as classes) can be declared through the use of the object type constructor *object*. They are positioned in an inheritance lattice which supports conventional structural top down multiple inheritance semantics. Conflicts are circumvented by demanding an unambiguous origin of the object type components and methods.

The definition of a subtype can make use of covariant redefinition of attributes and method signatures. The subtype relation also defines type substitutability and assignment compatibility, namely wherever an instance of a certain type can be used, it is also allowed to use an instance of one of its subtypes.

At the extensional level, we provide containers, which can be described as typed object sets. An instance of an object type can be added to any type-compatible container and can also be removed from it. The containers are user defined object sets which also provide persistence. An object persists the current session when it is in at least one object container or when it is referenced by another persistent object (*persistence by reachability*). Currently, there is no hierarchy defined between the containers. The object types are the factories and the containers are the warehouses of the object instances.

We assume the existence of a Turing complete procedural language for the implementation of methods and also of a declarative query language.

All object types have to be *well formed*, a property which restricts overriding of components and methods to the covariant case and avoids multiple inheritance conflicts in requiring that all methods and components have a unique origin.

If all object type definitions in a schema are well formed then the schema obeys the covariant subtyping principle. As usual, the object generating method `new()` is treated differently, since it can already be bound at compile time.

### 3.3 Conceptual schema definition

Before we discuss the realization of the security constraints presented in the previous chapter, we will briefly introduce the syntax and semantics of the schema definition language used in eXoT/C. Fig. 3 illustrates the implementation of the conceptual schema as described in section 2.1.

The structural aspects of the three OMT classes are represented as object types. Attributes and method signatures are specified in a straightforward manner. The associations of the OMT model are implemented as attributes in the object types which are used to reference the objects at the other end of the association. In this example, we arbitrarily chose to support bidirectional navigation by realizing such attributes on both sides of the relationships (e.g. `projects` and `members` in object types `Employee` and `Project`).

The three containers implement the extensional part of the model serving as persistence roots and entry points for queries.

The implementations of the methods are also included in the schema. For the sake of a focused presentation, some minor data type conversions were deliberately left out.

```
schema Enterprise {
object Employee {
  ssn : string;
  name : string;
  salary : money;
  birthday : date;
  sex : char;
  projects : set(Project);
  age() : integer;
}

object Manager : Employee {
  bonus : money;
  guided_projects : set(Project);
  total_budget() : money;
}

object Project {
  title : string;
  budget : money;
  category : string;
  leader : Manager;
  members : set(Employee);
}

container TheProjects : Project;
container TheEmployees : Employee;
container TheManagers : Manager;

method age(): integer in Employee {
  return today()-self.birthday;
}

method total_budget() : money in Manager {
  return sum(select budget
             from p
             in self.guided_projects);
}
} // Enterprise
```

**Fig. 3 Conceptual Schema in eXoT/C**

## 4 IMPLEMENTING SECURITY CONSTRAINTS

In this section we will show how our view mechanism can be used to achieve the desired level of security as specified in fig. 2. The main principle is likewise straightforward and elegant.

### 4.1 Schema mapping

We use the conceptual schema as a basis from which we construct an external schema (view). The schema derivation is specified through a slightly extended form of the schema definition language which was already used for conceptual schemas. Through the derivation specification we define a mapping between the conceptual schema and the external schema. The user of an external schema is neither aware of the underlying conceptual schema nor of the details of the mapping used to derive his view.

The elements of the external schema (derived types and containers) are described on the basis of conceptual schema elements. Type derivation allows one to use either all characteristics of a conceptual type or to restrict the external representation to a subset of the conceptually known attributes and methods. External types can also be extended via the specification of new methods. For external containers a subset of the conceptual extensions can be specified via a query expression.

All of the possible mapping mechanisms can be found in fig. 4 which shows the external schema mapping necessary to implement the security constraints presented in section 2.3. We will take a closer look at this mapping in the next section.

## 4.2 Security constraints revisited

We will explain the schema mapping in fig. 4 according to the classification of security constraints presented earlier.

### 4.2.1 Simple constraints

The derivation of the external type `Project` from the conceptual one takes the confidentiality of `budget` into account. The attribute does not appear in the projection list of the external type definition. The external appearance of projects is therefore "budgetless". Type restriction as applied in this case can be used to conceal the existence of conceptually defined attributes or methods from the user of the external type. Type projection typically implements simple security constraints (cf. sec. 2.2.1).

The projection mechanism is also used in the derivation of type `Employee`. Since attribute `salary` was specified to be secure, it is omitted from the projection list.

### 4.2.2 Content constraints

The implementation of a content security constraint can also be seen in the derivation specification of Employee. As the classification of attribute `birthday` depends on the sex of the employee (cf. sec. 2.2.2), the attribute value must be hidden for female employees while it is unclassified for male ones. Constraints of this type can only be dynamically evaluated at run time. The way to implement such constraints in eXoT/C consists of two steps. First, the dependent attribute (or method) has to be projected away by type restriction as mentioned above. This implies that no instance of the type has such an externally observable attribute. But since this is too restrictive according to the security requirements (the birthday of males is unclassified), in the second step, we need to construct a mechanism by which the unclassified values can be retrieved. It is rather evident to implement an additional method for `Employee`, which dynamically evaluates the condition of the content constraint. In the example, the externally defined method `birthday()` only returns the birthday if the method is sent to an object of type `Employee` with `sex='M'`. If the birthday of a female employee is requested, the method could either return a cover story, a null value or a special value. For the example we use the special value `N/A` (not available).

Method `birthday()` in fig. 4 is notable for the reason that it shows the access mechanism for conceptually defined attributes which were excluded from the external type definition by projection. The actual conceptually defined attribute is denoted by `object.attribute@`. In the case of `birthday()`, we use `self.birthday@` to access the conceptual attribute.

```
derive schema UnclassifiedEnterprise
from Enterprise {

derive Project {
  from Project {
    title: string;
     category : string;
     leader : Manager;
     members : set(Employee);
  }
}

derive Employee {
  from Employee{
    ssn : string;
    name : string;
    sex : char;
  }
birthday() : date;
age() : integer;
projects() :set(project);
}
method birthday() : date in Employee {
  if (self.sex = 'F')
     return (N/A)
  else
    return self.birthday@;
}
method age():integer in Employee {
  if (self.birthday()=N/A)
    return N/A
  else
    return self.age()@;
}
method projects() : set(Project) in Employee {
   return
   (select P from p in self.projects@
    where p.category <> 'critical');
}

derive Manager : Employee {
  from Manager {
    bonus : money;
  }
  guided_projects() : set(Project);
  total_budget() : money;
}
method guided_projects() : set(Project) in Manager
{
   return
   (select P from p in self.guided_projects@
    where p.category <> 'critical');
}
method total_budget() : money in Manager {
  if (card(self.guided_projects()) < 3)
    return (N/A)
  else
    return sum(
             select budget from p
             in self.guided_projects());
}


container TheEmployees : Employee =
     select E from E in TheEmployees@;

container TheManagers : Manager =
     select M from M in TheManagers@;

container TheProjects : Project =
     select P from P in TheProjects@
     where P.category <> 'critical';

container TheSalaries : money =
     select salary from E in TheEmployees@;

}; // UnclassifiedEnterprise
```

**Fig. 4 External schema definition**

It is important to stress the fact that this mechanism for *inter schema referencing* does not offer users a way to circumvent the security restrictions. The @-notation can solely be used by the designer of the schema mapping who implements the desired security boundary. There is no means for the user of an external schema to make any forbidden use of this feature even if he should gain access to the mapping specification or the description of the conceptual schema (which shouldn't be the case normally).

The @-notation allows the external schema designer to make use of all attributes and methods in the conceptual schema. Decisions about the classification of an attribute or return value of a method can therefore be made with the whole database being accessible.

This power does also imply the possibility to implement a "covert channel". It is the responsibility of the designer of the schema mapping to check for and eliminate such security holes. Since much of the security restrictions are implemented structurally via types this task is not so problematic as it may seem to be at the first glance. When a conceptual type is restricted via projection, the resulting external type definition is used to describe the observable structure of the conceptual instances for the user of the external schema. In the context of our example this means that projects do not have a budget, independent of the place where they occur (e.g. as attributes `projects` in type `Employee` and `guided_projects` in type `Manager`).

### 4.2.3 Complex constraints

Complex constraints (cf. sec. 2.2.3) can be implemented in a way rather similar to that used for content constraints. In our example, projects with 'critical' category are secure. This includes also associations with members and leaders and computations of budget sums. To restrict the view to unclassified projects only, we construct a derived container using an adequate selection criterion. The external container `TheProjects` includes all unclassified projects. The classified ones are never visible to the user. Associations require some careful considerations. While it is not necessary to restrict the navigation from secure objects to unclassified ones, it is essential to explicitly avoid references from unclassified objects to secure ones.

In the example this is realized through the methods `projects()` in `Employee` and `guided_projects()` in `Manager`. Both methods replace the corresponding conceptual attributes which were projected away during type derivation. They simply apply the same selection criterion as used for the container `TheProjects` to the conceptual attributes. Since the whole conceptual database is accessible for the methods by means of the @-notation, arbitrarily complicated complex constraints can be expressed.

### 4.2.4 Level-based constraints

The realization of a level based constraint can be seen in the method `age()` of `Employee`. Ages of employees are classified if their birthday is classified (cf. sec. 2.2.4). The implementation principle is again very similar to that of content based constraints. The conceptual attribute or method is projected away and a new method is introduced in the mapping specification which dynamically checks the classification of the independent attribute. Depending on the result of the check, the real value is returned or hidden.

Had we just projected the conceptually defined method `age()` in the external type definition of `Employee`, then the age of each employee would have been accessible. A conceptually defined method executes in the context of the conceptual schema and can use/call all attributes/methods defined therein. This feature is highly desired from the point of functionality and schema reuse, but viewed in the context of security requirements, it may seem to be a security hole. Had the original security constraints not included the explicit requirements for the classification of ages depending on the classification of birthdays, the schema designer may have forgotten to restrict access to it. As a consequence, birthdays may be inferred up to a resolution of one year by using the unclassified `age()` method.

### 4.2.5 Association constraints

Since association-based constraints disallow the connection between attributes of objects and their actual values (cf. sec. 2.2.5), they can be dealt with like simple constraints as a first step. By projecting away the attributes in the object types, we hide the attributes altogether. But this treatment alone would be too restrictive, since the values would also be unaccessible. We need to offer a way to access the isolated values. In the example we achieve this by the construction of an additional container `TheSalaries` which selects all salary values from the conceptual container for employees. Another possible solution would have been the realization of a second derived type based on the conceptual object type `Employee` with a single attribute `salary` and of a container referring to this type.

### 4.2.6 Inference Constraints

Inference constraints also require to implement additional restrictions of attributes or return values of methods. Possible inference paths must be specified and need special treatment. The example we gave in section 2.2.6 considered the inference from aggregate values to the values they where computed from in the case of just a few such values. Project budgets are classified, total budgets of managers are not. In fig. 4 the conceptual method `total_budget()` in Manager was replaced by a new external method which acts as a guard for the real aggregate value. As our example requires the budget of `'critical'` projects to be excluded from the budget sum, the selection in method `total_budget()` is based on the externally computed `guided_projects()`. If all budget values were allowed to be included in the budget sum, the method code of `total_budget()` would simply need to call the conceptual method `total_budget()` by using the @-notation.

## 4.3 Resulting external schema

The result of the schema derivation process according to the definitions in fig. 4 is presented in fig. 5 (on the next page) which should be self-explanatory. This is the only information about the database structure a user has evidence of. For the user, the mapping specification and the underlying conceptual schema are neither visible nor accessible by any means.

Hence, the object-oriented view approach does not just implement the external schemas and provide the mapping mechanism between the conceptual and external levels but also allows one to use this mapping to enforce a security boundary around the conceptual schema.

```
schema UnclassifiedEnterprise {

object Project {
  title: string;
  category : string;
   leader : Manager;
   members : set(Employee);
}

object Employee{
  ssn : string;
  name : string;
  sex : char;
  birthday() : date;
  age() : integer;
  projects() :set(project);
}

object Manager : Employee {
  bonus : money;
  guided_projects() : set(Project);
  total_budget() : money;
}

container TheEmployees : Employee;
container TheManagers : Manager;
container TheProjects : Project;
container TheSalaries : money;

}; // UnclassifiedEnterprise
```

**Fig. 5 Resulting external schema**

## 5 ASSESSMENT

Object-oriented views or external schemas can be described neither as a plain discretionary nor as a pure mandatory approach for access control. While they have properties of each of the two mainstream kinds of security models, they do not completely fit into one of these categories.

Comparing the external schema approach with discretionary models (e.g. Fernandez et al., 1993; Brüggemann, 1994) we note that the former does not explicitly include the notion of a grant/revoke mechanism. The grant granularity as presented here is a whole external schema which is defined and carefully tailored by a security administrator according to the central security requirements at a "schema definition time". The authorization models are more flexible in allowing users to grant/revoke access rights dynamically at run time. But as already mentioned in the first section of the paper, this also leads to a decentrally defined security "policy" which is a moving target and not enforceable. A security architecture with external schemas as their basic building block and an authorization model on top of it might be a promising framework combining the advantages of both approaches namely flexible delegation of rights within a tight security boundary.

In contrast to security models with mandatory access control (multilevel approaches with single level object representations), the view based realization of security requirements does have minimal influence on conceptual schema design. Even a small example should make this point obvious. In fig. 6, the realization of one simple security constraint is shown. The starting point (which is a simpler version of our running example) is a type `Employee` with a subtype `Manager`. `Employee` has the unclassified attributes `ssn` and `name` and a secure attribute `salary`. The type Manager has an additional unclassified attribute `bonus`.

The multilevel approaches split each of the conceptual types `Employee` and `Manager` into two separate single-level parts. This results in two secure and two unclassified components. The two secure parts are related by inheritance which is also true for the two unclassified parts which is stated in conventional OMT notation. The unfamiliar dashed lines linking the secure and the unclassified components of each of the two conceptual types may be interpreted either as security induced inheritance (Jajodia et al., 1995) or as association between composite objects (Bertino and Jajodia, 1993). Both variants provide means to view just the unclassified parts or to access the whole conceptual object.

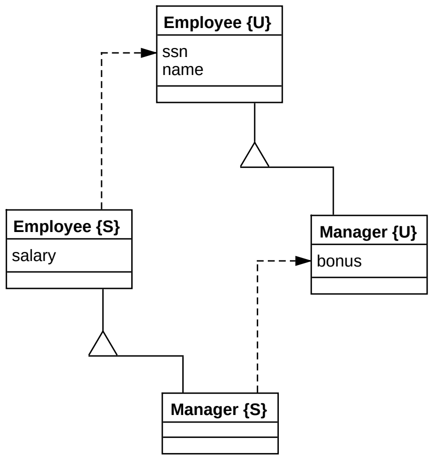

**Fig. 6 Simple constraint and multilevel classes**

In the presence of more classification levels and/or more complex security constraints the single level implementation can lead to rather complicated structures in the conceptual schema. The pure conceptual semantics of the schema are heavily distorted by the security requirements. This class partitioning is not the only problematic issue, others are class rearrangements and object migration. A more detailed discussion of these problems can be found is given by Hochmüller (1996).

A security solution based on eXoT/C allows for semantic inheritance hierarchies and navigation paths in the conceptual as well as in the external schemas. The derivation-based association between conceptual and external schemas is (virtually) orthogonal to traditional relationships between schema elements. This enables us to implement security requirements as inter schema mappings and preserves the original conceptual semantics.

But the virtues of the multilevel security models should not be neglected. The strong policy and its strict enforcement is the central feature. Views do not guarantee such a degree of control and security per se. The flexibility in external schema construction puts the burden of the decision between security and functionality of the external views onto the designer of the mapping. The mandatory models and the view based security approach have distinct strengths and limits making them suitable for different domains depending on the relative importance between rigid security and powerful functionality.

An open question is the feasibility of the mapping construction in the presence of complicated intertwined security constraints (cf. Burns 1992). Tool support for the semiautomatic generation of the derivation specifications starting from a MOMT-like static model would be advantageous for the task of schema design as well as for the resulting security level. A content-based security constraint like "all confidential projects are secure" could for instance be used to automatically generate corresponding filter expressions in all parts of the schema mapping where projects can appear.

## 6 CONCLUSION

Object-oriented views can be used as an effective means of access control and to implement a fine grained security mechanism. Since the usage of components and notably also of methods can be restricted via projection, the definition of external schemas enables the DBA to draw a tight security boundary between the parts of the database the user is entitled to work on and the part he has no clearance for. In particular, the usage of views does not only allow to restrict the rights of an user to access data elements. The power and flexibility of external schemas with respect to security is the ability to put strong confinements on the user concerning his possibilities to apply certain operations on those data objects.

Only the parts of the database, which are explicitly mapped to the outer interface of the external schema are accessible to the user of the external schema, and the way of manipulation of those visible parts can be restricted to any degree. By providing methods for certain selections or updates of the data, the external schema designer can make available operations for the user of the external schema in a strictly controlled way. These methods can be made arbitrarily restrictive on their execution. Checks for permissions of users on schema elements, validation of receivers and parameters of the methods and arbitrary complex access and modification rights can be implemented in such methods.

Nevertheless, these restrictions do not necessarily diminish the power or the usability of the external schemas to an unwanted degree. Since the implementor of the external schema has access to all parts of the whole underlying conceptual layer, he can design and provide quite powerful methods which are not restricted in any sense. Neither the scope of those methods is inherently confined to just a part of the conceptual schema, nor are there any system implied limitations on the operations such methods can execute.

So the designer of the external schema has available the full power of the complete conceptual schema. It lies in his responsibility to construct an adequate interface for users in terms of power and security. Such an external schema should provide only the necessary and sufficient operations on the data objects the users are entitled to see, but it should also be restrictive in terms of validation, plausibility and consistency.

## REFERENCES

Batini, C., Ceri, S., and Navathe, S.B. (1992). *Conceptual Database Design; An Entity-Relationship Approach.* The Benjamin/Cummings Publishing Company Inc.

Bertino, E. and Jajodia, S. (1993). Modeling Multilevel Entities Using Single Level Objects. Ceri, S., Tanaka, K., and Tsur, S. editors, *Deductive and Object-Oriented Databases (DOOD'93)*, pages 415-428, Springer, LNCS 760.

Brüggemann, H.H. (1994). Object-Oriented Authorization. In Paredaens, J., Tenenbaum, L. editors, *Advances in Database Systems; Implementations and Applications,* pages 139-160. Springer.

Burns, R.K. (1992). An Application Perspective on DBMS Security Policies. In Lunt, T.F., editor, *Research Directions in Database Security*, pages 227-233, Springer.

Castano, S., Fugini, M., Martella, G., and Samarati, P. (1995). *Database Security*. Addison-Wesley.

Catell, R.G.G. et al. (1997). *The Object Database Standard ODMG 2.0.* Morgan Kaufmann.

Dobrovnik, M. (1995). *Externe Schemata in objekt-orientierten Datenbankmanagementsystemen; Logische Datenunabhängigkeit durch Änderungen über Sichten*. PhD thesis, Universität Klagenfurt, also pulished as Dobrovnik, M. (1997)

Dobrovnik, M. (1997). *Externe Schemata in objekt-orientierten Datenbankmanagementsystemen; Logische Datenunabhängigkeit durch Änderungen über Sichten*. Infix. DISDBIS 25.

Dobrovnik, M. and Eder, J. (1994). Adding View Support to ODMG-93. In Mizin, I. A., Kalinichenko, L. A., and Zhuralev, Y. I., editors, *Proc. Intl. Workshop on Advances in Databases and Information Systems*, pages 74-81, Moscow, Russia. Moscow ACM SIGMOD Chapter.

Dobrovnik, M. and Eder, J. (1996). Logical Data Independence and Modularity through Views in OODBMS. In *Proc. Engineering Systems Design and Analysis Conference (ESDA'96)*, pages 13-20, Montpellier.

Fernandez, E.B., Larrondo-Petrie, M.M., and Gudes, E. (1993). A Method-Based Authorization Model for Object-Oriented Databases. In Thuraisingham, B., Sandhu, R., and Ting, T.C., editors, *Security for Object-Oriented Systems,* pages 135-150, Washington DC. Springer Verlag.

Hochmüller E. (1996). Inheritance Contradictions between Functional and Extra-Functional Requirements. In *Proc. Second World Conference on Integrated Design & Process Technology (IDPT'96)*, Vol.1, pages 106-113, Austin. SDPS.

Jajodia, S., Kogan, B., and Sandhu, R.S. (1995). A Multilevel Secure Object-Oriented Data Model. In Abrams, M. D., Jajodia, S., and Podell, H. J., editors, *Information Security - An Integrated Collection of Essays*, pages 596-616. IEEE CSP.

Kim,W. and Kelley, W. (1995). On View Support in Object-Oriented Database Systems. In Kim, W., editor, *Modern Database Systems. The Object Model, Interoperatibility and Beyond.* ACM Press.

Lunt, T.F. and Fernandez, E.B. (1990). Database Security. *ACM SIGMOD RECORD*, 19(4):90-97.

Marks, D.G., Sell P.J., and Thuraisingham, B.M. (1996). MOMT: A multilevel object modeling technique for designing secure database applications. *JOOP* 9(4):22-29.

Millen, J.K. and Lunt, T.F. (1992). Security for Object-Oriented Database Systems. In *Proc. Symposium on Research in Security and Privacy*, Oakland, CO. IEEE CS Press.

Motschnig-Pitrik, R. (1996). Requirements and comparison of view mechanisms for object-oriented databases. *Information Systems*, 21(3):229-252.

Pernul, G. (1994). Database Security. *Advances in Computers*, 38:1-72.

Pernul, G. (1995). Information Systems Security: Scope, State-of-the-art, and Evaluation of Techniques. *International Journal of Information Management,* 12(3):165-180.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991). *Object-Oriented Modeling and Design*. Prentice Hall.