11

Access controls for federated database environments - taxonomy of design choices

W. Eßmayr, F. Kastner, S. Preishuber
Research Institute for Applied Knowledge Processing
Softwarepark Hagenberg
Hauptstraße 99, A-4232 Hagenberg, Austria
e-mail: {wolfgang, fritz, spr}@faw.uni-linz.ac.at

G. Pernul
Institute of Applied Computer Science and Information Systems
University of Vienna
Liebiggasse 4/3-4, A-1010 Vienna, Austria
e-mail: guenther@ifs.univie.ac.at

A M. Tjoa
Institute of Software Technology
Technical University of Vienna
Resselgasse 4, A-1040 Vienna, Austria
e-mail: tjoa@ifs.tuwien.ac.at

Abstract

We provide a short survey on the terminology of database security in database federations and give, as the main task, a taxonomy of the major design choices concerning access control in database federations. The taxonomy is organized in the categories granularity, authorization, and access control. Additionally, the impact of distribution, heterogeneity and autonomy, three characteristics of database federations, is examined for each of the design choices.

Keywords

Database security, discretionary access controls, federated database systems, object-oriented database systems

R. Posch (ed.), *Communications and Multimedia Security*

1 INTRODUCTION

Many organizations maintain information spread over several independent, possibly heterogeneous databases. Often the information is considered as a valuable and important corporate resource and the organizations have become so dependent on the proper functioning of their systems that a disruption of service or a leakage of stored information may cause outcomes ranging from inconvenience to catastrophe.

While the independent databases may have been working sufficiently during the last years, in many organizations the need to integrate existing databases into a database federation, i.e. building a *federated database system* (FDBS), becomes evident. This may be because certain applications may wish to have federated views onto the information available, covering a more global aspect. Even database federations between different organizations or companies may often be desirable in order to enforce inter-company communication. It is important to note that in FDBSs the participating *component database systems* (CDBSs) must keep their autonomy and as a result, existing local applications are still supported.

In database federations security plays an augmented role. As the need to share information and the need to maintain the autonomy of local databases joining a federation often present conflicting requirements, some of the aspects of autonomy could be instances of negotiation to sacrifice in favor of achieving a powerful federation. However, local database administrators (DBAs) would only offer their local data to the federation, if secrecy and integrity were still warranted. So, the federated security system has to be at least as secure as each of the local systems and on the other hand as transparent as possible to federated users.

The main goal of this paper is to provide a taxonomy of the major design choices needed for developing a security system for a general purpose database federation. It reports about our first experiences gained in the IRO-DB ESPRIT-III, a European project aiming at designing a FDBS which integrates existing relational and new object-oriented database systems.

The paper is organized as follows: Section 2 defines the basic terminology of database security and database federations and provides some references to related work. Section 3 and its subsections present the taxonomy of design choices, grouped in granularity, authorization and access control choices. Section 4 gives some final conclusions.

2 BASIC TERMINOLOGY AND RELATED WORK

2.1 Database Security

In general, database security is concerned with ensuring *security requirements* for information stored in databases. These requirements are:

- *Secrecy*: Information must be protected from unauthorized disclosure either by direct retrieval or indirect logical inference. Especially in federated database systems, the problems of inference and aggregation are exacerbated. By combining the data through a query against the federation, users might be able to infer additional information that should not be available even indirectly. In addition, security has to deal with the possibility that legitimate users may act as 'information channels' by passing sensitive information to unauthorized users, either intentionally or without the knowledge of the authorized user.

- *Integrity*: Information must be protected from malicious or accidental modification, including insertion of false data, contamination or destruction of data. Federations will eventually need to include multi-site constraints to deal with secrecy and integrity.
- *Availability*: Information has to be available to authorized users when they need it.

The security requirements of a system are specified by means of a *security policy* enforced by various *security mechanism*, which can be classified in the following categories:

- *Identification*: Users have to identify themselves to the computer system before being able to gain access to a database.
- *Authentication*: The identity of a user has to be verified at log-on time with authentication methods such as passwords or more sophisticated methods like badge readers, biometric recognition techniques or signature analysis.
- *Authorization*: Comprises all policies that determine how access is granted to and delegated among particular users.
- Access Control: Comprises all system mechanisms that are required to check whether a request issued by a particular user is allowed or not, and moreover, all mechanisms that are required to enforce the corresponding decision.
- *Integrity, Consistency*: A set of constraints that define the correct states of the database during database operations. Integrity and consistency policies, therefore, govern malicious or accidental modification of information.
- *Auditing*: All security-relevant operations issued by the user should be recorded for further reviews and examinations in order to test the adequacy of system controls and to recommend changes in a security policy.

A *security model* is a mechanism that implements a *security policy* of an organization. It consists of *security objects*, which are passive entities that contain and receive information and are the asset to protect (e.g. classes, objects (instances of classes), attributes...), and *security subjects*, which are responsible for changes to database states and cause information to flow among different security objects and subjects (e.g. users, transactions...).

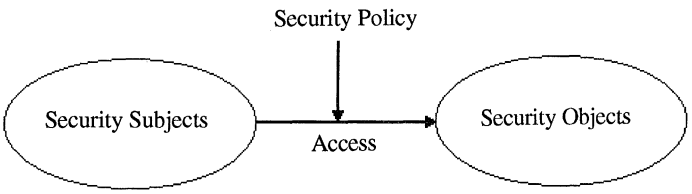

Figure 1 Basic schema of a security model.

Because of the diversity of application domains for databases, different security models and techniques have been proposed to counter various threats against security. In this paper we will not have such a broad view on security in database federations and mostly concentrate on authorization and access control. This seems to be legitimate because identification and authentication (often auditing, as well) is supported by the operating system software while integrity and consistency strongly depend on the data model or the physical design of the *database management systems* (DBMSs) used. For a more detailed study of this subject we refer to Pernul (1994). In this paper as the underlying security model *discretionary access*

control (DAC) will be assumed and the taxonomy of design choices will be within this policy. Discretionary controls specify the rules under which security subjects can, at their discretion, create and delete objects, and grant and revoke authorizations for accessing objects to other individuals. Discretionary security is not able to control the flow of information as, for example, mandatory policies do. However, it is important to note that we do not aim to develop a full functional trusted FDBS, instead the main goal of the project is to provide a certain degree of security in a general purpose FDBS.

2.2 Database Federations

Following the most common definition as developed in Sheth and Larson (1990) a *federated database system* (FDBS) is a collection of cooperating but autonomous *component database systems* (CDBSs). It represents a compromise between:

- *no integration*: users must explicitly interface with multiple autonomous databases, and
- *total integration*: autonomy of each CDBS is sacrificed so that users can access data through a single global interface but cannot directly access a CDBS as a local user.

Furthermore FDBSs can be characterized along three orthogonal dimensions, namely *distribution*, *heterogeneity* and *autonomy*. Figure 2 focuses on the autonomy dimension.

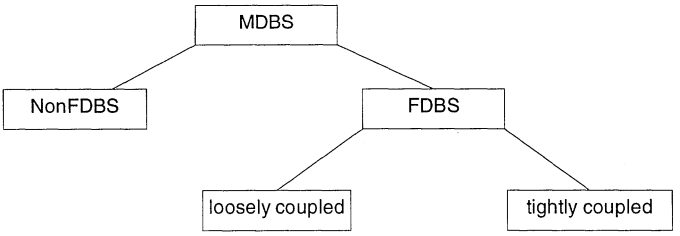

Figure 2 A taxonomy of multidatabase systems, Sheth and Larson (1990).

A *multi-database system* (MDBS) supports operations on multiple CDBSs. In the non-federated case (NonFDBS), CDBSs are *not* autonomous, whereas FDBSs preserve a certain degree of autonomy at the component sites, which participate in a federation to allow partial and controlled sharing of their data. Depending on who manages the federation, FDBSs can be categorized as *loosely coupled*, i.e. it is the user's responsibility to create and maintain the federation and no control is enforced by the federated system and its administrator(s), respectively *tightly coupled*, if the federation and its administrator(s) are responsible for creating and maintaining the federation and actively control the CDBSs. We will focus on tightly coupled federations, because they are the more interesting case from the viewpoint of security as they have a particular federation authority. Figure 3 shows a typical FDBS illustrating three possible kinds of components: CDBS 1, a centralized component database system having one central database (CDB 1), CDBS 2, a distributed database system with the distributed components CDB 2-1 to CDB 2-m, and CDBS n, a database system which itself forms a federation.

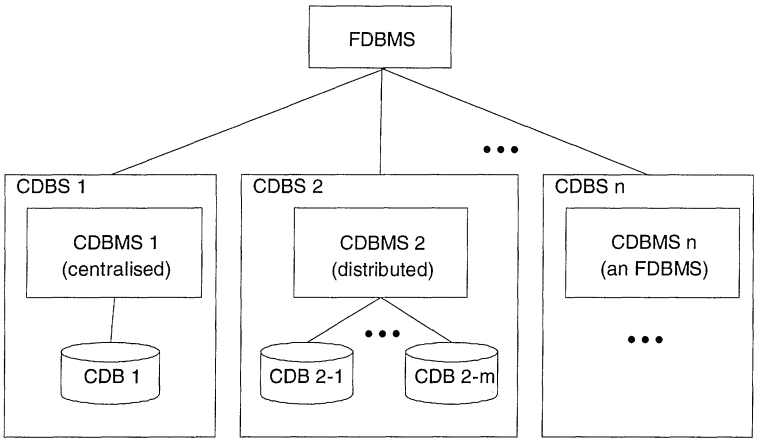

Figure 3 An FDBS and its components, Sheth and Larson (1990).

The following subsections provide a more detailed discussion of the three main characteristics of federated database systems, distribution, heterogeneity and autonomy, which together with the security mechanisms mentioned in section 2.1 form the basis of the subsequent taxonomy of design choices for authorization and access control in FDBSs.

Distribution

Distribution of data in federated systems can be regarded under several aspects:

- data may be partitioned vertically as well as horizontally (in relational terms) over component databases,
- there may be one or more computer systems involved in the federation,
- CDBSs may be collocated or geographically spread connected with local or wide area networks,
- copies of some or all of the data eventually not identically structured may be maintained.

Distributed data being available from multiple sites cause the problems of inference and aggregation to exacerbate. The FDBS needs to be able to prevent users from obtaining an unauthorized collection of data, that contravenes a more global policy on the aggregation of information, even when the CDBSs are unaware of this global policy.

The advantages of distribution can be summarized as increased availability and reliability as well as improved access time due to optimized location of data.

Heterogeneity

Many types of heterogeneity occur because of technological differences in e.g. hardware, system software (operating system) or communication systems. These problems have been addressed for several years and as in most cases standards are available can be regarded as more or less solved. Other types of heterogeneity are due to differences in the database systems. Figure 4 depicts a taxonomy of these kinds of heterogeneity.

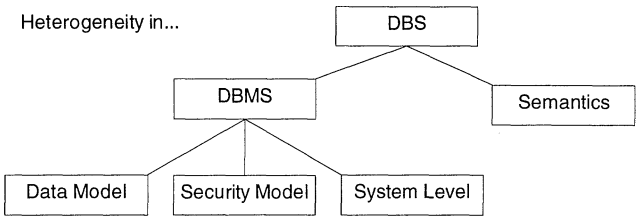

Figure 4 Taxonomy of DBS heterogeneities.

Heterogeneous data models comprise different structures (e.g. relational vs. object-oriented) and differences in constraints or query languages. Heterogeneous security models may differ mainly in granularity aspects, authorization paradigms and access control mechanisms. This is even the case if all CDBSs support discretionary controls. Heterogeneous system level support include different transaction management (concurrency control, commit protocols and recovery). Semantic heterogeneity occurs when there is a disagreement about the meaning, interpretation, or intended use of the same or related data. In general, semantic heterogeneity and heterogeneity due to differences in the DBSs are difficult to decouple.
Concerning security the federation has to deal with problems of three levels:

- No support of security mechanism at a CDBS (especially the case if newer object-oriented systems are included in the federation). The federated security system has to enforce its own policies to provide access control at the component site.
- No fundamental differences between the security models of a component and the federation site (e.g. both levels provide discretionary security). This is the point this paper intends to deal with.
- Fundamental differences between the security models of a component and the federation site, meaning, a mapping between e.g. discretionary and mandatory security has to be established.

Autonomy

There seems to be a trade-off between the federation's functionality and the degree of local autonomy at a component site, which should lead to some negotiations about how much autonomy the local systems are willing to sacrifice in favor of a powerful federation. There are various types of autonomy that a component could exhibit:

- design autonomy: implies that a CDBS has the ability to choose its own design, including the data being managed, the representation and semantic interpretation of the data, the constraints, the functionality and the implementation of the system. Therefore, local design autonomy evokes heterogeneity.
- communication autonomy: implies that a CDBS has the ability to determine whether to communicate with another component site or not, including the decision when and how to respond to a request.

- execution autonomy: implies that a CDBS treats all external operations issued from federated users in the same way as internal operations issued from local users. This includes the ability to determine the order in which to execute external operations and the ability to abort any operation that does not meet the local constraints.
- association autonomy: implies that a CDBS is able to decide whether and how much it wishes to share its functionality and resources with others, including the ability to join and leave a federation at any time and to participate in one or more federations.

The need to maintain the autonomy of a CDBS and the need to share information often present conflicting requirements.

2.3 Related Work

This subsection presents some references to related literature dealing with security in database federations. Wang and Spooner (1987) address discretionary access control in heterogeneous *distributed database management systems* (DDBMSs). They propose a solution to the problems of content-dependent access control in a heterogeneous DDBMS and present a framework for investigation of other access control issues in heterogeneous DDBMSs. Jonscher and Dittrich (1993 and 1994) discuss confidentiality for database federations enforced by discretionary access control. They present the *CHASSIS* project which aims at providing an integration framework to support the secure construction and operation of interoperable information systems. Morgenstern et al. (1992) is a panel contribution on *security issues* for FDBSs. It summarizes the viewpoints of several authors, namely M. Morgenstern, T. Lunt, B. Thuraisingham and D. Spooner. Thuraisingham (1994) deals with multilevel (mandatory) security in heterogeneous and autonomous FDBSs. She discusses issues on heterogeneity, autonomy, security policy and architecture of federated systems. Oliver (1994) proposes a model for multilevel secure federated databases with an underlying object-oriented paradigm. The model is intended for loosely coupled federations, where local classification of data item is honored by all members of the federation and each site can decide on the level of sensitivity of its data. Rusinkiewicz et al. (1991) specify *interdatabase dependencies* in multidatabase environments. They consider dependency specifications, mutual consistency requirements, and consistency restoration techniques for maintaining consistency of related data in multidatabases. Pernul (1992) gives a methodology of how to incorporate discretionary and mandatory protected CDBSs in a database federation. This work is within the AMAC access control technique.

3 TAXONOMY OF DESIGN CHOICES

This section provides a taxonomy of the major design choices for *discretionary* security in database *federations*. We assume an *object-oriented* data model at the federation level, integrating *heterogeneous* (object-oriented and relational) and *autonomous* CDBSs. The list of choices provided can never be complete in any respect, but it can at least provide a general classification schema listing relevant design choices, the possible alternatives together with resulting advantages and limitations. It addresses the subsequent questions: What are the problems in connection with providing discretionary access control in FDBSs? Which design choices have to be made? What are the consequences, the advantages respectively limitations

for each of the possible alternatives? How do distribution, heterogeneity and autonomy, the dimensions of database federations, influence the design choices?

3.1 Granularity

First of all, some choices concerning the granularity of security objects, security subjects and access types have to be made. In general, a higher granularity makes it easier to fine-tune the security system so that it better suits particular requirements, but burdens the usage and maintenance of the system. Many heterogeneities among component databases are due to differences in granularity.

Security Objects

Security objects are passive entities that contain and receive information and represent the asset to be protected. In an object-oriented data model a hierarchy of security object granules can be stated: a *database* consists of *classes*, each having an number of *attributes/methods* and concrete *objects* (note the difference between the terms *security object* and *object* as an instance of a class). Some environments distinguish between class attributes/methods and object attributes/methods (e.g. Smalltalk), others allow for explicit relationships among classes (e.g. the object database standard Atwood et al. (1993)). Note, security objects may mutate to active security subjects if a method of an object is executed. The granularity of a security object increases depending on up to which level of the hierarchy access controls are supported.

- Up to the *database level*: This is the lowest granularity possible and allows to protect the database as a whole against unauthorized access.
- Up to the *class level*: This allows to protect a class in admitting access to all or none of its objects as a whole.
- Up to the *class attribute/method level*: In this case vertical projections (in relational terms) are possible including or excluding particular attributes or methods from being accessed for all objects of the class.
- Up to the *object level*: It includes horizontal projections (in relational terms) by restricting access to a particular number of objects.
- Up to the *object attribute/method level*: This is a combination between vertical and horizontal projection (in relational terms). Access is restricted to a number of attributes and methods for a particular set of objects.

If a security model granules only up to the class level a higher granularity can be achieved by adding a kind of relational view mechanism in order to support content-dependent predicates.
A federated class may combine several local classes (or tables) and therefore have distributed attributes and methods (vertical distribution). Figure 5 gives an example of vertically distributed data, where a federated site located at Wolfsburg is responsible for producing *cars* and combines two local sites, one producing *chassis* in Chicago and one producing *engines* in Detroit.

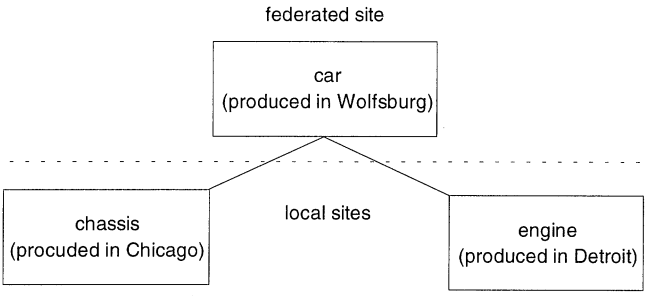

Figure 5 An example for vertical distribution.

A federated class may as well contain objects residing at different local sites (horizontal distribution). Figure 6 illustrates horizontally distributed data: the federated class *student* combines the information about all the students located at the local sites of Vienna and Graz.

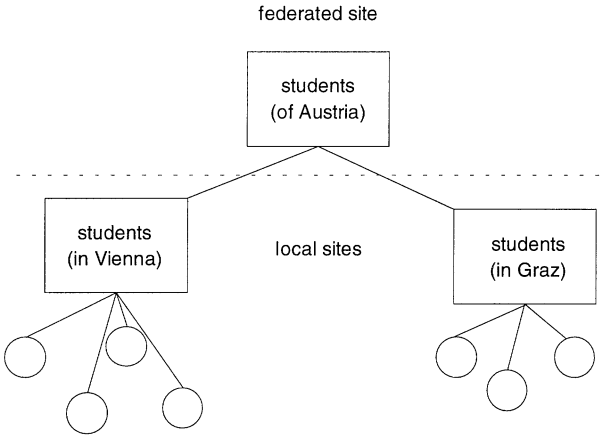

Figure 6 An example for horizontal distribution.

Concerning security objects there may be two dimensions of heterogeneity: different granularity of the security model and different data models. In object-oriented models the type of security object granules that may be supported ranges from database, class, class-attribute/method, object to object-attribute/method, while in relational models it may range from database, relation, tuple, attribute to attribute value. From the viewpoint of access control, the federation should provide the highest granularity of all component sites. As an example, if at a CDBS local security restrictions are stated on a database object basis, access

controls at the FDBS level have to support the same level of granularity at least. If not, a kind of view mechanism should be used to simulate a higher granularity at the FDBS.

Design autonomy is the reason for different granularities of security objects at component sites. If local databases joining a federation are still autonomously able to decide which of their local objects respectively which parts of these objects should be protected, they preserve design and association autonomy.

Security Subjects

Security subjects are the active entities of a security system, which are responsible for changes of a database state and cause information to flow among different security objects and security subjects. Normally, there are two kinds of security subjects:

- existing persons (*users*) operating in the system and issuing requests against a database. These could be either local users or federated users, depending on which site they are operating.
- pieces of executable machine code (*non-users*), for instance programs, functions, methods, processes, transactions etc. which are evoked by a user.

There are two possibilities to deal with non-users: On the one hand they could be regarded just as executable machine code, working on behalf of a user and therefore having the same rights as the user, who issued it. On the other hand programs, functions, methods a.s.o. could be handled as autonomous items with their own identity. The former possibility is more common, but prone to sophisticated attacks (e.g. Trojan Horses), the latter depicts a higher granularity and is therefore a more flexible mechanism, but tackling to handle and maintain.

Security subjects can be distributed over a possibly wide network of CDBSs joining the federation. There are two kinds of subjects: *local* ones which are only able to use one local database and *federated* ones, which may retrieve data from several component databases. A federated security subject (e.g. a user) has to have an identity at each local site, he/she wishes to access data.

To ensure a maximum of communication autonomy of the local sites, federated security subjects would have to be identified and authenticated at the federation as well as the component sites they try to gain access to. This might be cumbersome for federated users since they frequently have to type in passwords. Additionally, it endangers security, because identifiers and passwords have to be transmitted over the network, passwords possibly in an unencrypted form due to operating system heterogeneities. If the local sites trust the federation that it securely and correctly authenticates the identity of federated security subjects, the federation only has to deliver these identities together with the requests to the local sites. Federated users are then authenticated only once at the federation site and identification on local sites can be done automatically. The most convenient case for federated users is, when the local system is not interested in who is the federated user gaining access to the local data. Instead, it completely trusts the federation to securely identify and authenticate its federated users. Only the federation itself has to authenticate to the local site, using an artificial identity like 'FederationID'. In this case the CDBS looses communication autonomy because it is not able to decide with whom it wants to communicate.

Sets of Security Subjects

In many cases it is useful to support sets of security subjects like *groups* or *roles*. They may relieve the administrational expenditure and are apt to model the organizational structure of a

company or reflect the functional and social position of a user. Sets of security subjects may as well be security subjects themselves, meaning they analogously have rights to access security objects. If groups are nested respectively a generalization hierarchy of roles can be designed, the effect of propagated authorization has to be considered as well as policies to resolve conflicts in the case of multiple superior roles, if users may belong to multiple groups or may play more than one role at a time.

Access Types

An *access type* is the kind of action a security subject may execute over a security object. The lowest possible granularity would be *access* or *no access*, which means that a security subject may execute all possible actions over a security object or not. In many cases this is insufficient. Especially OSs dealing with files, programs and directories support a three-level granularity, namely *read*, *write* and *execute*. The typical relational access types are the well known *select*, *insert*, *update* and *delete* operations.

In general, the following five atomic actions can be mentioned: *create*, *delete*, *read*, *write* and *execute*. Most of the other kinds of access types are either identical to these atomic actions or could be combinations of them. For instance, the relational insert-operation on tables comprises the abilities to create and write tuples in that table, the update-operation is a combination of the types read and write a.s.o. Some systems provide rather special access types like write-append, write-change, write-update etc. which of course could not be expressed adequately with these atomic actions mentioned above. In any way, the exact meaning of an access type has to be clearly specified in order to avoid misinterpretations and depends strongly on the system, for which access controls have to be established.

Methods are a unique feature of the object-oriented paradigm and can be treated in two different ways: either they correspond to access types, meaning, the internal actions of the methods, e.g. what other methods they call and what attributes they access, are the responsibility of the system and no special authorization is required for that, or, they correspond to security objects demanding a special kind of access type, e.g. execute, in order to control the use of methods.

Furthermore, assumptions on propagated authorization have to be made, stating that for instance the right to create a database implies the right to create classes and objects, according to the hierarchy of security object granules.

As mentioned above there are many possibilities for semantic heterogeneity of access types of different CDBSs. Sometimes only naming differences occur (e.g. select/read), sometimes access types differ in the exact scope of atomic actions comprised by one particular access type. In DBSs especially relational ones, the relational operations are common. For FDBSs integrating relational and object-oriented CDBSs it might be more convincing to apply some kind of atomic actions allowing to resolve composed access types.

Design autonomy is the reason for heterogeneity of access types in CDBSs. If local databases joining a federation are still autonomously able to decide how their local objects should be protected, they preserve design autonomy.

3.2 Authorization

Authorization comprises all policies that determine how access is granted to and delegated among particular security subjects.

Closure Assumption

First of all a fundamental assumption describing the basic attitude of an authorization system has to be made, namely should the system be *open* or *closed*. In an open system everything is accessible unless explicitly (or implicitly due to propagated authorization) forbidden, whereas a closed system represents the inverse situation. If security is an important objective, a closed system should be used while flexibility requires open systems.

There are four possible options as shown in Figure 7: (a) a *closed* federated system with a *closed* component system, (b) a *closed* federated system with an *open* component system, (c) an *open* federated system with a *closed* component system and (d) an *open* federated system with an *open* component system.

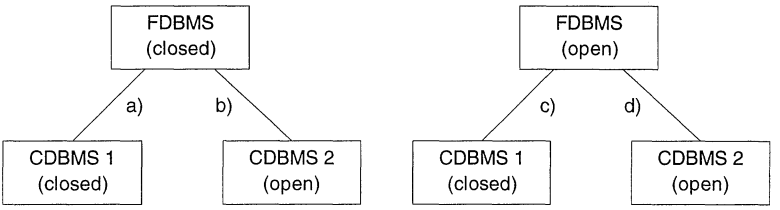

Figure 7 Closure assumption heterogeneity.

Options a) and d) depict compatible systems and are in fact not heterogeneous with respect to the closure assumption. In the case of b) and c) it depends on the degree of autonomy the component system wishes to preserve. Again there are four possibilities:

- *closed policy overrides open policy*: guarantees a maximum of security with a possible loss of component autonomy,
- *open policy overrides closed policy*: guarantees a maximum of flexibility with a possible loss of component autonomy,
- *component policy overrides federated policy*: preserves component autonomy and
- *federated policy overrides component policy*: sacrifices component autonomy.

Kinds of Authorization

A second fundamental decision concerns another basic attitude of the authorization system: Should it be *positive*, *negative* or *mixed* in the sense that positive systems only give *permissions* (often in connection with a closed policy), negative systems only *prohibitions* (frequently used with an open policy) and mixed systems allow for permissions and prohibitions, which of course requires a policy to solve conflicts. In positive systems it is burdensome to exclude one particular security subject from a particular kind of access whereas the inverse situation is true for negative systems. Mixed systems are more flexible than strictly positive or negative ones although they may be harder to maintain.

Authorization Paradigm

Decentralized authorization follows the *ownership paradigm*, saying that security subjects own those security objects they have created, i.e. they possess all possible rights to access

them. In order to make information accessible to other security subjects, access may be delegated under the discretion of the owner. Decentralized authorization is rather flexible and apt to the particular requirements of individual security subjects. Due to the principle of delegation, the problem of cascading and cyclic authorization may arise. Furthermore, decentralized authorization contradicts the situation in many enterprises, where information is owned by the whole company rather than by several individuals and the enterprise can monitor authorization only at great expense.

Centralized authorization follows the *administration paradigm*, saying that only one central authorization unit may grant and delegate access to security subjects. Authorization remains easy to survey and reflects the situation in many enterprises. Additionally there is no problem of cascading or cyclic authorization, but it is rather inflexible, since security subjects have to negotiate with the administrator if they need access to particular security objects.

In the case of database federations, a centralized approach may be preferable since security subjects are distributed over several database sites and decentralized authorization therefore will be hard to survey.

Propagated Authorization

Most of the security models imply some sort of automatic authorization, meaning that not every kind of access has to be explicitly specified or granted. If there are large numbers of security objects and/or security subjects, the administrational expenditure may be significantly reduced, especially if authorization is centralized. Authorization may be propagated for instance from groups to users, along a role-hierarchy, along a class-hierarchy, within complex objects or along the hierarchy of security object granules (e.g. from classes to their objects a.s.o.). For a comprehensive discussion of propagated (implied) authorization see Bertino et al. (1991). In any case, the side effects of propagating authorization have to be thoroughly considered. For instance, a security subject that is authorized to fully access a particular class should not have propagated the same access to subclasses of that class, since they are more specific and could have been designed by a different security subject.

3.3 Access Control

Access control comprises all system mechanisms that are required to check, whether a request issued by a particular security subject is allowed or not, and moreover, all mechanisms that are required to enforce the corresponding decision (Jonscher and Dittrich (1993)).

Access Rules

Discretionary access controls (DAC) are based on a collection of concepts, including a set of *security objects* (O), a set of *security subjects* (S), a set of *access types* (T) defining the kind of operations a security subject may execute over a security object, and, in order to represent for instance content-based access rules, a set of *predicates* (P). The tuple $\langle o,s,t,p\rangle$ is called an *access rule* and a function f, defined as $\mathrm{f:O\times S\times T\times P\rightarrow\{True,False\}}$ evaluates *True*, if subject $s\in S$ has the permission of type $t\in T$ to access object $o\in O$ within the range of predicate $p\in P$.

Most of the common database systems supporting DAC store access rules in an *access control matrix*. In its simplest form, the rows of this matrix represent security subjects, the

columns represent security objects, and the intersection of a row and a column contains the access type that the subject has authorization for with respect to the object.

security-	object 1	object 2	...
subject 1	read	execute	...
subject 2	read, write, create, delete	-	...

Figure 8 An example for an access control matrix.

This information is usually represented with one of the five following mechanisms:

- *capabilities*: security subjects possess protected identifiers (capabilities) to access security objects. Capabilities may be passed from one security subject to another, but normally may not be altered without mediation of the database administration.
- *profiles*: A list of security objects, called profile, is associated to every security subject. Creating, deleting and changing access to security objects requires many operations since multiple users' profiles must be updated.
- *access control lists* (ACLs): The access control matrix is implemented by representing the columns as lists of security subjects attached to the particular security object. If sets of security subjects or wildcards are used these lists do not have to be excessively long. Default ACLs are used for the user friendliness of the DAC mechanism.
- *protection bits*: Bits instead of ACLs are associated with security objects, which denotes an incomplete implementation of the access control matrix model, since access to security subjects cannot conveniently be on any single user basis.
- *passwords*: Each security subject possesses its own password to each security object, which then is a 'ticket' to that object. The possibly enormous amount of passwords a user could have to remember and the sharing of passwords among users outside of the system are serious problems.

In database federations, each component site maintains a set of access rules which are most likely represented heterogeneously and therefore have to be translated in a federation common form. In addition the FDBS may add access rules in order to protect federated security objects and could copy the component access rules to the federation site for access time reasons.

Predicates

Predicates are used to additionally restrict access to security objects dependent on several constraints, which can be categorized as:

- *content-dependent*: restricting access to security objects based on the content (value) of particular attributes (vertical projection in relational terms). These constraints are frequently enforced either by *view protection* (System-R-based) or by *query modification* (Ingres-based). In any way, these mechanisms are also used to restrict access to particular attributes (horizontal projections in relational terms), if the security model does not support this level of security object granularity.
- *context-dependent*: restricting access to security objects based on external facts like system time, location, history etc. which could be important in database federations, since many possible sources of external events or facts are involved.

Conflict Resolution

Conflicts occur, if two access rules determining contradicting access types between one particular security subject and one particular security object exist. This may be explicit rule conflicts because of different federation and component policies, for instance, or implicit conflicts due to propagated authorization. Examples are:

- mixed authorization (conflicting permissions and prohibitions)
- multiple security object granules (e.g. conflicting access rules for a class and an object of that class)
- multiple security subject granules (e.g. conflicting group-rights and user-rights)
- multiple superior roles (combining two roles with conflicting access rights)
- multiple roles at a time (a user may play more than one role at a time)
- multiple groups at a time (a user may belong to more than one group)

Several superior policies for conflict resolution are feasible like preferring the more restrictive rule (for security reasons), preferring the more permissive rule (for flexibility reasons), preferring the federated respectively component rule, depending on autonomy a.s.o. Furthermore access rules could be time-stamped or could have explicit priorities, allowing to make a decision which rule should be applied.

4 CONCLUSIONS

Database federations are still challenging since problems concerning distribution, heterogeneity and autonomy of component database systems joining a federation have not been fully solved yet. Many efforts have been performed on translating schemas of different data models, overcoming semantic heterogeneities or globalizing transaction management, but the interests in developing federated security systems have been very small so far.

Although a taxonomy of major design choices as presented above can never be complete in any way, this paper should contribute a general categorization schema of choices, listing the possible alternatives together with resulting advantages and limitations. It is important to note that the main characteristics of database federations, namely distribution, heterogeneity and autonomy, have a significant influence on choosing the design for a database federation security system. We concentrated on the main security mechanisms, authorization and access control, firstly because of the required briefness of the paper and secondly, we do not intend to build a full functional trusted system, instead we want to provide a certain degree of security in a general purpose database federation.

Currently, we participate in IRO-DB ESPRIT-III, a project intending to develop a federated database system achieving interoperability of pre-existing relational databases and new object-oriented databases. Our contribution to this project is on two parts. Firstly we implement the query evaluation and query delegation components of the IRO-DB query processor and secondly we are responsible for the development and implementation of the access control subsystem of IRO-DB. This paper depicts a compilation of security issues raised during the ongoing work of this project.

5 ACKNOWLEDGEMENTS

The authors would like to thank all partners of IRO-DB ESPRIT-III, a joint project of GMD-IPSI (Germany), Ibermática (Spain), Intrasoft (Greece), Intellitic, Infosys, O2 (France), GOPAS (Germany), and FAW Linz (Austria), for many interesting discussions on technical meetings.

6 REFERENCES

Atwood, T., Duhl, J., Ferran, G., Loomis, M., and Wade, D. (1993) *The Object Database Standard: ODMG-93, Release 1.1.* Morgan Kaufmann Publishers, San Francisco, California.

Bertino, E., Kim, W., Rabitti, F. and Woelk, D. (1991) A Model of Authorization for Next-Generation Database Systems. *ACM ToDS,* Vol. 16, No. 1.

Fernandez, E.B., Gudes, E. and Song, H. (1994) A Model for Evaluation and Administration of Security in Object-Oriented Databases. *IEEE Trans. on Knowl. & Data Eng.,* Vol. 6, No. 2.

Jonscher, D. and Dittrich, K.R. (1993) Access Control for Database Federations. *DBTA Workshop on Interoperablity of Database Systems and Database Applications*, Fribourg.

Jonscher, D. and Dittrich, K.R. (1994) An Approach for Building Secure Database Federations. *Proc. 20th VLDB Conference,* Santiago, Chile.

Morgenstern, M., Lunt, T.F., Thuraisingham, B. and Spooner, D.L. (1992) Security Issues in Federated DBSs: Panel Contributions. *Proc. of the Working Conference of the IFIP WG 11.3 on Database Security.*

Oliver, M.S. (1994) A Multilevel Secure Federated Database. *Proc. of the Working Conference of the IFIP WG 11.3 on Database Security.*

Pernul, G. (1992) Canonical Security Modeling for Federated Databases. *Proc. of IFIP TC2/WG 2.6 Conf. on Semantics of Interoperable Database Systems (DS'5)*, Lorne, Australia, Nov. 1992.

Pernul, G.(1994) Database Security. In: *Advances in Computers*, Vol.38, pp. 1-72. (M. C. Yovits, ed.). Academic Press.

Rusinkiewicz, M., Sheth, A.P. and Karabatis, G. (1991) Specifying Interdatabase Dependencies in a Mutlidatabase Environment. *IEEE Computer*, Dec. 1991.

Sheth, A.P. and Larson, J.A. (1990) Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, Vol.22, No.3.

Thuraisingham, B. (1994) Security Issues for Federated Database Systems. *Computers & Security*, Vol. 13, No.6.

Wang, C.Y. and Spooner, D.L. (1987) Access Control in a Heterogenous Distributed Database Management System. *Proc. Sixth Symp. on Reliability in Distributed Software and Database Systems, IEEE Computer Society Press.*