Security In Databases: A Survey Study

Ahmad Baraani-Dastjerdi *Josef Pieprzyk*
Reihaneh Safavi-Naini

Department of Computer Science
University of Wollongong
Wollongong, NSW 2522
AUSTRALIA
e-mail: [ahmadb,josef,rei]@cs.uow.edu.au

February 7, 1996

Abstract

In this paper, we survey the security of coventional databases and object-oriented databases that have been reported in the current literature. This is an area of substantial interest in databases because (1) the use of databases is becoming very important in today's enterprises, (2) databases contain information that is a major enterprise asset. Security concerns, requirements, and problems that arise in the pursuit of meeting these requirements for security in databases are illustrated. We discuss access control issues in databases, and futher identify some promising research directions.

1 Introduction

Information is a critical resource in today's enterprises, whether they are military, industrial, commercial, educational, medical, etc. These organizations are now automating not only their basic operational functions, such as invoicing, payroll, and stock control, but also management-support functions such as sales forecasting, budgeting, and financial control. In order to support these functions, enterprises maintain more and more computerized information in databases, and they increasingly depend on that information for their correct functioning. The continued successful operation of an enterprise demands that:

1. *Confidential data is available only to authorized persons, so that privacy requirements are satisfied and enterprise information is protected.*

2. *The data accurately reflects the state of the enterprise, that is, the data is protected against either malicious or accidental modification.*

These requirements are of the concern of *database security.* Database security comprises a set of measures, policies, and mechanisms to provide *secrecy, integrity,* and *availability* of data and to protect the system from possible attacks which might be launched by insiders and outsiders, either malicious or accidental [23, 111]. The aim of *secrecy* (or confidentiality) is to keep information unreadable for outsiders while making it available for *authorized* users. *Integrity* of information covers methods and techniques to protect information against illegal modification. *Availability* of information ensures that authorized users can have access to information whenever they required [32]. To achieve security in the database environment, it is necessary first to identify threats so later a selection of proper security *policies* and *mechanisms* can be made. Security policies define *what* the security system is expected to do [104]. Security mechanisms define *how* the security system should achieve the security goals [9].

In order to obtain database protection, Denning [32] lists four kinds of control: *access control, information flow control, cryptographic control,* and *inference control.* An *access control* ensures that all direct accesses to the system are authorized according to access rules given by the security policies. The access control governs who can access objects. Once the access is granted, the involvement of the access control ends. Often, leakage of information happens not because a defective access control, but as the result of lack of a proper policy about information flow. An *information flow control* makes sure that protected information "contained" in some objects does not flow explicitly (through copy) or implicitly into less protected objects, and regulates how the information is accessible (irrespective where it is stored). The *cryptographic control* makes data unintelligent to anybody except someone who knows the correct cryptographic key. The *inference control* protects information against its disclosure via different ways of deduction.

In this paper, a survey of the access control models is presented. The emphasis is put on access control models in OODBSystems. The survey discuss neither the application of cryptographic techniques in databases security nor inference control. For references, see [2, 23, 32, 63, 86, 107, 108, 123, 145].

Sections 2 and 3 state threats to database security and the security requirements for databases, respectively. Section 4 describes the concept of the access control, the policy choices, and classifies the access control policies. In Sections 5 and 6, discretionary access control in conventional databases and OODBS are discussed. Sections 7, 8, and 9 are devoted to mandatory access control (MAC). The Bell-LaPadula security model, MAC models in conventional databases, and MAC models in OODBS are presented. A brief discussion of role-based access control is presented in Section 10. Three types of architecture for multi-level databases are discussed in Section 11. Sections 12 and 13 provide a general concepts of inference control and cryptographic control. Section 14 summaries the paper.

2 Threats to Database Security

Usually, a *threat* can be identified with a hostile agent who either accidentally or intentionally gains an unauthorized access to the protected database resources [23]. Threats to the database security may be physical or logical. *Physical threats* range from a forced disclosure of passwords, a theft, a destruction of physical storage devices to a power failure. The protection against these threats comprises a variety of different methods and techniques. The restriction of physical access to database storage facilities and the database backup and recovery are common protection methods.

Logical threats involve unauthorized logical (i.e. via software) access to information. They can result with: disclosure of confidential information, illegal modification of data, or destruction of database resource. The logical threats can be classified as follows [23, 34]:

- *Disclosure of information* includes direct or indirect access (by inference) to protected information.
- *Illegal modification of data* caused by improper possible accidental data handling or intentional modifications by an illegal user (these threats related to all attacks to data integrity).
- *Denial of Service* caused by monopolizing system resources in such a way that other users cannot access them. This involves all attacks on availability.

These threats can occur intentionally or accidentally. *Accidental threats* include: *natural disasters* such as earthquakes or water/fire damage, *errors or bugs in hardware or software*, or *human errors*. *Intentional threats* can be caused by *authorized users* who abuse their privileges and authorities, or by *hostile users* who execute some hidden codes within some legitimate

functions in order to violate the security of the system. *Trojan Horses, viruses,* and *trapdoors* are examples of hidden codes. A Trojan Horse is hidden code that under a legitimate function collects information which is later used to break the security of the system. A virus is a code able to copy itself and damage permanently the environment where it gets reproduced. Trapdoors are code segments within programs that allow their owners to skip the protection mechanisms and to access data or system sources beyond their rights.

3 Database Security Requirements

To eliminate logical threats, it is necessary to define a proper security policy. The database is considered to be secure if a protection mechanism correctly enforces the security policy. Note that there can be many security mechanisms which realize the same security policy.

A security policy must have appropriate security features. These features should be supported by a security mechanism. The following list gives features of a security policy for databases [23, 37, 107].

- **Access control policy** ensures that all direct accesses to the system objects proceed according to the privileges and the access rules. Access control policies can be either *mandatory* or *discretionary.* A *discretionary access control policy* specifies users' privileges to different system resources, including their ability to transfer their privileges to other users. A *mandatory access control policy* restricts the access of users to system objects on the basis of *security labels* assigned to objects. Mandatory access control policies are of concern in multi-level databases.

- **Inference policy** specifies how to protect classified information from disclosure when the information is released indirectly in the form of statistical data.

- **User identification/authentication policy** indicates the requirements for correct identification of users. The user identification is the basis of every security mechanism. Users are allowed the access to data after identification as *authorized* users.

- **Accountability and audit policy** provides the requirements for the record keeping of all accesses to the database. It is an useful deterrent tool for data physical integrity, as well as it is useful for analysis of the access profile.

- **Consistency policy** defines the states in which the database is considered valid or correct and includes *operational integrity, semantic integrity,* and *physical integrity of database.* Operational integrity aims to ensure the logical consistency of data in a database during concurrent transactions. Semantic integrity ensures the logical consistency of modified data by the controlling data values in the allowed range. Physical integrity of database aims to ensure that the database is immune to (or reconstructable after) physical threats such as power failures.

Most of the effort of the research in the database security has been spent on the first two issues.

It is crucial to be able to evaluate to what degree the security features have been incorporated into the mechanism. The evaluation of security (trusted) systems (mechanisms) is an integral part of the design process. Here is a list of evaluation criteria:

- US Department of Defense (*Trusted Computer System Evaluation Criteria (TCSEC)* known as the *Orange Book*) [38],

- Commission of the European Communities (*Information Technology Security Evaluation Criteria (ITSEC)* known as the *White Book*) [25], and

- Canadian System Security Centre (*Canadian Trusted Computer Product Evaluation Criteria (CTCPEC)*) [22].

The evaluation of trusted systems (products) can be characterized by:

1. *Completeness.* This is measured by a list of possible threats against which the system is immune.
2. *Confidence.* The degree of trust that the system is immune to the specified threats.
3. *System flexibility.* The system should be able to implement different variants of security policies.
4. *Ease of use.* The system should not impose unnecessary or cumbersome restrictions.
5. *Tamperproof.* If a security mechanism itself is protected from unauthorized modification, then this mechanism is said to be tamperproof. This is an essential characteristic, since even if a security mechanism were proved to be correct, any later modifications to it could degrade its security.
6. *Low overhead.* The difference in performance of the system with and without its security mechanism should be as small as possible.
7. *Low operating cost.* This includes the cost of special hardware or software, salaries of security officers and others who involve in performing security-related functions, and cost of security audits.

4 Access Control

There are two classes of resources in any computer system: *(active) subjects* and *(passive) objects*. The ways a subject accesses an object are called access privileges (or access modes). Access privileges allow subjects to either manipulate objects (read, write, execute, etc.) or modify the access control information (transfer ownership, grant and revoke privileges, etc.).

The correctness of access control heavily relies on the following:

- *the proper user identification, and*
- *the protection of access control mechanism.*

The access control may be based on different policies. The choice of a security policy is important because it influences the flexibility, usability, and performance of the system [47]. While working out a proper security policy, it is necessary to proceed with it according to "a good design guide" which includes the following principles [23, 47, 85]:

Minimum vs maximum privilege principle. According to *minimum privilege principle*, subjects should use the minimum set of privileges necessary for their activity (also called the least privilege). The opposite of this is *maximum privilege principle* which is based on the principle of the maximum availability of data in a database. Subjects have access to the largest range of information.

Open vs closed system principle. In an open system, all accesses that are not explicitly forbidden are allowed. While in a closed system, all accesses are allowed only if explicitly authorized. A closed system is inherently more secure. When security is an important objective, a closed system should be implemented.

Centralized vs decentralized administration principle. The principle addresses the issues who is responsible for the maintenance and management of privileges in the access control model. In centralized administration, a single authority (or group) controls all security aspects

of the system, while in a decentralized system different authorities control different portions of the database. The choice between centralized or decentralized administration has to be made. There are some advantages and disadvantages related to each choice (for detailed discussion of those, the reader is directed to [16]). There are, however, some intermediate choices such as: *delegation*, *owner-based administration*, and *co-operative administration*.

Delegation can be used in a centralized database system to avoid bottleneck and support local autonomy in a distributed database system. A central authority delegates their administrative rights for subsets of the database to local authorities. The central authority can nominate and dismiss local authorities.

Owner-based administration - the central authority passes all responsibilities and rights to the owners of objects. The central authority only controls the data, and admits new users to the database system.

Co-operative administration - some privileges are reserved to groups. To exercise the access, a single member of the groups needs to get permission from the rest of the group.

Granularity principle - a clear specification of size and structure of smallest object to which access control can be handled. Five types of granularity can be distinguished [32, 47]:

1. *Name-dependent* (also called content-independent). All accessible objects are identified by their unique names.

2. *Content-dependent* (also called constraint-based). Accessible objects are determined by checking the requested attributes. This granularity can be vary fine depending on the selection of the applicable attributes.

3. *Context-dependent.* The access is granted to the object whose structure is defined by a rule which checks not only names and attributes but also their structural relation (context).

4. *History-dependent.* This is a generalization of the context-dependent granularity in which the current access request is checked also against all previous access requests.

5. *Time-dependent.* The collection of accessible objects can vary in time (for instance, a subject may be authorized to read from the class *EMPLOYEE* only between *8:00am* and *5:00pm*).

Access privilege (or access mode) principle. A basic collection of allowable access privileges includes: *read, write, delete, execute,* and *create.* They may be ordered. So if a user (subject) has a privilege of higher order, this implies that (s)he has also all privileges of lower order. Users can be assigned explicitly a collection of suitable privileges for each protected object - this is also called *positive authorizations.* On the other hand, in the *negative authorizations*, users are given a collection of privileges which are explicitly denied.

There are three different types of access control: *discretionary access control (DAC), mandatory access control (MAC), and role based access control (RBAC).* A system may employ DAC, MAC or their combination of both, or RBAC for protection.

A *discretionary access control* specifies privileges of subjects in accessing objects, and the rules whereby subjects can, at their discretion, grant and revoke their privileges to other subjects.

A *mandatory access control* identifies the rules whereby subjects can obtain direct or indirect access to classified data. The rules can also be used for sanitizing and reclassifying data. The MAC applies only to *multi-level databases,* which are databases that contain information of different security classification.

A *role-based access control* enforces the least privilege and the separation of duties. This is especially important in some database applications where subjects (users) can be assigned roles to perform specific tasks (defined by the role). Each role has an associated collection of privileges. This collection is automatically transferred to a subject who plays the role.

5 Discretionary Access Control Models

Discretionary Access Control (DAC) allows privileges to be granted to other subjects by the object owners. In discretionary security the way in which individual users (subjects) manipulate specific objects is specified through explicitly access rules. They are fundamental to operating systems (as a means of protecting files, memory segments, etc.) as well as database systems. DAC has been studied in the context of the *access matrix model*.

5.1 Access Matrix Model

The access matrix model was developed by Lampson in 1971 [84] and extended by Graham and Denning [57]. Later, Harrison, Ruzzo, and Ullman [66] developed a more general version of the model. They defined the safety problem and showed that it was undecidable. The access matrix model is defined for three components: *subjects* (active entities such as users, their processes, etc.), *objects* (passive entities such as files, records, classes, instances, views etc.), and the collection of *privileges* (read, write, delete, create, execute, etc.). Note that the class of objects contains all subjects. Having these three components, it is possible to define an access matrix A. Rows of A are indexed by all subjects (their names) (S) and columns - by names of all objects (O). Each entry $A[s, o]$ contains a collection of privileges held by subject s to object o. A representation of an access control matrix is shown in Table 1.

	Objects				
Subjects	O_1	...	O_j	...	O_m
S_1	$A[s_1, o_1]$		$A[s_1, o_j]$		$A[s_1, o_m]$
.					
.					
S_i	$A[s_i, o_1]$		$A[s_i, o_j]$		$A[s_i, o_m]$
.					
.					
S_n	$A[s_n, o_1]$		$A[s_n, o_j]$		$A[s_n, o_m]$

Table 1: Access matrix

The access control matrix can be used for protection in both operating systems and databases. In databases, access control matrix needs to be extended. Every entry $A[s, o]$ of the matrix contains (apart from privileges) a suitable condition which needs to be satisfied by the subject s to access the object o. The condition can incorporate different types of access such as content-dependent, context-dependent, etc. (for full list of access types, see Section 4). Fernandez [46], and Conway [27] showed how to generalize the access matrix model by using predicates and other components.

Observe that in general, an access control matrix A is sparse. A direct storage of the matrix A wastes a lot of memory. A simple solution is to store the matrix A as a sequence of either rows (*Capability List*) or columns (*Access Control List*).

For each subject s, there is the unique capability list (row of A). Each element of the list indicates an object and the collection of privileges the subject has to the object. Note that the list does not contain empty entries. If there is no list entry for a given object, the subject cannot access it. Capability lists (CL) allow the system to identify quickly the collection of all accessible objects for a given subject. The opposite, i.e. finding the collection of all subjects who have access to a given object, is difficult and time consuming. Capability lists are used by the operating systems only [37].

Access control lists (ACL) are associated with their objects (columns of the access control matrix A). For a given object, the list consists of all non-empty entries of the column of A. This implementation allows for quick identification of subjects who can access an object. On the other hand, the recreation of the capability list for a subject from ACL is difficult. ACL are used in both operating systems and database systems [37].

Note that in both implementations the storage requirements increase with the growth of the number of subjects and objects. In particular, the maintenance of such lists is expensive in terms of time and consumed computing resources.

However, the access matrix provides a flexible model which can be used to analyze its security properties. It is known [23, 95] that the general safety problem is undecidable, i.e., there is no algorithm which can be used to verify the security of the access control matrix model. But it is still possible to restrict the model and design an algorithm which can be used to prove some security properties. Some work has been done to extend the access matrix model to make the safety problem decidable. This includes the *Schematic Protection Model* [6, 113, 114] and the *Typed Access Matrix Model* [116].

6 DAC Models in OODBS

In recent years there has been considerable efforts in the research and development of object-oriented databases. ORION [82], ONTOS [105], O_2 [39], Objectivity/DB [101], Iris [143], and VERSANT [142] are examples of such efforts. The driving forces behind these efforts are the advantages offered by O-O approach to database modeling. In particular it is possible to represent the composite (or complex) structure of objects and to simulate the behavior of objects through operations encapsulated within the classes. In order to fully exploit the benefits of the O-O paradigm, it is important to consider how the O-O data model impacts on access control models. To avoid confusion, we will use the term *entity* to refer to a passive item in protection system instead of object, and the term *object* has the meaning usually associated with the O-O environment. Note that the term *object* is usually used in literature for such a passive item. Let us consider the following important issues related to DAC in O-O environment [16, 23, 93, 127, 131].

- *Access privileges (or access modes).* One of attractive features of OODBS is *encapsulation.* Encapsulation allows us data to be stored as values of properties (or instance-variables) that are encapsulated in an object and available only through the methods defined for the object. In order to take advantage of the encapsulation feature, the access control model (or authorization model) should support privileges to execute these methods (instead of traditional privileges such as read, write, and create). Moreover since methods can call other methods, it would be appropriate also to consider methods as subjects.

- *Propagation of privileges (or authorizations).* OODBSystems allow that classes are organized into a hierarchy of classes. There are two types of hierarchies which are orthogonal to each other: a *class-composition hierarchy* and a *class-inheritance hierarchy* [83]. The class-composition hierarchy captures the *is-part-of* relationship between a parent class and its component classes. Whereas the class-inheritance hierarchy represents the *is-a* relationship between a superclass and its subclasses. There is a question - how to assign privileges to objects in these hierarchies. It means for the class-composition hierarchy, whether a privilege that a subject has to a class, is valid also to all descendants of the class (*visibility from above* [85]). Or for the class-inheritance hierarchy, whether a privilege to a subclass is valid to the property values of the superclasses on the higher levels of the hierarchy (*visibility from below* [85]). Additionally, OODBS must provide a policy for solving possible authorization conflicts between explicit privileges (privileges assigned directly) and implicit privileges (privileges derived through the hierarchies).

- *Authorization administration.* In most access control models when a user (subject) creates a new element, (s)he becomes its owner. The owner controls the distribution of privileges to it. So, the unit of ownership cannot be a class since classes often represent just templates for users to derive their own instances. Hence, there is a need to allow class instances to be units of ownerships. Being more specific, the access control policy must clearly spell out how the class owner can affect the privileges of other users especially in the context of instance of, class-inheritance, and class-composition relationships.

- *Granularity of authorization.* It should be possible to control access to single object-instances, entire classes, and properties (instance-variables) [109]. For the *class-based authorization,* a subject can access a class and all its instances. For the *instance-based authorization,* instances of a class are units of authorizations. A subject may have authorization for a subset of the instances of a class. For *instance-variable-based authorization,* access control on instance-variables or properties is allowed. A subject can be limited to access only a part of an object. Moreover, the model of access control should take into account semantic constructions such as composite objects and versions.

- *Content-based authorization.* Authorization may be defined in terms of the ability to invoke a certain method on an object. The access control policy must specify how we can access the object content for checking the condition if a method which can access the required properties cannot be invoked?

Access control models for OODBS are still being investigated [11]. Although there is an undeniable progress in the area, there are few OODBS only (Orion [109] and Iris [3]) whose provides access control models similar to those provided by current relational databases.

Dittrich [40] divided OODBS into three classes: *structurally, behaviorally,* and *fully* OODBS. *Structurally* OODBSystems can handle composite (or complex) object structures (i.e. objects that may result from aggregating other objects) using generic privileges. *Behaviorally* OODBSystems provide an interface to deal with objects and methods on different levels of the inheritance hierarchy. *Fully* OODBSystems combine the properties of structurally and behaviorally OODBSystems.

Access control models for these classes of OODBSystems were discussed in [3, 12, 13, 14, 43, 45, 60, 85, 109, 110]. The models promise to offer some solutions to the forthcoming issues. However, each of them addresses only some of the issues, therefore many problems remain open.

6.1 DAC in Structurally OODBS

In structurally OODBS, the privileges are typically generic system-defined operations such as *read, write, delete,* and *read-definition.*

6.1.1 DAMOKLES Access Control Model

Dittrich, Hartig, and Pfefferle [43] developed a DAC model for the DAMOKLES system [41]. DAMOKLES is a structurally OODBSystem for CASE and similar applications and its data model called DODM (design object data model) [42]. Two type of objects are supported by the model: DODM objects, and DODM relationships. The access control allows a user u to grant a privilege to other users if u is the owner of an object. When a user creates a database object, an object or a relationship, (s)he becomes the owner of the created object. The ownership may be transferred (by the owner) to other users or user groups. However, at any time there is exactly one owner for every object. In other words, the ownership is transferable but is not transitive. In the model, each access must be explicitly authorized. The absence of appropriate privileges is interpreted as *access not allowed.* The model handles both composite objects and versioned objects as well.

The model applies the following access privileges:

- *Exist* - this privilege allows a user to read keys of objects.
- *Read* - permits a user to read object properties, objects components and relationships roles.
- *Write* - allows a user to modify objects properties and roles, to insert new components/create new versions, and to remove components/delete versions.
- *Delete* - enables a user to remove objects.

The privileges are partially ordered. If a user holds a privilege of higher order, then they also hold all privileges of lower order. The assumed order is: *exist* $<$ *read* $<$ *write* $<$ *delete.*

The triple (s, o, r) specifies a single access rule. It means that a subject s is assigned an access privilege r to an object o. The subject s is a pair $s = (u, p)$ where u is a user or a group of users and p is a program or program group. The pair (u, p) can be read as "user u while executing p". The object o is a protection object (p-object). In DAMOKLES data model (DODM), every object (in the O-O sense) is further broken down into smaller access units called *protection objects (p-objects).* These p-objects are: the descriptive part D (object's properties), the structural part S (the components/composite objects), version part V (the object's versions), and role part ρ consisting of the relationship's roles that is participating objects. Note that the access is explicitly granted to the D part of a p-object. However once granted, the access extends to all object's properties.

There are two types of authorization: *simple* and *complex.* In the simple authorization, a p-object o is D, S, V, or ρ part of a complex object. In the complex authorization, a privilege is given to an entire object including its components which belong to the same owner. Thus if an object actually contains components of various owners, a subject has to get permission from all of them to be able to work with entire object.

6.1.2 LPGSF Access Control Model

Larrondo-Petrie, Guides, Song, and Fernandez [60, 85] developed an access control model (LPGSF model). The model is based on a set of policies that define authorization inheritance through class hierarchies. Negative authorizations can be used to override implied privileges. Predicates (content-dependent) with positive authorization and instances along the class hierarchy can also be applied. The model allows a decentralized administration of authorizations by users.

The access control model uses the following access entities as elementary protected objects: classes, instances of classes, and their properties (or instance-variables).

An access rule is a tuple $(s, o, r, [c])$. The subject s is a user or a user group. The r is a privilege or a set of privileges. They can be positive or negative. The object o is a class, an entire object O or its components, or a set of its properties, i.e., $o = \{O.p_1, O.p2, \ldots\}$. Properties p_i $(i = 1, 2, \ldots)$ must be either defined for object O or inherited by it. c is a condition that must be satisfied for the object o that the subject s can use the privilege r. The model enforces the following policies:

Inheritance policy. *A user that has access to a class is allowed to have the same type of access to subclasses and to the properties inherited from the class satisfying the inherited conditions.*

Visibility from below policy. *The access to a class implies the access to the properties defined in the class as well as to properties inherited from the higher classes (this is applicable to the class-relevant values of these attributes only). If there is more than one ancestor (in the case of multiple inheritance), the access to the union of the inherited properties is granted. Note that properties defined in a subclasses are not accessible by any of its superclasses.*

Visibility from above policy. *The access to an object of a composite object implies the access to all components of the object.*

Since the model supports negative and positive authorizations, the following policy intends to solve possible authorization conflicts. The notation "[...,...,...]" is used to express implicit authorizations whereas "(...,...,...)" is used for explicit authorizations.

Overriding policy. *An explicit positive authorization is a triple* $(s, o, +r)$. *An explicit negative authorization is a triple* $(s, o, -r)$. *An implicit positive authorization is a triple* $[s, o, +r]$. *An implicit negative authorization is a triple* $[s, o, -r]$ *where* s *is a subject,* o *is an object, and* r *is an access privilege. We have the following order:* $(s, o, -r) > (s, o, +r) > [s, o, -r] > [s, o, +r]$.

6.1.3 ORION Access Control Model

In 1991, Rabitti, Bertino, Kim, and Woelk [109] developed a DAC model which has been implemented for ORION [82]. Their model handles the following types of authorization: *implicit*, *explicit*, *positive*, *negative*, *strong*, and *weak*. Implicit authorizations are deduced from explicit authorizations stored by the system. Positive authorizations grant users access privileges to objects whereas negative authorizations (or lake of authorizations) deny access privileges to objects for users. A *strong authorization* cannot be overridden whereas weak authorizations can be overridden by a strong or other weak authorization. Triples $(s, o, +r)$ and $(s, o, -r)$ denote a *strong positive authorization* and a *strong negative authorization*, respectively. Triples $[s, o, +r]$ and $[s, o, -r]$ are a *weak positive authorization* and a *weak negative authorization*, respectively. As usual, s (subject) is a user or a group of users, o is an entity protected by the system, and r is an access privilege.

In the model, two access bases are defined. The *strong access base SAB* contains all explicit strong authorizations including both positive and negative strong rights. The *weak access base WAB* contains all explicit weak authorizations including both positive and negative weak rights.

A privilege r is granted to an object o for a subject s (or the access request (s, o, r) is valid) if the return value of the following function is true.

Function f(s, o, r)

if *there exists an explicit or implicit* $(s, o, +r)$ *in SAB*
then return *True*
else if *there exists an explicit or implicit* $(s, o, -r)$ *in SAB*
then return *False*
else if *there exists an explicit or implicit* $[s, o, +r]$ *in WAB*
then return *True*
else if *there exists an explicit or implicit*
$[s, o, -r]$ *in WAB*
then return *False*

end

Note that the function works correctly if there exist at least a weak authorization for each possible privilege.

The cornerstone of the discussed access control model is the implicit authorization which can be propagated along each of the three dimensions (parameters) of access rule, namely, subjects, entities, and access privileges. To reduce the number of subjects involved in explicit authorizations, users and groups of users are assigned roles. Roles can be arranged into a *role hierarchy* (see Figure 1).

The following two rules govern the propagation of authorization for roles.

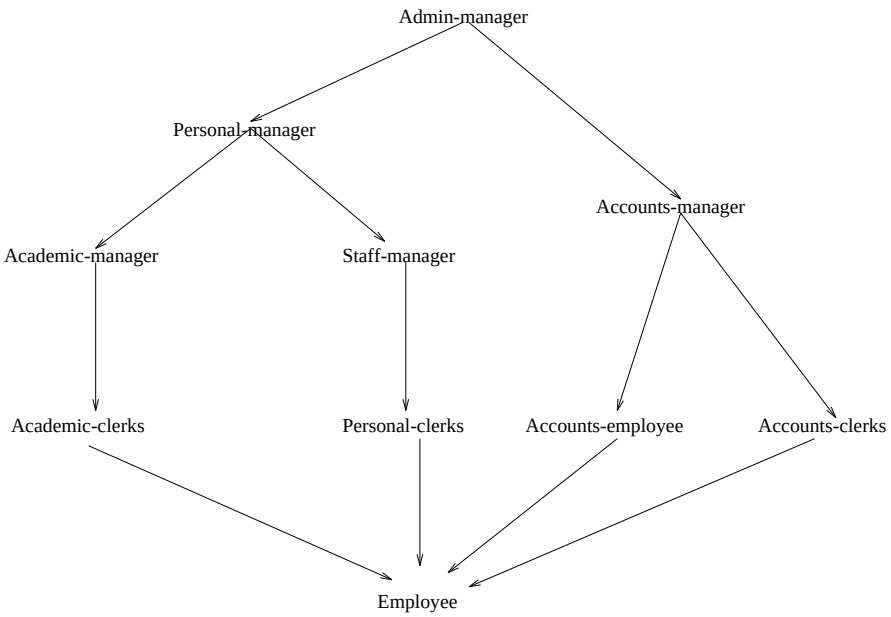

Figure 1: A sample of role hierarchy.

Implication Rule 1 *Explicit positive authorizations for some roles result in implicit positive authorizations for all higher-level (upper) roles.* □

Implication Rule 2 *Explicit negative authorizations for some roles result in implicit negative authorizations for all lower-level roles.* □

Access privileges also create a privilege hierarchy.

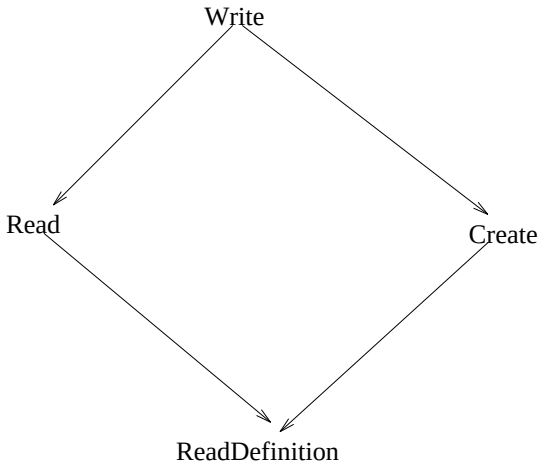

Figure 2: A sample privilege hierarchy.

The next two rules specify how to generate authorizations along the privilege hierarchy.

Implication Rule 3 *If a positive authorization is given for an access privilege in the privilege hierarchy, then this implies the right of all access privileges below it.* □

Implication Rule 4 *If a negative authorization is given for an access privilege in the privilege hierarchy, then this also implies the negative authorization of all access privileges above it.* □

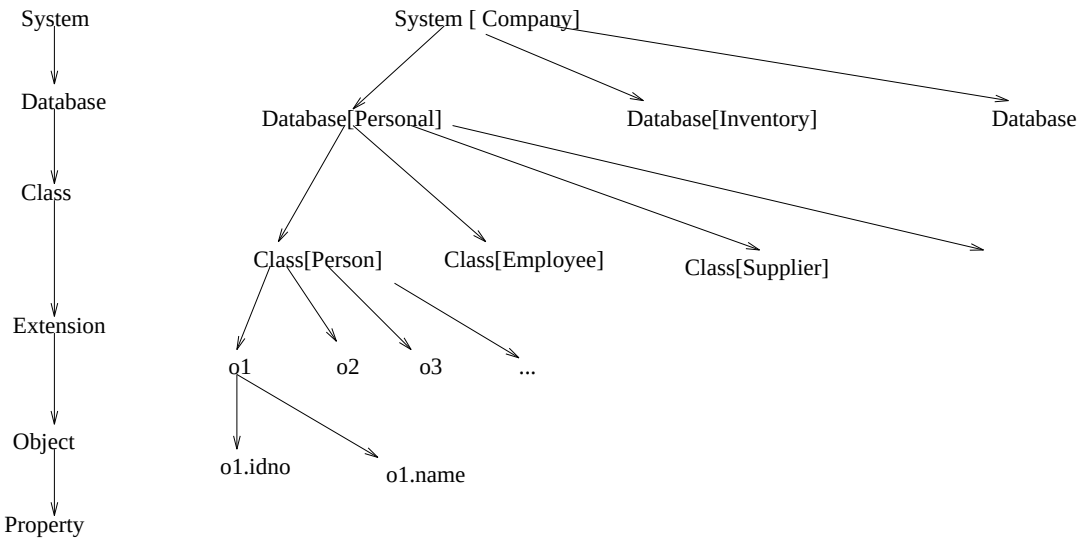

Figure 3: A sample Entity Schema and Entity Hierarchy.

Entities are also organized as the *entity hierarchy*. The entity hierarchy is the instantiation of the *entity schema*. Figure 3 shows an example of the entity hierarchy and the entity schema.

The deduction of authorization in the entity hierarchy depends on not only the entity but the mode of privilege involved as well. For example, the *read* and the *write* access privileges for certain objects require the corresponding class definitions to be readable. The direction of this implication is *upward* in the entity hierarchy. On the other hand, the *read* permission on the extension of a class implies the *read* permission on all the instances of that type. In this case, the direction of the implication is *downward*. Some privileges do not have any implications, such as creation (definition) of a new class. So all access privileges are split into three classes: (1) *OpUp* contains those access privileges having upward implications, (2) *OpDown* contains those access privileges having downward implications, and (3) *OpNil* contains those privileges having no implications.

Implication Rule 5 *Let $r \in OpUP$. The explicit right (s, o, r) yields the implicit authorization (s, o', r) for all entity o' above o in the entity hierarchy.* □

Implication Rule 6 *Let $r \in OpUP$. The explicit authorization $(s, o, -r)$ yields the implicit right $(s, o', -r)$ for all entity o' below o in the entity hierarchy.* □

Implication Rule 7 *Let $r \in OpDown$. The explicit authorization (s, o, r) yields the implicit right (s, o', r) for all entity o' below o in the entity hierarchy.* □

Implication Rule 8 *Let $r \in OpDown$. The explicit right $(s, o, -r)$ yields the implicit authorization $(s, o', -r)$ for all entity o' above o in the entity hierarchy.* □

Implication Rule 9 *Let $r \in OpNil$. The explicit authorization (s, o, r) generates no implicit ones.* □

Note that the *read* right on methods acts the *execute* privilege. All accesses that are performed during the execution of the method must be authorized independently. For example, if during the execution of a method invoked by a subject s, an attempt is made to update a particular property value o of an instance, the triple $(s, o, write)$ needs to be verified. Other hierarchical structures (which are orthogonal to the entity hierarchy) are inheritance, composite, and version hierarchies.

There are two approaches to privilege of instances of a subclass. In the first one, the creator of a class should have no implicit right on the instances of a subclass derived from the class. This position encourages the reuse of existing classes without diminishing privacy. However, a query whose scope of access is a class and its subclasses will only be evaluated against those classes for which the user issuing the query has a read privilege. In the second approach, the creator of a class should have implicit rights on instances of subclass. This means that a query (whose scope of access is a class and a class sub-hierarchy is rooted at the class) will be evaluated against the class and its subclasses. The first approach is used as the default while the second one is an option.

Implication Rule 10 *A read (or write) privilege on a class of a inheritance hierarchy implies read (or write) rights on all classes in the inheritance hierarchy.* □

The following rule enforces consistency between the SAB and WAB for positive and negative authorization in multiple inheritance hierarchy.

Implication Rule 11 *The ordering between the access privilege subclass-generator and other access privilege is: write>subclass-generator and subclass-generator>read-definition. So a user with the privilege write on a class receives implicitly the subclass-generator right, i.e., the user can derive subclasses from the class. A user with the privilege subclass-generator on the class C implicitly receives the read-definition right, i.e., the user can read the class definition of C.* □

Implication Rule 12 *Authorization on a composite class C implies the same right on all instances of C and on all objects that are components of the instances of C. Similarly, privilege on a composite object implies the same privilege on each component of the composite object.*□

To solve conflicts caused by the combination of strong/weak, explicit/implicit, and positive/negative authorizations through the composite hierarchy, the following order of authorizations is assumed:

$(s, o, +r) > (s, o, -r) > (s, o, \overline{+r}) > (s, o, \overline{-r}) > [s, o, +r] > [s, o, -r] > [s, o, \overline{+r}] > [s, o, \overline{-r}]$, where $\overline{r}$ indicates an implicit privilege.

There are suggestions that the composite object should be considered as a unit of authorization, i.e., giving privilege to a composite object implies the same privilege to all components of the composite object. Two types of authorization: *partial and total* are defined. The next two rules specify how to enforce *partial and total* authorizations.

Implication Rule 13 *If a user has total write (read) privileges on a component of a composite object, then (s)he has the same rights on all descendants, and partial write (read) rights on all ancestors (or top-level objects).* □

Implication Rule 14 *If a user has partial write (read) privileges on a component of a composite object, then (s)he has the same rights on all ancestors (or top-level components).* □

There are two ways to bind an object with a versioned object: *static* and *dynamic*. In static binding, the first object directly references the second object. In dynamic binding, the first object references a *generic instance* of the second object. A generic instance maintains the history of the object versions derived so far. When a generic instance receives a message, the message is forwarded to one of the versions, which has been designed as the default version. Furthermore, on the basis of the types of operation that may be allowed on versions, two types of versions: *transient* and *working* can be distinguished. A transient version may be modified or deleted by the user who has created it whereas a working version may be deleted but not updated. A new version cannot be derived from a transient version. A transient version must first be *promoted* to a working version before a new version may be derived from it.

Implication Rule 15 *An authorization on a set-of-generic-instances implies the same right on all generic instances of the class. The write privilege on a generic instances allows the user to modify the generic instance itself . It also implies authorization on the object versions described by the generic instance. The read right on a generic instance means that the user has the same right on all versions. The write privilege on the set-of-versions implies the same right on the versions described by the generic instance and also gives the right to create a new version from a working version of the instance.* □

The model does not address content-dependent, and method-based authorizations. Bertino and Weigand [13] extended the model so it could handle content-dependent access rules. They introduced two different modes for authorization administration, *centralized* and *decentralized* administrations. In the decentralized administration, every user creating an instance of the class is considered to be the owner of the instance and, therefore, can grant and revoke other users' privileges to the instance. For the centralized administration, all instances of a class are considered to have the same owner who is called the *class administrator.* Instances of the class created by any user belong to the class administrator (see [13] for detailed discussion). A main problem of the model is how to efficiently evaluate conditions associated with authorizations. In particular, the processing of the conditions could require two references to the object (one to evaluate the conditions and the second to filter data that satisfy the user query).

6.2 DAC in Behaviorally OODBS

The models presented so far take into consideration many of the characteristics of O-O data models such as inheritance hierarchy, versions, and composite objects. However, they do not apply encapsulation of the O-O model. In a behaviorally OODBSystem, all interactions are conducted through messages that are parts of object interfaces. A subject invokes a initial method m_1 which in turn calls m_2, and so on. In an access control model for behaviorally OODBS, we are concerned with how the subject gets permission for methods in the sequence of calls.

6.2.1 Faatz-Spooner Access Control Model

Faatz and Spooner [45] describe a DAC model for O-O engineering databases. In the model, objects consist of the structural and operational parts. The structural part includes instance-variables that hold the data associated with the object. The operational part contains a set of methods or procedures that can be performed on the data in the structural part of the object. Only methods defined for an object can be used to manipulate the contents of the object instance-variables. These variables are not visible outside the object. The access control model applies object interfaces (object views) in order to restrict the number of messages an object accepts from other objects. Users are allowed to interact with the objects only by calling methods defined in their interface.

Although the above approach requires no extensions to the O-O paradigm, it provides only a partial solution to the access control. A complete solution has to handle access control to object classes with hierarchical structure. It also has to work properly in th context of the inheritance.

6.2.2 Iris Access Control Model

In 1992, Ahad, Davis, Gower, Lyngback, Marynowski, and Onuegbe [3] developed a DAC model which has been implemented for Iris [143]. In the Iris data model, both instance-variables and methods are represented as functions. Instance-variables are defined as *stored functions*, and methods - as *derived functions.* Objects are encapsulated by a set of functions that users can call. The access control model is based on a concept of the function call control and the evaluation of calls.

The access entities are: *stored functions, derived functions, generic functions, guard functions*, and *proxy functions*. *Stored* and *derived* functions are instance-variables and methods, respectively. A *generic function* is the specification of a function which may have a set of associated *specific functions* that are defined for different types. When a generic function is called, a specific function based on the type of the argument object in the call is selected for execution. If a function has an associated guard function, the function can be executed only if the guard function returns *true*. *Proxy functions* provide different implementations of specific functions for different users. A function may have several associated proxy functions. When a user calls a function, the appropriate proxy is executed in place of the original function.

The access control is implemented using functions that are allowed to be called by a subject (user). An access rule is a triple (u, f, t) where u is the user name, f is a function or a set of functions, and t is the type of the argument to the function. Note that the access of u is limited by the set of functions f the user is allowed to evaluate. Functions f can be either stored or derived. Access to derived functions can be *static* or *dynamic*. If a user u has a dynamic access to the derived function f, u must have a permission to call all the underlying functions. For a static access, the user u dose not need to have call privileges[1] to the underlying functions. While creating a derived functions, the user must specify whether the access to the underlying functions is going to be static or dynamic. Note that the creator of the derived function must have call privileges to all the underlying functions. The access control model supports time-dependent and content-dependent authorizations using guard and proxy functions, respectively. Derived functions can also be used to support content-dependent authorizations including condition.

The concept of ownership is supported as well. A user who creates a function is its *owner*. The owner of a function automatically has a *call privilege* to the function. If the dynamic access is specified when the object is created, the owner can freely grant call privileges for other users. By contrast, for static access, the owner cannot grant call privilege unless he has grant privilege to underlying functions. Access to a function can be granted and revoked to/from other users by users who have grant privilege as well.

The model supports also the database administrator (DBA) concept. The system may have a DBA or a group DBA who has the privileges implied by the owners of functions. Moreover (s)he can grant call privilege to functions with static access to selected users.

6.2.3 Bertino Access Control Model

An interesting example of another access control model is given by Bertino in [14]. The model uses methods as a tool to control the access to objects. That is why it is also called the *Data-hiding* model. There are two types of methods: distinguishes between *private* and *public*. Private methods can be invoked by other methods only that are presented in the list associated with the method called *invocation scope* of the method. On the other hand, a *public* method can be called by all users and methods.

The access control can have two levels: external level, and internal level. For the external access control, a triple (s, o, m) indicates that a user s can call the method m on the object o. The internal access control is content-dependent and is implemented as part of the method. Users can have privileges to public methods only. For instance, if a user u invokes a method m_1 on o which in turn calls m_2, then m_1 has to be public and u has to have execution right on that. If m_2 is public, the entry (u, o, m_2) must exist. If m_2 is private, m_1 must belong to the invocation scope of m_2.

A user u can grant the execution right on an object if u is the owner of the object. An object may have several owners. Only the creator may grant/revoke ownership authorizations to/from other users. However, the creator may grant the creator privilege to a user u. When the creator does so, u becomes the only creator of the object. The old creator looses any privilege

[1] A user who has a call privilege on a function is authorized to evaluate the function.

on the object. Sometimes it is useful to enable a user to execute a method m without giving him/her the execute rights on all public methods that are invoked by m. The model introduces the notion of *protection mode*. If the user u grants the user u' the right to execute method m in the protection mode, then all invocations of public methods made by m are checked for authorizations against u (instead of u') when u' executes m.

The reader is directed to [14], for formal definition of the model and its detailed discussion. The main problem of the model is the lack of flexibility as content-dependent authorizations is as part of method implementations. Changes in authorizations would require a change in the specification of the methods. As pointed out in [131], the impact of inheritance hierarchy on the model needs more investigation.

7 Mandatory Access Control Models

Some databases contain sensitive or classified data. A record (tuple) may be composed with elements of different security level (or classification). The security of entity can be classified on n levels (in practice $n = 4$) and can be further subdivided into compartments by category. For example, multi-level databases store data with different security classification (or security classes). The security class is a pair (L,C) where L represents a *security level* and C denotes a *category*. The security level can be unclassified, confidential, secret, and top secret. Note that they are partially ordered by the the relation "$\geq$". The category can be the name of the application that the entity is associated with. They generally have no ordering, but may be further subdivided. There are two hierarchies for both security classification of stored data and users of the database. Users are classified according to their clearance.

Unlike the discretionary access control, the mandatory access control (MAC) makes sure that the flow of information complies with a well-defined security policy. MAC enforces that users with their clearance can only access entities on "proper" security levels. The mandatory access control (MAC) is also called the multi-level access control.

Bell and LaPadula in [10] introduced their MAC model. This is an extension of the access matrix model. The Bell-LaPadula model is based on two properties: the *simple security property* and the **-property*. According to the simple security property, a subject (or an active entity) is allowed to read information from an object (or a passive entity) if the clearance level of the subject dominates the security level of the entity. The *-property requires that a subject has write access to an object if the subject clearance level is dominated by the security level of the entity. Informally, a subject can *read-down* (simple security property) and can *write-up* (*-property). For two security classes $X = (L1,C1)$ and $Y = (L2,C2)$, X *dominates* Y (denoted by $X \geq Y$) if and only if $L1 \geq L2$ and $C1$ is compartment of $C2$ or is equal to $C2$ ($C1 \sqsubseteq C2$). X *strictly dominates* Y (denoted by $X > Y$) if and only if $L1 > L2$ and $C1 \sqsubseteq C2$. The set of possible access privileges (or access modes) in the model is determined by the combinations of these properties. The privileges are: neither observe nor alter, observe only (READ-ONLY), alter only (APPEND), observe and alter (WRITE), execute a program (EXECUTE).

A generalization of the Bell-LaPadula MAC model was suggested by Denning [32, chapter 5] and called the information flow system. It is based on a lattice of security levels. Information in an entity is allowed to flow (either directly or indirectly) only on entities with higher security levels.

The application of the Bell-LaPadula model to the protection of database systems introduces new security requirements such as *entity integrity, referential integrity,* and *polyinstantiation integrity. Entity integrity* requires that no tuple can have null values for any primary key attribute. *Referential integrity* insists that no tuple in a relation can refer to a nonexistent tuple. *Polyinstantiation* means that one record can appear (be instantiated) many times, with different security levels. In order to deal with these new security requirements, extensions of the Bell-LaPdula model have been proposed for multi-level security databases (MLS/DBS). The

relational data model has dominated much of the work on MLS/DBS [29, 36, 44, 62, 68, 71, 74, 75, 76, 90, 119, 122, 128, see for example]. MLS/DBS have been developed not only as prototypes but also as products [91, 133]. Security issues have also been investigated in other systems such as OODBS [15, 19, 20, 72, 81, 97, 98, 135], entity relationship systems [51] and knowledge based systems [137] among others. A detailed report on the recent development in database security is given in [136].

8 MAC Models in RDBS

A multi-level relational database system (ML/RDBS) supports data with different security levels (or classifications) and users with different security clearances. The granularity of a multi-level system is the smallest unit of data which has its own security level. Sometimes, data may be classified at the *attribute level* (all the data associated with a particular attribute has the same security level); at the *tuple level* (every tuple has a single security level); at the *relation level* (all the data in the relation has the same security level); at the data element level; or combination of these (we have the following order: *relation level* $<$ *tuple level* $<$ *attribute level* $<$ *element level*). A relation with these security levels is called a multi-level relation. A multi-level relation R is represented by a schema $R(A_1, C_1, \ldots, A_n, C_n)$, where each data attribute A_i has a corresponding security level attribute C_i. Multi-level security affects the data model because not all data are visible to all users (data is seen differently by users with different clearances). Now consider how *entity integrity*, *referential integrity*, and *polyinstantiation* are resolved in ML/RDBS.

For a tuple, all the primary key elements must have the same security level or be at least as low as the security levels of other elements in the tuple. Otherwise, a user with low clearance would see nulls for the primary key which violates *entity integrity*.

In a ML/RDBSystem, *referential integrity* means that a tuple of a low security level cannot reference a tuple of a high security level because the referenced tuple would appear to be nonexistent to users with a low clearance.

Polyinstantiation emerges in the form of: *polyinstantiated relations, polyinstantiated tuples,* and *polyinstantiated elements* [36]. A *polyinstantiated relation* occurs when two subjects with different views of the real world entity try to create a relation of the same name [135]. A *polyinstantiated tuple* happens when a user inserts a tuple that has the same primary key value as the existing but invisible tuple (because it has a higher security level). A *polyinstantiated element* is created if a user updates created element in a tuple which looks to them to be null. The element is not null and it contains data with a higher security level. Thus a second tuple is added to the relation with the same primary key but different security level [93].

Several mandatory access control models for RDBS have been proposed so far [35, 36, 52, 62, 69, 75, 90, 91, 96, 128]. The main difference among the models is the way they solve the integrity and polyinstantiation problems. Let us consider three such models whose security evaluation classification was aimed to be at Class A1[2] level. They are as follows.

- **SeaView**. The SeaView project has its roots in the Summer Study on Multi-level Data Management Security held by the Committee on Multi-level Data Management Security of the U.S. Air Force Studies Board [26]. The project was a three-year joint work by SRI International, Gemini Computers, and Oracle Corporation sponsored by the U.S. Air Force, Rome Laboratory. The SeaView was developed at SRI by Lunt, Denning, Schell, Shockley, Heckman, and Neumann and provided element-level labeling with Class A1 assurance for mandatory security. SeaView generates virtual multi-level relations from physical *single-level* relations. The physical relations are stored in segments managed by an underlying mandatory security kernel. SeaView uses GEMSOS Trusted Computing

[2] In the evaluation criteria defined by US Department of Defense (DOD) [38], security systems is classified to four hierarchical division (D, C, B, A). Class A1 requires a formally verified system design.

Base (TCB) as the kernel. GEMSOS enforces the mandatory security policy using a label-based mechanism. Whenever a subject attempts to retrieve a physical object, a label comparison is performed. SeaView supports element, tuple, and relation polyinstantiation. A multi-level query language called MSQL is used in SeaView to define and manipulate multi-level data. Furthermore, to deal with the polyinstantiation, MSQL provides the following functions: *highest-class*, *highest-tuple*, *most-recent-tuple*, and *most-recent*. The function *highest-class* returns the tuple with the highest security level for a potentially polyinstatiated element. The *highest-tuple* function returns the tuple with the highest tuple level from among a set of polyinstantiated tuples. The function *most-recent-tuple* retrieves the most recently updated or inserted tuple from among a set of polyinstantiated tuples. The *most-recent* function returns the tuple with the most recently updated or inserted value for a potentially polyinstantiated element. For detailed discussion of the model, the reader is directed to [90, 91][23, Section 10 of Chapter 2].

- **Lock Data Views (LDV).** The LDV is a ML/RDBSystem that was developed by a group at Secure Computing Technology Corporation (SCTC). It is designed to run on SCTC's LOgical Coprocessor Kernel (LOCK) Trusted Computing Base (TCB). LDV's security policy builds on the security policy of LOCK. Its design is based on three assured pipelines for the query, update, and meta-data management operations. It allows an application to specify classification constraints for how incoming and outgoing data are to be labeled. LDV supports the tuple polyinstantiation where the tuples are polyinstantiated based on maintenance level. The maintenance level of a tuple is a security level at which the tuple was inserted into the database. However because of classification constraints, some of data elements that make up a tuple could be stored at security level above the tuple security level. TCB enforces the constraint that the level at which data is stored is the lowest security level at which the data can be retrieved. See [128, 129] for detailed discussion.

- **ASD-Views**. The ASD-Views is a prototype developed at TRW Defense System Group. The main aim for the ASD-Views project was to achieve a suitably-sized Trusted Computing Base (TCB) that met the criteria for evaluation of class B2 and above. The main feature of the project is that the ASD allows interconnection between the mandatory and discretionary access control. Moreover, only one copy of the shared data is stored within the system. This copy is accessible to users at different security levels. The ASD allows to create a subset of rows (tuples) and columns (attributes) from only one underlying base relation. Joins, aggregate functions and arithmetic expressions are excluded and does not support polyinstantiations [52, 69].

9 MAC Models in OODBS

A number of security models for OODBS have been proposed ([15, 20, 72, 79, 80, 81, 92, 93, 97, 98, 102, 135, 140, as examples]), but the design of an O-O multi-level security model is still an active research topic and there is no dominant design. The main difference among the various models is how to assign security levels to data stored in objects.

Some proposals consider *single-level objects*. That means that for every object, a unique security level is assigned and this level applies to all the content of the object (properties and methods) [15, 72, 97, 135]. This approach is attractive for its simplicity and its compatibility with the security kernel. Its most important advantage is that the security kernel is small enough so that it can be verified. Moreover there is no need to handle the multi-level update problem [135].

However in the real world, there are situations when it is necessary to classify instance-variables of an object at different security levels. That is, the security model has to support *multi-level objects.* There has been also proposals that introduce a finer grain of classification

by assigning the security level to each instance variable of an object [80, 89, 102]. Unfortunately, these proposals require trusted enforcement mechanisms on the object layer and require a complex security kernel.

In order to maintain the security kernel compatibility of the single-level object and to overcome the difficulties of multi-level object, some researchers proposed to design the schema which handles various security constraints (*simple, content,* and *context* constraints) [72, 97, 135]. For example, if we want the GAP instance-variable of the class $STUDENT$[3] to be secret, we need to create a class $STUDENT_GAP$ with security level SECRET to be a subclass of the class $STUDENT$ (see Figure 4).

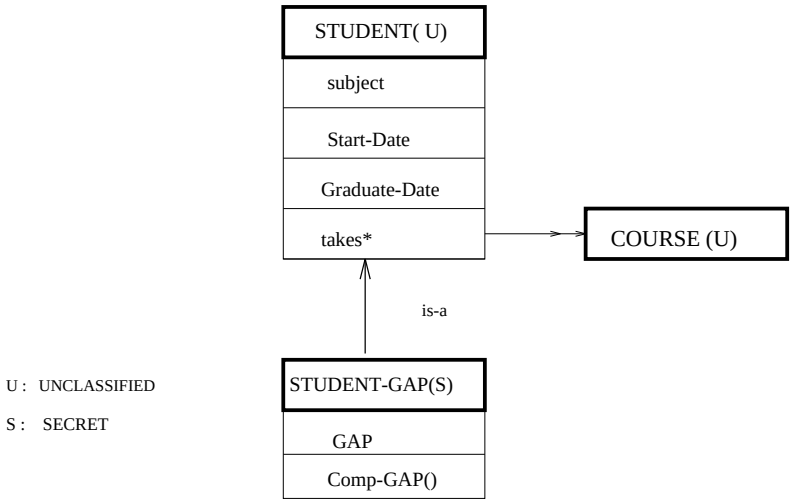

Figure 4: An example of representation of simple constraint.

The main drawback of this approach is that it requires information to be replicated at different levels, thus raising the problems of ensuring consistency data on different levels.

An alternative approach for modeling multi-level entities through single-level objects was proposed independently by Bertino and Jajodia [15], and Boulahia-Cuppens, Cuppens, Gabillon, and Yazdanian [19]. Bertino's approach is based on the use of *composite objects*. Instead of replicating low security level data in higher security level objects, a reference to the object containing the low level data inserted in the higher level object. The class $STUDENT_GAP$ has a composite property say $Student_Specification$ whose domain is the class $STUDENT$ (see Figure 5). Note that in this approach properties of a multi-level entity are stored in different objects. Thus if a user wants to see the entity, more objects must be accessed.

Boulahia-Cuppens and his group based the decomposition of a multi-level entity to single-level entities such that each level is stored in a single-level database. Dynamic links are also created between the objects of these single-level databases for properties with low security levels. For instance a multi-level object O, which is an instance of a multi-level class $STUDENT$, will be decomposed to U-O, C-O, and S-O levels. Each of them is actually a single-level object corresponding to unclassified, confidential, and secret level which is physically stored in a single-level database. In the S-O level there are pointers to the confidential properties stored in C-O level and consequently in the C-O level, there are pointers to unclassified properties. This means that if an classified user updates the unclassified properties of U-O level, this update will be automatically propagated to the instance C-O and S-O levels. Note that the values of the properties which point to low-level objects can be updated by the users cleared to access them. If it happens, then the pointer to the low-level database is broken, and the value of the object in the low-level database considers as a cover story.

[3]The specification of STUDENT is shown in Figure 6 of Appendix A

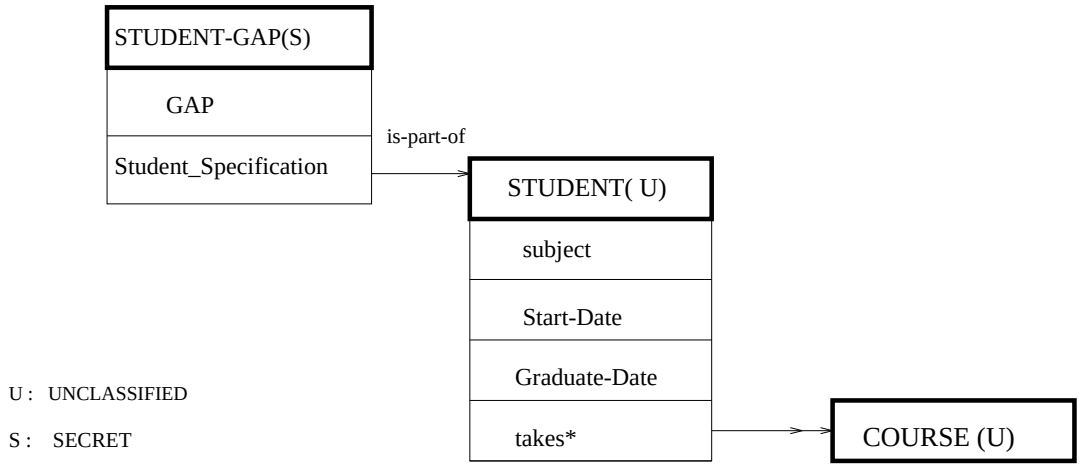

Figure 5: An example of representation of a multi-level entity through composite objects.

By employing view model mechanisms, we also propose a new design approach for modeling multi-level entities through single-level objects (see [8]). The central idea is to provide the user with a *multi-level view* derived from a single-level secure OODBSystem. Hence the database operations performed on the multi-level views are decomposed into a set of operations on the single-level objects which can be implemented with any conventional mandatory security kernel. The detailed discussion is presented in [8].

9.1 Security Models

Some of the proposed security model for OODBS are discussed briefly. For detailed discussion, the reader is directed to cited papers.

9.1.1 Jajodia-Kogan Security Model

Jajodia and Kogan [72] proposed a security model for OODBS that controls access by using the encapsulation characteristic of O-O systems. In O-O systems, communication between two objects can be done via the exchange of messages only. The model controls information flow by filtering the messages transmitted between objects. That is why it is also called the *message filter* model. It is defined in terms of *subjects* and *entities*. *Entities* are objects which may have dual roles entity/subject. They are objects as they store information. On the other hand, they are subjects as they can act by sending messages. A security level is assigned to each object at the creation time, and is fixed for the life of an object. In the model, all objects are single-level ones. Every message exchanged between objects is intercepted by the message filter. The message filter decides how to handle the message based on the security levels of the sender and receiver, and information encountered in a chain of method executions. The list of possible actions that the message filter can undertake, includes: leaving the message unaltered, blocking the message, and enforcing restriction on the execution of the method invoked by the message.

9.1.2 SORION Security Model

SORION is a security model proposed by Thurainsingham [135] to incorporate a secure access control into the ORION model [7, 82]. The SORION model is defined in terms of *subjects, entities,* and *access privileges*. A *subject* is any user of the system. Every user is assigned a security level. *Entities* in the system are *classes, objects, properties*, and *methods*. Object may be *primitive* objects or *composite* objects. The model allows the *read, write,* and *execute* access privileges. The access to an entity is controlled by checking security level of the subject (who

wants to access the entity) and the security level of the entity which is computed according to the security rules of the system. For full detail and the list of security rules, the reader is referred to [135]. The model requires all objects to be single-level ones. However, as we have already observed, real-world entities are often multi-level. The representation of multi-level entities through single-level objects are then proposed to design the schema which handles various security constraints (*simple, content,* and *context* constraints).

9.1.3 Millen-Lunt Security Model

Another security model that enforces a mandatory security policy in OODBS is proposed by Millen and Lunt [97]. Their security model is defined in terms of *subjects, entities,* and *access privileges.* A *subject* is an active entity that executes methods upon reception of messages and can also send messages. *Entities* are *classes,* and *objects. Access privileges* are *create, delete, addmessage, getvar,* and *setvar.* The model requires all objects to be single-level ones. Each request from a subject to execute a method or to write/read instance-variables is allowed only if it satisfies the security properties. A list of security properties that must be satisfied by the system is provided.

9.1.4 SODA Security Model

SODA proposed by Keefe, Tsai, and Thurainsingham [79, 81] is a security model for OODBS with multi-level entities. The model is based on the Smalltalk model [55]. The SODA model is defined in terms of *subjects*, and *entities. Entities* are objects or properties (instance-variables). Objects may be entities or subjects. Objects or properties (instance-variables) are assigned ranges of security levels. Subjects are assigned clearance levels. Every message that travels through the system carries with it a current security level and a clearance level of the subject. The current security level is the least upper bound of all security levels of information the message has read or has access to, and is adjusted whenever an object or property with higher security is accessed. Classification rules based on the current security level and clearance level of a message and on an object's or property's security level determine whether method should be allowed access to an object or property.

9.2 Security Rule Requirements for a Secure OODBSystem

Olivier, Sebastiaan and Von Solms [102, 103] gave a taxonomy for secure OODBS. They specified security issues and properties that are relevant to the modeling and implementation of a secure OODBSystem. The taxonomy identifies eight major design parameters every designer of a multi-level secure OODBSystem must consider. These eight parameters are grouped into three categories: *labeling semantics, structural labeling,* and *dynamic labeling.* The first group which consist of two parameters answers the two questions: *what is a protected entity?* and *how is an entity protected?* The *structural labeling* talks about protected entities, the way security labels are assigned to them, and specifies the restrictions the relationship among the entities places on their security labels. The last group of requirement ensures that the secrecy is not compromised by activities in an OODBSystem. For detailed discussion, the reader is directed to [102, 103]. We use this taxonomy and represent some security classification rules that must be satisfied for a secure OODBSystem. Note that the term labeling refers to the assignment of a security levels to an entity. Denote by LUB the least upper bound, GLB the greatest lower bound. The function *Level*(e) displays the security level of an entity e. The function *class(e)* returns the class of an entity e. c and o denote a class and an object, respectively. Denote by *sup(e)* the set of superclasses of an entity e, and $< x, c >$ a facet x of a class c where x can be a property or a method.

In a secure model, it must first be clarified what is exactly protected. It is possible to protect (control) the access to an entity (called *access protection* model), to hide the fact of existence of an entity (called *existence protection* model). Note that in an *existence protection* model, since the existence of an entity is hidden from subjects it is possible an entity to be re-created by subjects. The system must therefore support polyinstantaition.

9.3 Mandatory Classification Rules

In general, an OODBSystem consists of a non-homogeneous collection of entities (objects, methods, instance-variables, classes, class methods, class variables, etc.). These entities are related by relationships such as *encapsulation, instantiation, inheritance, composition, association,* and *data structure membership.*

- *Encapsulation* - the relationship that exists between an object and its facets.
- *Instantiation* - the relationship that exists between a class and its objects. Each object is an instance of a class.
- *Inheritance* - the relationship that exists between a class and its subclasses.
- *Composition* - the relationship between objects that are combined into a large object.
- *Data structure membership* - the relationship that exists between data structure (such as a list) and member of the data structure.
- *Association* - the relationship that exist for an object associated with other objects.

In this section, the influence of the structure of data on the classification of entities is considered.

9.3.1 Security Classification Rules

The first classification rule that we consider are imposed by the *encapsulation* of an object. An object encapsulates everything inside it. This implies that an encapsulated facet cannot be accessed by a subject who is not allowed to access the corresponding object.

Classification Rule 1 *(Encapsulation property) The security level of a facet of an object (class) dominates the security level of the object (class)*

$$(\forall o(\forall x \in o))[Level(< x, o >) \geq Level(o)]$$

or

$$(\forall c(\forall x \in c))[Level(< x, c >) \geq Level(c)].\Box$$

The second group of classification rules is imposed by *instantiation.*

Classification Rule 2 *(Instantiation property) The security level of an instance must dominate the security level of its class either in an existence protection model or in an access protection model, i.e.,*

$$(\forall o)[Level(o) \geq Level(class(o))].\Box$$

Classification Rule 3 *(Facet property) The security level of a facet in an instance must dominate the security level of the facet in the class and also dominate the security level of the instance itself. It is given by*

$(\forall o(\forall x \in o))[Level(< x, o >) = LUB(Level(< x, class(o) >), Level(o))]$ *(for existence protection) or*

$(\forall o(\forall x \in o))[Level(< x, o >) \geq LUB(Level(< x, class(o) >), Level(o))]$ *(for access protection).* $\Box$

A class may have several subclasses. The following properties regulate the classification subclasses and inherited facets.

Classification Rule 4 *(Inheritance property) The security level of any subclass must dominate the security level of its superclass(es)*

$$(\forall c(\forall d \in sup(c)))[Level(c) \geq Level(d)].\Box$$

Classification Rule 5 *For the access protection, if any class c inherits a facet x from a superclass $d \in sup(c)$, then*

$$Level(< x, c >) \geq LUB(Level(< x, d >), Level(c)).\Box$$

Classification Rule 6 *(Facet inheritance property) For the existence protection,*

1. *if a class c inherits a facet $< x, c >$, the security level of $< x, c >$ may only be different from its security level in the superclass when*

$$Level(< x, c >) = Level(c)$$

2. *if a facet is defined in only one superclass of a given class c (or the class c has only one superclass, or the O-O model allows single inheritance only), that facet may be inherited*

$$Level(< x, c >) \geq LUB(Level(c), Level(sup(c)))$$

3. *if a facet x is redefined in c, then the security level of $< x, c >$ must be dominated by that of every liked-named facet (facet with the same name x) in any superclass of c (whenever the security level of the facet dominates the security level of c).*

$$(\forall d \in sup(c))[Level(c) \leq Level(< x, c >) \leq LUB(Level(< x, d >), Level(c))]$$

4. *if x is inherited from a specific superclass $d' \in sup(c)$, then for every superclass $d \in sup(c)$ that has a facet x*

$$LUB(Level(< x, d' >), Level(c)) \leq Level(< x, c >)$$

and

$$Level(< x, c >) \leq LUB(Level(< x, d >), Level(c))$$

5. *the security level of an inherited facet $< x, c >$ must be dominated by the security level of all like-named facets in superclasses of c, whenever the security level of the like-named facet dominates the security level of c.* $\Box$

The last point of the rule above indicates two strategies in the existence protection that can be used to resolved the conflict caused by multiple inheritance: ensure that

1. the like-named facet with the lowest security level is always inherited, or
2. the security level of the class is an upper bound for the security levels of all concerned facets in superclasses.

If an object o is constructed from objects $o_1, o_2, \ldots, o_n$ by using array or set, then the following classification property must be satisfied.

Classification Rule 7 *(Data structure membership property)*

If Array-like structure (or homogeneous structure) is used to construct o, then all members of the array must have the same security level in the existence protection, i.e.,

$$Level(o) = Level(o_1) = Level(o_2) = \ldots = Level(o_n).$$

If o is a set-object, the security level of o is the least upper bound of the security levels of the element objects,

$$Level(o) \geq LUB(Level(o_1), Level(o_2), \ldots, Level(o_n)).\Box$$

Classification Rule 8 *(Association property) If an association R is defined between objects o_1 and o_2, then the security level of R must satisfy*

$$Level(R) \geq LUB(Level(o_1, o_2).\Box$$

9.4 Dynamic Labeling

As long as an encapsulated object moves as a unit from one to another location in the system (probably as a parameter of a message), security of information will not be compromised. For example, suppose that SALARY object has *PrintSalary* method. A subject that obtains SALARY from EMPLOYEE object will still not be able to invoke the *PrintSalary* method if it is not authorized even though the subject accessed the SALARY. Encapsulation feature of object-orientation, combined with security level labels, provides very natural mechanism of protection.

However if, in the example above, the salary was not encapsulated in a SALARY object, but rather stored as a real number, the value would have no natural protection once it leaves the EMPLOYEE object. A similar problem occurs when methods return values of instance-variables. Therefore, a secure system must impose some restrictions based on flow of authorization and flow of information that ensures that the security will not be compromised.

- *Authorization flow restrictions.* Messages act on behalf of a subject and therefore the clearance of the message depends on that of the subject. The system must specify how the clearance (or authorization) of a message is determined, and how the security level of a message is determined.

- *Information flow restrictions.* If some of the classified information contained in a message is stored in variables of the receiving object, it must be ensured that an unauthorized subject cannot now access the information in this object. The system should indicate any flow restrictions and any conventions for reclassifying.

For full detail of how the clearance of a message is determined and how they are combined, and full detail of possible strategies that can be used to control flow of information, the reader is referred to [102, 103].

10 Role-Based Access Control Models

Like military agencies, civilian government agencies and commercial firms are very much concerned with the confidentiality of the information. This includes the protection of personal data, marketing plans, product announcements, formulas, manufacturing and development techniques. Many of these organizations have even greater concern with integrity of information[24]. Integrity is particularly relevant to such applications as funds transfer, clinical medicine, environmental research, air traffic control, avionics, etc.

Each organization has unique security requirements, many of which are difficult to meet using traditional access control such as MAC and DAC. In many organizations, the end users do not "own" the system objects for which they are allowed access. In these organizations, the corporation or agency is actual "owner" of system objects as well as the programs that process them. The control is often based on the employee position in the organization hierarchy rather than their data ownership. Access control decisions are often determined by the roles individual users take on as a part of their job. The roles are usually specified by user responsibilities, and qualifications. For each user, *a role-based access control (RBAC)* policy determines the collection of roles the user is allowed to perform (undertake) within the organization. With each role, there is a associated set of functions (transactions) which is allowed to execute. The user cannot pass access permissions to other users [50]. This is the fundamental difference between RBAC and DAC.

A *role* can be thought of as a set *transactions* that a user or set of users can perform within an organization. A transaction can be considered as a *transformation procedure (TP)* (a program or portion of a program) plus a set of data items accessed by the TP. Each role has an associated set of individual members. The set of roles and their association with transactions is defined by the system administrator. Moreover, membership in a role is granted and revoked by the system administrator [50].

The usual grouping mechanism of DAC can be used to implement roles [117]. The difference between groups and roles is the same as the one between a policy and a mechanism. Two very important differences between groups and roles are as follows [118]:

1. Groups are essentially a discretionary mechanism whereas roles are non-discretionary. The ability to assign permissions to a group is usual discretionary. On the other hand, the allocation of permissions to a role, as well as determination of membership in a role, are both intended to be non-discretionary.

2. The nature of permissions allocated to a role is significantly different than the usual read, write, execute, etc.

A common challenge in designing roles is to ensure the *separation of duties.* The separation of duties requires that for particular sets of transactions, no single individual is allowed to execute all transactions with the set. The separation of duties can be either static (being built directly into the role definitions) or dynamic (with access constraints based on the previous access history of the affected entities) [1]. The static separation requirements can be implemented simply by the assignment of individuals to roles and allocation of transactions to roles. A more difficult case is the dynamic separation of duty, where the compliance with requirements can only be determined during the system operation. For example, the dynamic policy will allow an individual to take on both the initiator and authority roles for payment, with an exception that no one could authorize payments they had initiated. The system must use both the role and the user ID to check access to transactions based on *audit* records of previous accesses. The *audit* records of previous accesses introduce problems such as: how long must audit records be retained? and how are they managed in a distributed environment? Sandhu [115] discusses an automated separation of duties with the use of roles and *transactions control expressions (TCE).*

For detailed discussion of definition of RBAC, different possible approaches for role organization, and different proposed solution for separation duties the reader is directed to [5, 24, 48, 50, 49, 54, 67, 99, 100, 106, 115, 118, 120, 121, 124, 131, 132, 139, 140]

11 Implementation Strategies

The 1982 Air Force Summer Study [26] suggested two architectures for building secure multilevel database management systems (DBMS): *Trusted Subject Architecture* and *Woods Hole*

architectures. The *Woods Hole architectures* are the *Kernelized*, the *Replicated* architectures, and the *Integrity Lock*. In the *Trusted Subject Architecture* both a trusted DBMS and a trusted operating system (OS) are used, while in the *Woods Hole architectures* an untrusted DBMS is employed with an additional trusted filter. Thus, the *Trusted Subject Architecture* requires either the development from scratch a new DBMS, or the extension of the security features of an existing DBMS. In the following, we present a brief review of these architectures. For detailed discussion, the reader is directed to [23, chapter 4].

In the *Kernelized* DBMS, the multi-level database is partitioned into single-level databases which are stored separately. A trusted OS which is responsible for the physical access to data in the database is used to enforce the mandatory access control. Decomposition and recovery algorithms must be properly defined to guarantee correctness and system efficiency. The decomposition algorithm partitions a multi-level database into several single-level databases[19, 36, 73, see for example]. The Recovery algorithm is performed on single-level entities when they are retrieved to generate a multi-level entity containing only the data the user is cleared to access [77, as an example]. This architecture is used by *SeaView* [90, 91], the commercial DBMS *Oracle* [23], and in the most current research in secure databases [15, 19, 97, 135, as examples].

In the *Replicated* architecture, there is a database at each security level which contains all data whose classification are less or equal to the database security level. Each database is associated with a separate DBMS. In this architecture, we need means to replicate lower data in all database containing higher level data. There only a few research projects which are based on this architecture [28, 29, 94].

In the *Integrity Lock*, each data element is assigned with a checksum that denotes its security level which initially is in two level secret/confidential. The *Integrity Lock* consists of three main components: the *Untrusted Front End (UTFE)*, the *Trusted Front End (TFE)*, and the untrusted DBMS. The TFE authenticates and verifies a user, updates tuples, and handles all projections and creations of new data entities. The UTFE handles parsing query, formatting output, and computations. The untrusted DBMS handles searches of the database, selects requested tuples, inserts data tuples into the database, reorganizes database, and manages the storage. Integrity Lock can provide security at the record, attribute, or data element level. This architecture is vulnerable to *Trojan Horse* attacks, and inference attacks (see [34, 72]). Denning [34] proposes a new approach called a *commutative filter*. The *commutative filter* is inserted between the users and the DBMS in order to assure the elimination of the inference attack provided that the DBMS is free of *Trojan Horses* that leak data. The implementation of the integrity lock design was done in the MISTRESS database management system with the Unix operating system [58].

12 Inference Control

The word "inference" means *"forming a conclusion from premises"*. Users of any database can draw inferences from the information they have obtained from the database and prior additional information (called supplementary knowledge) they have. The inference can lead to information disclosure if the user is able to access to information they are not authorized to read. This is the *inference problem* in the database security. Thus for database systems that can contain sensitive information about individuals or companies, such as statistical databases or multi-level security databases, information flow controls may be inadequate. New solutions and mechanisms are required to deal with the inference problem.

12.1 Statistical Databases

A *statistical database* (SDB) is a database that is used for statistical queries (for example, averages, counts) on subsets of the database entities. *Inference* in an SDB allows to reveal confidential information of single entities by analysis of statistical queries [23]. A lot of work

(see [32, chapter 6], [23, chapter 5], and [2]) has been done and many solutions are proposed to make SDB secure against inference. The solutions can be classified into four categories: *conceptual, query restriction, data perturbation (or masking)*, and *output perturbation*. For detailed discussion of each of these mechanisms, the reader is directed to [2, 23, 32].

12.2 Multi-level Databases

In multi-level security database systems (MLS/DBS), the inference problem arises whenever some low-level data x can be used to derive partial or exact information about some other high-level data y. In some cases, even learning of the existence of the information may be unacceptable. An inferential link that may allow information to flow from a high security class to a low security class is called an *inference channel* or a *covert channel*. Wiseman [145] identifies four aspects of the inference problem in multi-level security databases: *addressing inference problem, relationship inference problem, aggregation problem*, and *architectonic problem*.

The *addressing inference channel* arises through the data role in addressing rather than from it being accessed. In an access protection model, labeling does not prevent the data from being addressed, only prevents from being seen directly. Then, a user can receive the unauthorized information by addressing the data using the query mechanisms and inferring its existence from the result [17, 35, 144].

The *relationship inference channel* arises whenever the system objects is classified lower than the relationships between the objects. To protect a database against the relationship inference channel, the control mechanism should allow users with low clearances to access information only if the relationship with a higher classification cannot be deducted [86, 108].

The *aggregation problem* arises when a user can form aggregates of related unclassified data that infers classified data. Here, the collection of low classified information has a high classification. So it often requires us to assign the collection a higher security level [30, 61, 88].

The *architectonic problem* arises when the structural information of the database objects (database schema) is classified lower than the database objects. Then low clearance users can use the structure information of database together with authorized information to deduce unauthorized information [126].

12.2.1 Proposed Solutions

Recently some solutions have been proposed to handle the inference problem. The inference problem can be dealt during database design [36, 87, 125, 126, 138], or during query processing [78, 134, 137].

In the first approach, security constraints during database design are handled in such a way that security violations via inference cannot occur. So many inference problems can be overcome through good design. The SeaView [93, 90], ASD-Views [52], and SWORD [112] projects are examples of this approach. Other works have also been done to provide tools which allow data designer to analyze a database schema for potential inference problems and remove those. DISSECT [53, 108], Multilevel Semantic Net [138], IAT [70], and Database Inference Controller [21] are examples. There are some non-reference formal models which can be used to verify any good design against them as well. The works of [126], [130], [70], [18], and [86] are examples.

In the second approach, the query processor is augmented with a logic-based inference engine to handle inferences during query processing. The inference engine will attempt to prevent users from the disclosure of the protected information. Some researchers argue that inferences can be the most effectively handled and thus prevented during query processing because the most users build their supplementary knowledge from responses they receive by querying the database. LDV [61] is an example of this approach.

13 Cryptographic Control

The information was protected against the disclosure by providing a suitable operating system and a suitable access control mechanism for database system. Unfortunately, there is a great deal of evidence that even sufficiently complex operating system and database system may have security holes or Trojan Horses which could allow illegal information leakage. Better protection of information can be obtained using several security measures simultaneously.

The use of cryptographic techniques to protect database systems represents another important security mechanism. Through these techniques secrecy of information is assured by making data unreadable to anyone but authorized users [32]. In general, cryptographic techniques might be used in different areas of database security: to provide user authentication, to maintain the integrity (data authenticity) and the secrecy of data, to protect information during transition and during processing from disclosure, to protect private data from accessing unauthorized user in hierarchical access control, etc.

Encryption of the data has the potential to solve both the problem of data alteration and the problem of data disclosure. However, it has some disadvantages particularly in databases such as: record searching (particularly in the case of partial-match and range queries due to scrambling of structure of data), key generating, data expansion, statistic computing, and time consuming. A lot of work has been done for application cryptography for database security [4, 31, 33, 32, 56, 59, 63, 64, 65, 107, 113, 123, 141, see for example]. For detailed discussion of cryptography applications in database security, the reader is directed to [32, 63, 107].

14 Summary and Remarks

In this paper, we have illustrated security requirements, threats to, and discretionary and mandatory security models for the protection of conventional database systems and OODBSystems. We have also discussed the types of security issues and concerns that are being expressed within OODBSystems. Then, several proposals for discretionary and mandatory security models for the protection of conventional databases and OODBSystems are presented and drawback of some of them noted. All the models illustrated offer some solutions to problem of protecting OODBS. However, each of them addresses only some of the issues, therefore leaving many question open. The following topics need further work:

- Design a more efficient mechanism to the access control on the instance-level authorization. When the number of objects grows fast, the support of protection matrix and the search at the granularity of object-instances is extremely hard and time consuming. Finding implicit authorizations will even be harder.
- Design a general discretionary access model and mechanism which includes problems such as content-dependent and context-dependent authorizations in a clean way, and constraint with positive and negative authorizations.
- Design an extended multi-level access control model which allows to use the *class, object instance,* or the *instance-variables* as the unit of authorization or labeling.
- Design a formal model for verifying the correctness of the access control model.

References

[1] M. D. Abrams. Role-Based Access Control Position Paper. In *Proceedings of the 17th National Computer Security Conference*, volume 2, page 491, Baltimore, Maryland, October 1994.

[2] N. R. Adam and J. c. Wortmann. Security-Control Methods for Statistical Databases: A Comparative Study. *ACM Computing Surveys*, 21(4):515–556, December 1989.

[3] R. Ahad, J. davis, S. Gower, P. Lyngbaek, A. Marynowski, and E. Onuegbe. Supporting Access Control in an Object-Oriented Database Language. In *Proceedings of the 3rd International Conference on Extending Database Technology, EDBT'92*, volume 580 of *Lecture Notes in Computer Science*, pages 184–200, Vienna, March 1992. Springer-Verlag.

[4] S. G. Akl and P. D. Taylor. Cryptographic Solution To A Multilevel Security Problem. *ACM Transactions on Computer Systems*, 1(3):239–248, 1983.

[5] P. E. Ammann and R. S. Sandhu. Implementing Transaction Control Expressions by Checking for Absence of Access Rights. In *8th Annual Computer Security Applications Conference*, pages 131–140. IEEE Computer Society Press, 1992.

[6] P. E. Ammann and R. S. Sandhu. The Extended Schematic Protection Model. *Journal of Computer Security*, 1(3,4), 1992.

[7] J. Banerjee, H.-T. Chou, J. F. Garza, W. Kim, D. Woelk, and N. Ballou. Data Model Issues for Object-Oriented Applications. *ACM Transaction on Office Information Systems*, 5(1):3–26, January 1987.

[8] A. Baraani-Dastjerdi, J. Pieprzyk, and R. Safavi-Naini. A Security Model for Multi-level Object-Oriented Databases Based on Views. Tr-96-03, Department of Computer Science, The University of Wollongong, Wollongong, Australia, 1996.

[9] D. E. Bell. Lattices, Policies, and Implementations. In *Proceedings 13th National Computer Security Conference*, October 1990.

[10] D.E. Bell and L.J. LaPadula. Secure Computer System: Unified Exposition and Multic Interpretation Report ESD-TR-75-306. Technical report, MITRE Corp., Bedford, Mass., March 1976.

[11] E. Bertino, S. Jojodia, and P. Samarati. Access Controls in Object-Oriented Database Systems: Some Approaches and Issues. In N. Adam and B. Bhargava, editors, *Advanced Database Concepts and Research Issues*, volume 759 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[12] E. Bertino, F. Origgi, and P. Samarati. A New Authorization Model for Object-Oriented Databases. In J. Biskup, M. Morgenstern, and C. E. Landwehr, editors, *Database Security VIII (A-60)*, pages 199–222. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1994.

[13] E. Bertino and H. Weigand. An Approach to Authorization Modeling in Object-Oriented Database Systems. In P. P. Chen and R. P. Van de Riet, editors, *Data & Knowledge Engineering*, pages 1–29. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1994.

[14] Elisa Bertino. Data Hiding and Security in Object-Oriented Databases. In F. Golshani, editor, *Proceedings of the Eight International Conference on Data Engineering*, pages 338–347. IEEE Computer Society Press, 1992.

[15] Elisa Bertino and Sushil Jajodia. Modeling Multilevel Entities Using Single Level Objects. In *Proceedings of the Deductive and Object-Oriented Databases, Third International Conference, DOOD'93*, volume 760 of *Lecture Notes in Computer Science*, pages 415–428, Phoenix, Arizona, USA, December 1993. Springer-Verlag.

[16] Elisa Bertino and Pierangela Samarati. Research Issues in Discretionary Authorizations for Object Bases. In *OOPSLA'93 Workshops on Security for Object-Oriented Systems*, pages 183–199. ACM SIGPLAN Notices, October 1993.

[17] L. J. Binns. Inference and Cover Stories. In B. M. Thuraisingham and Landwehr, editors, *Database Security VI*, pages 169–178. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1993.

[18] L. J. Binns. Inference Through Secondary Path Analysis. In B. M. Thuraisingham and Landwehr, editors, *Database Security VI*, pages 195–208. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1993.

[19] N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, and K. Yazdanian. Decomposition of Multilevel Objects in an Object-Oriented Database. In *Computer Security ESORICS 94, Third European Symposium on Research in Computer Security*, volume 875 of *Lecture Notes in Computer Science*, pages 375–402. Springer-Verlag, November 1994.

[20] N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, and K. Yazdanian. Virtual View Model to Design a Secure Object-Oriented Database. In *Proceedings of the 17th National Computer Security Conference*, volume 2, pages 66–76, Baltimore, Maryland, October 1994.

[21] L. J. Buczkowski. Database Inference Controller. In D. L. Spooner and Landwehr, editors, *Database Security III*, pages 311–322. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[22] Canadian System Security Centre Communications Security Establishment Government of Canada. *The Canadian Trusted Computer Product Evaluation Criteria (CTCPEC).* Version 3.0e, ftp.cse.dnd.ca, January 1993.

[23] S. Castano, M. Fugini, G. Martella, and P. Samarati. *Database Security.* Addison-Wesley, ACM press., 1995.

[24] D. D. Clark and D. R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proceedings IEEE Computer Society Symposium on Security and Privacy*, pages 184–194, Oakland, CA., 1987.

[25] Commission of the European Communities. Information Technology Security Evaluation Criteria. Technical report, Brussels, September 1992.

[26] Committee on Multilevel Data Management Security. Multilevel Data Management Security. Technical report, Washington, D. C.: Air Force Studies Board, National Research Council, National Academy Press, 1983. For Official Use Only.

[27] R. W. Conway, W. L. Maxwell, and H. L. Morgan. On the Implementation of Security Measures in Information Systems. *Communications of the ACM*, 15(4), April 1974.

[28] Oliver Costich. Transaction Processing Using an Untrasted Scheduler in a Multilevel Database with Replicated Architecture. In S. Jajodia and C.E. Lanwehr, editors, *Database Security V*, pages 173–191. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1992.

[29] Oliver Costich and John McDermott. A Multilevel Transaction Problem for Multilevel Secure Database Systems and its Solution for Replicated Architecture. In *Proceedings of IEEE computer Society Symposium on Research in Security and Privacy*, pages 192–203, Oakland, CA., May 1992. IEEE Computer Society Press.

[30] F. Cuppens. A Model Logic Framework to Solve Aggregation Problems. In C. E. Landwehr and S. Jajodia, editors, *Database Security V*, pages 315–333. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1992.

[31] G. I. Davida and Y. Yeh. Cryptographic Relational Algebra. In *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, pages 111–116. IEEE Computer Society Press, 1982.

[32] D. E. Denning. *Cryptography and Data Security.* Addison-Wesley Publishing Company, 1983.

[33] D. E. Denning. Cryptographic Checksums for Multilevel Database Security. In *Proceedings of the 1984 IEEE symposium on security and privacy*, pages 52–61. IEEE Computer Society Press, April 1984.

[34] D. E. Denning. Commutative Filters for Reducing Inference Threats in Multilevel Database Systems. In *Proceedings of the 1985 IEEE symposium on security and privacy*, pages 134–146. IEEE Computer Society Press, April 1985.

[35] D. E. Denning, S. G. Akl, M. Heckman, T. F. Lunt, M. Morgenstern, P. G. Neumann, and R. R. Schell. Views for Multilevel Database Security. *IEEE Transactions on Software Engineering*, SE-13(2):129–140, February 1987.

[36] D. E. Denning and T. F. Lunt. A Multilevel Relational Data Model. In *Proceedings of Symposium on Computer Security and Privacy*, pages 220–234, Oakland, CA., 1987. IEEE Computer Society Press.

[37] Dorothy E. Denning. Database Security. *Annual Reviews Inc.*, 3:1–22, 1988.

[38] Department of Defense. Department of Defense Trusted Computer System Evaluation Criteria. Technical report, Department of Defense DOD 5200.28-STD, December 1985.

[39] O. Deux. The O_2 System. *Communications of the ACM*, 34(10):34–48, October 1991.

[40] K. Dittrich. Object-Oriented Database Systems: The Notations and Issues. In *Proceedings of the First International Workshop on Object-Oriented Database Systems*, Pacific Grove, CA., September 1986. IEEE Computer Society Press.

[41] K. R. Dittrich, W. Gotthard, and P. C. Lockemann. Complex Entities for Engineering Applications. In S. Spaccapietra, editor, *Entity-Relationship Approach.* North-Holland, 1987.

[42] K. R. Dittrich, W. Gotthard, and P. C. Lockemann. DAMOKLES - The Database System for the UNIBASE Software Engineering Environment. In *IEEE Data Engineering*, volume 10(1), 1987.

[43] K. R. Dittrich, M. Hartig, and H. Pfefferle. Discretionary Access Control in Structurally Object-Oriented Database Systems. In C. E. Landwehr, editor, *Database Security II: Status and Prospects*, pages 105–121. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1989.

[44] P. A. Dwyer, G. D. Jelatis, and M. B. Thuraisingham. Multilevel Security in Database Management Systems. *Computers & Security*, 6(3):252–260, June 1987.

[45] D. B. Faatz and D. L. Spooner. Discretionary Access Control in Object-Oriented Engineering Database Systems. In S. Jajodia and C.E. Lanwehr, editors, *Database Security IV*, pages 73–83. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[46] E. B. Fernandez, R. C. Summers, and C. D. Coleman. An Authorization Model for a Shared Database. In *Proceedings of the 1975 ACM SIGMOD International Conference*. ACM Press, 1975.

[47] E. B. Fernandez, R. C. Summers, and C. Wood. *Database Security and Integrity*. Addison-Wesley Publishing Company, 1981.

[48] E. B. Fernandez, J. Wu, and M. H. Fernandez. User Group Structures in Object-Oriented Database Authorization. In J. Biskup, M. Morgenstern, and C. E. Landwehr, editors, *Database Security VIII (A-60)*, pages 57–76. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1994.

[49] D. Ferrailo, J. Cugini, and D. Kuhn. Role-Based Access Control (RBAC): Features and Motivations. In *Proceedings of The 11th Annual Computer Security Applications Conference*, pages 241–248, New Orleans, USA, December 1995. IEEE Computer Society Press.

[50] David Ferrailo and Richard Kuhn. Role-based Access Controls. In *Proceedings of the 15th National Computer Security Conference*, volume II, pages 554–563, Baltimore, Maryland, October 1992.

[51] G. E. Gajnak. Some Result from the Entity/Relationship Multilevel Secure DBMS Project. In T.F. Lunt, editor, *Discussions of topics presented at a Workshop held at the Vallombrosa, Conference and Retreat Centre, Menlo Park, CA May 1988, Research Directions in Database Security*, pages 173–190. Springer-Verlag, 1992.

[52] C. Garvey and A. Wu. ASD-Views. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy, Washington*, pages 85–95. IEEE Computer Society Press, 1988.

[53] T. D. Garvey, T. F. Lunt, X. Qian, and M. E. Stickel. Toward a Tool to Detect and Eliminate Inference Problems in the Design Of Multilevel Databases. In B. M. Thuraisingham and Landwehr, editors, *Database Security VI*, pages 149–167. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1993.

[54] L. Giuri. A New Model for Role-based Access Control. In *Proceedings of The 11th Annual Computer Security Applications Conference*, pages 249–255, New Orleans, USA, December 1995. IEEE Computer Society Press.

[55] A. Goldberg and D. Robson. *SMALLTALK-80 The Language and Implementation*. Addison-Wesley, Reading, MA, 1983.

[56] Li Gong. *Cryptographic Protocols for Distributed Systems*. PhD thesis, Jesus College, University of Cambridge, United Kindom, 1990.

[57] G. S. Graham and P. J. Denning. Protection- principles and practice. In *Proceedings of the Spring Joint Computer Conference*, volume 40, Montvale, New York, 1972. AFIPS Press.

[58] R. D. Graubart and K. J. Duffy. Design Overview for Retrofitting Integrity-lock Architecture onto a Commercial DBMS. In *Proceedings of the 1985 IEEE symposium on security and privacy*, pages 147–159. IEEE Computer Society Press, April 1985.

[59] E. Gudes, H. S. Koch, and F. A. Stahl. The Application of Cryptography for Database Security. *National Computer Conference*, pages 97–107, 1976.

[60] E. Gudes, H. Song, and E. B. Fernandez. Evaluation of Negative, Predicate, and Instance-based Authorization in Object-Oriented Databases. In S. Jajodia and C.E. Lanwehr, editors, *Database Security IV*, pages 85–98. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[61] J. T. Haigh, R. C. O'Brien, P. D. Stachour, and D. L. Toups. The LDV Approach to Database Security. In D. L. Spooner and Landwehr, editors, *Database Security III*, pages 323–339. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[62] J. T. Haigh, R. C. O'Brien, and D. J. Thomsen. The LDV Secure Relational DBMS Model. In S. Jajodia and C.E. Lanwehr, editors, *Database Security IV*, pages 265–279. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[63] Thomas Hardjono. *Applications of Cryptography for the Security of Database and Distributed Database Systems.* PhD thesis, University College, University of NSW, Sydney, Australia, 1991.

[64] L. Harn, Y.-R Chien, and T. Kiesler. An Extended Cryptographic Key Generation Scheme For Multilevel Data Security. In *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, Oakland, CA., May 1990. IEEE Computer Society Press.

[65] L. Harn and H. Y. Lin. A Cryptographic Key Generation Scheme for Multilevel Data Security. *Computers & Security*, 9(6):539–546, 1990.

[66] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in Operating Systems. *Communications of the ACM*, 19(8), August 1976.

[67] William R. Herndon. An Interpretation of Clark-Wilson for Object-Oriented DBMSs. In T. F. Keefe and Landwehr, editors, *Database Security VII*, pages 65–85. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1994.

[68] H. R. Hinke, C. Garvey, N. Jensen, J. Wilson, and A. Wu. A1 Secure DBMS Design. In *Proceedings of the 11th National Computer Security Conference*, Baltimore, Maryland, October 1988.

[69] T. Hinke. Inference Aggregation Detection in Database Management Systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA., April 1988. IEEE Computer Society Press.

[70] T. H. Hinke and H. S. Delugach. AERIE: An Inference Modeling and Detection Approach For Databases. In B. M. Thuraisingham and Landwehr, editors, *Database Security VI*, pages 179–193. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1993.

[71] D. K. Hsiao, M. J. Kohler, and S. W. Stround. Query Modifications as Means of Controlling Accesses to Multilevel Secure Databases. In S. Jajodia and C.E. Lanwehr, editors, *Database Security IV*, pages 221–240. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[72] S. Jajodia and B. Kogan. Integrating an Object-Oriented Data Model with Multilevel Security. *IEEE Computer Society Press*, pages 76–85, 1990.

[73] S. Jajodia and R. Sandhu. A Novel Decomposition of Multilevel Relations into Single-level Relations. In *IEEE Symposium on Research in Security and Privacy*, Oakland, CA., 1991.

[74] S. Jajodia and R. Sandhu. Polyinstantiation Integrity in Multilevel Relations Revisited. In S. Jajodia and C.E. Lanwehr, editors, *Database Security IV*, pages 297–307. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[75] S. Jajodia and R. Sandhu. Toward a Multilevel Relational Data Model. In J. Clifford and R. King, editors, *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data.* SIGMOD RECORD, ACM Press, 1991.

[76] Sushil Jajodia and Boris Kogan. Transaction Processing in Multilevel Secure Databases Using Replicated Architecture. In *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 360–368. IEEE Computer Society Press, 1990.

[77] Iwen E. Kang and Thomas F. Keefe. Recovery Management for Multilevel Secure Database Systems. In B. M. Thuraisingham and Landwehr, editors, *Database Security VI*, pages 225–249. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1993.

[78] T. F. Keefe, M. B. Thuraisingham, and W. T. Tsai. Secure Query Processing Strategies. *IEEE Computer Society Press*, 22(3):63–70, March 1989.

[79] T. F. Keefe and W. T. Tsai. Security Model Consistency in Secure Object-Oriented Systems. In *5th Annual Computer Security Applications Conference Tucson, Arizona*, pages 290–298. IEEE Computer Society Press, December 1989.

[80] T. F. Keefe and W. T. Tsai. Prototyping the SODA Security Model. In D. L. Spooner and C. E. Landwehr, editors, *Database Security II*, pages 211–235. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[81] T. F. Keefe, W. T. Tsai, and M. B. Thuraisingham. A Multilevel Security Model For Object-Oriented Systems. In *Proceedings of the 11th National Computer Security Conference*, pages 1–9, Baltimore, Maryland, October 1988.

[82] W. Kim, N. Ballou, J. Banerjee, H.-T. Chou, J. F. Garza, and D. Woelk. Features of The ORION Object-Oriented Database System. In *Proceedings of the 13th International VLDB Conference*, pages 319–329, 1987.

[83] Won Kim. Object-Oriented Databases: Definition and Research Directions. *IEEE Transactions on Knowledge and Data Engineering*, 2(3):327–341, September 1990.

[84] B. W. Lampson. protection. In *Proceedings of the 5th Princeton Symposium on Information Science and Systems*, Reprinted in ACM operating System Review, Vol. 8 (1). Jan. 1974, March 1971.

[85] M. M. Larrondo-Petrie, E. Guides, H. Song, and E. B. Fernandez. Security Policies in Object-Oriented Databases. In D. L. Spooner and Landwehr, editors, *Database Security III*, pages 257–269. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[86] T. Y. Lin. Inference Secure Multilevel Databases. In B. M. Thuraisingham and Landwehr, editors, *Database Security VI*, pages 317–333. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1993.

[87] T. F. Lunt. Access Control Policies For Database Systems. In C. E. Landwehr, editor, *Database Security II: Status and Prospects*, pages 41–52. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1989.

[88] T. F. Lunt. Aggregation and Inference: Facts and Fallacies. In *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, pages 102–109, Oakland, CA, May 1989. IEEE Computer Society Press.

[89] T. F. Lunt. Multilevel Security for Object-Oriented Database Systems. In D. L. Spooner and Landwehr, editors, *Database Security III*, pages 199–209. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[90] T. F. Lunt. SeaView. In T. F. Lunt, editor, *Discussions of topics presented at a Workshop held at the Vallombrosa, Conference and Retreat Centre, Menlo Park, CA May 1988, Research Directions in Database Security*, pages 2–32. Springer-Verlag, 1992.

[91] T. F. Lunt and P. K. Boucher. The SeaView Prototype: Project Summary. In *Proceedings of the 17th National Computer Security Conference*, volume 2, pages 88–102, Baltimore, Maryland, October 1994.

[92] T. F. Lunt, D. E. Denning, R. R. Schell, W. R. Shockley, and M. Heckman. The SeaView Security Model. *IEEE Transactions on Software Engineering*, June 1990.

[93] T. F. Lunt and E. B. Fernandez. Database Security. *SIGMOD RECORD, ACM Press*, 19(4):90–97, December 1990.

[94] C. McCollum and L. Notargiacomo. Distributed Concurrency Control with Optional Data Replication. In S. Jajodia and C.E. Lanwehr, editors, *Database Security V*, pages 149–173. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1992.

[95] John Mclean. The Specification and Modeling of Computer Security. *Computer*, January 1990.

[96] C. Meadows. Constructing Containers Using A Multilevel Relational Data Model. In D. L. Spooner and Landwehr, editors, *Database Security III*, pages 127–141. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[97] J. K. Millen and T. F. Lunt. Security for Object-Oriented Database Systems. In *Proceedings of IEEE computer Society Symposium on Research in Security and Privacy*, pages 260–272, Oakland, CA., May 1992. IEEE Computer Society Press.

[98] M. Morgenstern. A Security Model for Multilevel Object With Bidirectional Relationships. In S. Jajodia and C. E. Landwehr, editors, *Database Security IV: Status and Prospects*, pages 53–71. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[99] L. Notargiacomo, B. T. Blaustein, and C. D. McCollum. A Model of Integrity and Dynamic Separation of Duty for a Trusted DBMS. In T. F. Keefe and Landwehr, editors, *Database Security VII*, pages 237–246. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1994.

[100] G. Matunda Nyanchama. *Commercial Integrity, Roles and Object Orientation.* PhD thesis, Department of Computer Science, Faculty of Graduate Studies , University of Western Ontario, London, Ontario, Canada, 1994.

[101] Objectivity, Inc., Menlo Park, CA. *Objectivity Database System Overview*, 1990.

[102] M. S. Olivier and S. H. Von Solms. A Taxonomy for Secure Object-Oriented Databases. *ACM Transactions on Database Systems*, 19(1):3–46, March 1993.

[103] Martin S. Olivier. *Secure Object-Oriented Databases.* PhD thesis, Computer Science, Faculty of Natural Sciences, RAND Afrikaans University, johannesburg, South Africa, December 1991.

[104] I. Olson and A. Marshall. Computer Access Control Policy Choices. *Computers & Security*, 9(8), 1990.

[105] Ontologic, Inc., Billerica, Massachusetts. *ONTOS Reference Manual*, 1989.

[106] R. Pascale and J. R. Mc Enerney. Using THETA to Implement Access Control for Separation of Duties. In *Proceedings of the 17th National Computer Security Conference*, volume 2, pages 47–55, Baltimore, Maryland, October 1994.

[107] Charles P. Pfleeger. *Security in Computing.* Prentice-Hall, Inc., 1989.

[108] X. Qian, M. E. Stickel, P. D. Karp, T. F. Lunt, and T. D. Garvey. Detection and Elimination of Inference Channels in Multilevel Relational Database Systems. In *Proceedings 1993 IEEE Symposium on Research in Security and Privacy*, pages 196–205. IEEE Computer Society Press, 1993.

[109] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A Model of Authorization for Next-Generation Database Systems. *ACM Transactions on Database Systems*, 16(1):88–131, March 1991.

[110] F. Rabitti, D. Woelk, and W. Kim. A Model of Authorization for Object-Oriented and Semantic Databases. In *Proceedings of International Conference on Extending Database Technology*, volume 303 of *Lecture Notes in Computer Science*, pages 231–250. Springer-Verlag, March 1988.

[111] D. Russell and G. T. Gangemi. *Computer Security Basic*. O'Reill & Associates, 1991.

[112] S. Wiseman et. al. The Trouble with Secure Database. In *Proceedings of MILCOMP 89*, pages 164–170, September 1989.

[113] R. S. Sandhu. Cryptographic Implementation of A Tree Hierarchy For Access Control. *Information Processing Letter*, 27(2):95–98, 1988.

[114] R. S. Sandhu. Expressive Power of the Schematic Protection Model. *Journal of Computer Security*, 1, 1992.

[115] Ravi Sandhu. Separation of Duties in Computerized Information Systems. In S. Jajodia and C.E. Lanwehr, editors, *Database Security IV*, pages 179–189. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[116] Ravi S. Sandhu. The Typed Access Matrix Model. In *Proceedings of IEEE computer Society Symposium on Research in Security and Privacy*, pages 122–136, Oakland, CA., May 1992. IEEE Computer Society Press.

[117] Ravi S. Sandhu. Role-Based Access Control: A position Statement. In *Proceedings of the 17th National Computer Security Conference*, volume 2, page 492, Baltimore, Maryland, October 1994.

[118] Ravi S. Sandhu and H. Feinstein. A Three TIER Architecture for Role-Based Access Control. In *Proceedings of the 17th National Computer Security Conference*, volume 2, pages 34–46, Baltimore, Maryland, October 1994.

[119] Ravi S. Sandhu and Sushil Jajodia. Referential Integrity in Multilevel Security Databases. In *Proceedings of the 16th National Computer Security Conference*, pages 39–52, Baltimore, Maryland, September 1993.

[120] Ravi S. Sandhu and P. Samarati. Access Control: Principles and Practice. *IEEE Communications Magazine*, 9:40–48, September 1994.

[121] Ravinderpal S. Sandhu. Recognizing Immediacy in an N-Tree Hierarchy and Its Application to Protection Groups. *IEEE Transactions on Software Engineering*, 15(12):1518–1525, December 1989.

[122] L. M. Schlipper, J. Filsinger, and V. M. Doshi. A Multilevel Secure Database Management System Benchmark. In *Proceedings of the 15th National Computer Security Conference*, volume II, pages 399–408, Baltimore, Maryland, October 1992.

[123] J. Seberry and J. Pieprzyk. *CRYPTOGRAPHY: An Introduction to Computer Security, Advances in Computer Science Series*. Prentice Hall Inc., 1989.

[124] W. R. Shockley. Implementing The Clark/Wilson Integrity Policy Using Current Technology. In *Proceedings of the 11th National Computer Security Conference*, pages 29–37, Baltimore, Maryland, October 1988.

[125] G. Smith. Modeling Security-Relevant Data Semantics. In *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, Oakland, CA., May 1990. IEEE Computer Society Press.

[126] B. Sowerbutts and S. Cordingley. Data Base Architectonics and Inferential Security. In S. Jajodia and Landwehr, editors, *Database Security IV*, pages 309–324. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[127] D. L. Spooner. The Impact of Inheritance On Security In Object-Oriented Database Systems. In C. E. Landwehr, editor, *Database Security II*, pages 141–150. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1989.

[128] P. D. Stachour and B. Thuraisingham. Design of LDV: A Multilevel Secure Relational Database Management System. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):190–209, June 1990.

[129] Paul Stachour. LOCK Data Views. In T. F. Lunt, editor, *Discussions of topics presented at a Workshop held at the Vallombrosa, Conference and Retreat Centre, Research Directions in Database Security*, pages 63–80, Menlo Park, CA May 1988, 1992. Springer-Verlag.

[130] Tzong-An Su. Multivalued Dependency Inference in Multilevel Relational Database Systems. In D. L. Spooner and Landwehr, editors, *Database Security III*, pages 293–300. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[131] Roshan K. Thomas and Ravi S. Sandhu. Discretionary Access Control in Object-Oriented Database: Issues and Research Directions. In *Proceedings of the 16th National Computer Security Conference*, pages 63–74, Baltimore, Maryland, September 1993.

[132] Roshan K. Thomas and Ravi S. Sandhu. Conceptual Foundations for a Model of Task-based Authorization. *IEEE Computer Security Foundations Workshop*, 7:66–79, 1994.

[133] D. J. Thomsen, W. T. Tsai, and M. B. Thuraisingham. Prototyping as a Research Tool for MLS/DBMS. In C. E. Landwehr, editor, *Database Security II*, pages 63–84. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1989.

[134] M. B. Thuraisingham. Security Checking in Relational Database Management System Augmented with Inference Engines. *Computers & Security*, 6(6), December 1987.

[135] M. B. Thuraisingham. Mandatory Security in Object-Oriented Database Systems. In *Proceedings International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 203–210, New Orleans, October 1989.

[136] M. B. Thuraisingham. Recent Developments in Database Security. In *Tutorial Proceedings of the IEEE COMPSAC Conference*, Orlando, FL, September 1989.

[137] M. B. Thuraisingham. Towards the Design of a Secure Data/Knowledge Base Management System. *Data & Knowledge Engineering*, 5(1), March 1990.

[138] M. B. Thuraisingham. The Use of Conceptual Structures for Handling The Inference Problem. In C. E. Landwehr and S. Jajodia, editors, *Database Security V*, pages 333–362. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1992.

[139] T. C. Ting. A User-Role Based Data Security Approach. In C. E. Landwehr, editor, *Database Security: Status and Prospects*, pages 187–208. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1988.

[140] T. C. Ting, S. A. Demurjian, and M. Y. Hu. Requirements, Capabilities, and Functionalities of User-Role Based Security for an Object-Oriented Design Model. In S. Jajodia and C. E. Landwehr, editors, *Database Security V: Status and Prospects*, pages 275–296. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1992.

[141] Hui-Min Tsai and Chin-Chen Chang. A Cryptographic Implementation for Dynamic Access Control in a User Hierarchy. *Computers & Security*, 14:159–166, 1995.

[142] Versant Object Technologies, Inc., Menlo Park, CA. *VERSANT Technical Overview*, 1990.

[143] K. Wilkinson, P. Lyngbaek, and W. Hasan. The Iris Architecture and Implementation. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), March 1990.

[144] J. Wilson. Views as the Security Objects in a Multilevel Secure Relational Database Management System. In *Proceedings of Symposium on Computer Security and Privacy*, Oakland, CA., April 1988. IEEE Computer Society Press.

[145] S. R. Wiseman. On the Problem of Security in Databases. In D. L. Spooner and Landwehr, editors, *Database Security III*, pages 301–311. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

A An Example of a Database Schema

Here is an example of an O-O database schema, which is graphically represented in Figure 6.

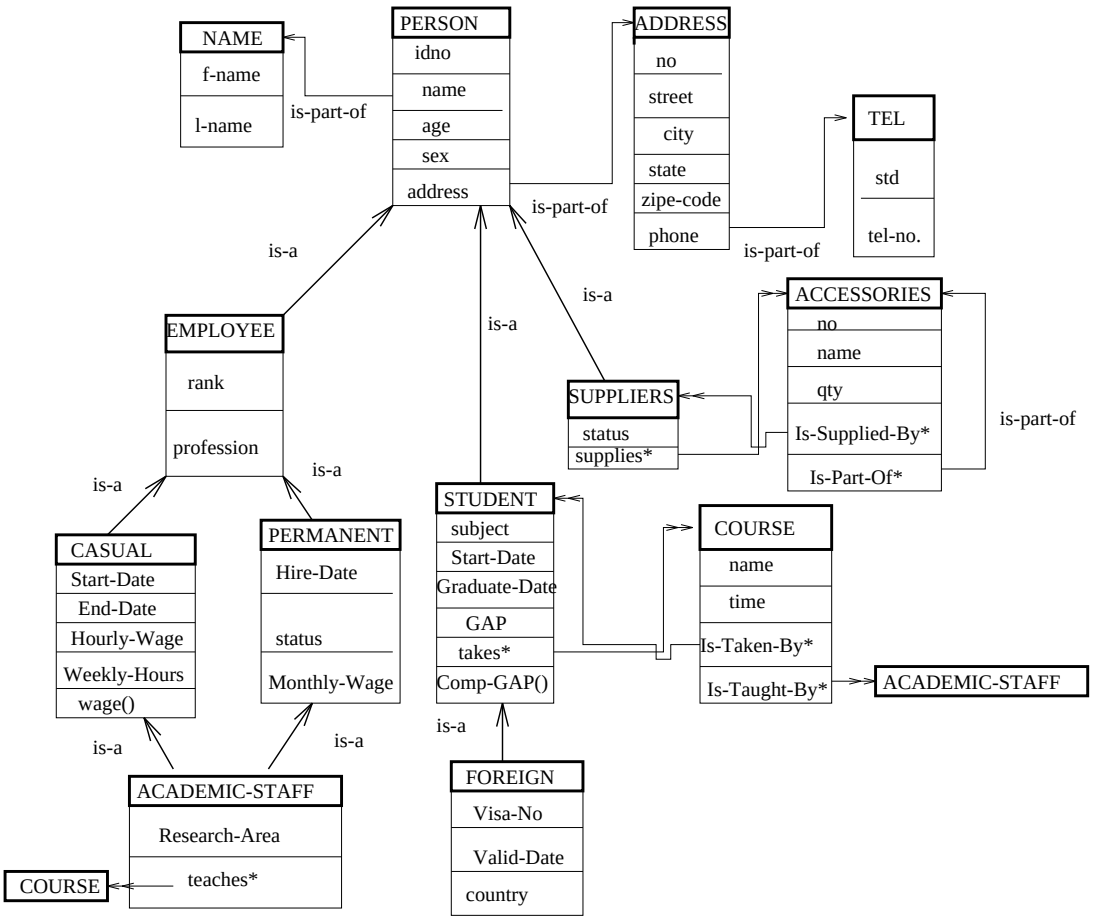

Figure 6: Example of a database schema: a portion of university database schema.

In the figure, each node table represents a class. A node is sub-divided into two parts. The bold part contains the name of the class. The second part contains properties and methods. For example, the class *STUDENT* contains the properties *subject, Start-Date, Graduate-Date, GAP, takes*, and the method *Comp-GAP()* which determines for a student the greatest average point of all courses taken by the student. The properties labeled with the symbol "*" are multi-valued properties. The node which represents a class c can be related to the node with a class c'. This relation is indicated by an arrow. The arrow can be either thin - this indicates that c' is the domain of a property of c. For example, the classes *ADDRESS* and *TEL* are the domains of properties *address* and *phone*, respectively, or bold - this indicates that c' is the subclass of c. For instance, the classes *EMPLOYEE, STUDENT,* and *SUPPLIERS* are subclasses of *PERSON*.