Udo Kelter

Group-Oriented Discretionary Access Controls for Distributed Structurally Object-Oriented Database Systems

Mathematik und Informatik

Informatik-Berichte
93 – 05/1990

deposit_hagen
Publikationsserver der
Universitätsbibliothek

FernUniversität in Hagen

Group-Oriented Discretionary Access Controls for Distributed Structurally Object-Oriented Database Systems

Udo Kelter
FernUniversität Hagen, Praktische Informatik V
Postfach 940, D-5800 Hagen, Germany

April 1990

Contents

Abstract

Structurally object-oriented database systems [Di86] are a new class of dedicated data storage systems which are intended to be a basis of CAD, CASE, and other design environments which shall support large development teams.

This paper presents a concept for discretionary access controls for structurally object-oriented database systems. It addresses two particular problems:

A distinguishing feature of the data

model of structurally object-oriented database systems are complex objects. Complex objects are nested and can overlap, i.e. they can share components. Arbitrary complex objects should be units of access control. Shared components cause particular problems because the objects in which they are contained might have contradicting access rights. This problem is solved by introducing certain constraints on the way in which access rights can be granted or denied.

A second major problem results from the organization of development projects which use design environments: typically, this is a hierarchy of nested groups. Our concept is group-oriented in the sense that it supports such subgroup hierarchies. Two different interpretations of a subgroup structure, termed group paradigms, are supported. Under one paradigm, a group is used to give several users the same rights, whereas under the other paradigm a group has the set of rights which corresponds to the task of the group.

Two final noteworthy features of our concept are that it employs a 4-valued logic which supports explicit denials of access and that it makes provision for distribution of the database.

Keywords: discretionary access controls, object-oriented databases, distributed databases, complex objects, shared objects, hierarchical groups, group paradigms, denial of access

1 Introduction

Design environments for CAD, CASE or similar application domains impose new requirements on their underlying data management system. Conventional database systems or file systems do not fulfill these requirements. This has led to the development of a new class of data management systems termed object-oriented or non-standard database management systems.

This paper deals with discretionary access controls for one particular type of such database systems, termed **structurally object-oriented** [Di86]. More specifically, we will mainly refer to systems which have been designed to be a basis of *software development environments*. We will use the term **object management system (OMS)** to refer to such database management systems and the term **object base** to refer to the database managed by the OMS. The main features of such OMSs will be presented in section 2.2.1.

We understand that an environment and its OMS shall support large development projects, which are organized in many working groups and roles. In such environments (as opposed to single-user environments), access controls are indispensable. OMSs will not be accepted by industry unless they provide access controls which are as powerful as those known from conventional database systems or formerly used project libraries.

This paper deals only with **discretionary access controls (DAC)**, not with mandatory access controls. DAC are means to restrict access to data granules on the basis of the identity of **subjects** and/or groups to which the subjects belong. The controls are discretionary in the sense that certain subjects ("**owners**") of an data granule can determine whether

and how other subjects can access this data granule.

DAC concepts for conventional database management or file systems, e.g. conventional access control lists or views, are not adequate for OMSs because they do not meet the novel conditions for, and requirements on, access controls in OMSs[1]:

- There is a hierarchy of nested, *overlapping complex objects.* A complex object, e.g. a document, a parse tree, a module hierarchy, or parts thereof, is the typical unit of access in software development environments, rather then a set of atomic objects which is specified by a query. Therefore, each complex object must be a granule of access control.

  Most OMSs are, in fact, oriented towards navigational access and do not have a powerful descriptive query language which could be a basis for defining nested data-dependent views[2]. The concepts presented here are specifically designed for such OMSs.

- Design environments are mostly based on workstations (with or without local disc) and servers which are connected by a local area network. *Distribution* of data in such architectures must be supported.

- The user groups are *hierarchically organized.* Such hierarchies must be supported; we call such access controls **group-oriented**. Users and user groups cooperate, rather than compete, in such environments. Therefore, access controls must support cooperative working. Nonetheless, groups can be in conflict in the sense that they correspond to different roles which must not be executed at the same time by one user.

The resulting requirements on access controls are discussed in more detail in section 2. In all, access controls for OMSs present a new challenge and require new approaches.

This paper presents a concept for access controls in OMSs which meets the requirements mentioned above. This concept has been developed for one particular OMS, namely PCTE[3] [PCTE87, PCTE88, PCTE+88]. PCTE is specifically designed to be a basis for software development environments and differs from other such OMSs in several aspects: it has external schemata, its programming interface is upwards compatible with the UNIX file system and it is transparently distributed.

However, we present our concepts on a level which abstracts from most details of the data model of PCTE (or other OMSs) because they are not very relevant here and because our concepts are actually applicable to a wide range of structurally object-oriented OMSs for CASE and other application domains. Of course, detailed features of other OMS data models may necessitate certain adaptations.

Access control concepts often address data integrity, too. Structurally object-oriented OMSs differ considerably in their data model inherent integrity constraints and their features for specifying integrity constraints. The concept presented here does not address data integrity, but can be extended in this direction for certain classes of OMSs.

The rest of this paper is organized as follows:

[1] See also [EURAC89, DiHP88, GPI89a, GrS87, Ke88, Ke89, Pe89].

[2] One main reason for this situation stems from the particular circumstances of distribution. See also sections 2.2.1 and 2.5.1.

[3] PCTE is the acronym of "A Basis for a Portable Common Tool Environment"

Section 2 introduces for several problem areas background information, definitions, basic features of our concept, and a summary of the main problems of this area.

Section 3 introduces a central notion of our concept, access right determinations.

Section 4 discusses how the group structure is to interpreted. It is shown that two different interpretations, termed group paradigms, must be supported.

Section 5 discusses the particular problems due to complex objects and sharing of components. The proposed solution is based on a consistency constraint for rights on nested data granules.

Section 6 presents the list of access modes.

Section 7 compares our approach with other proposals and surveys an extension, namely type level access controls.

2 Problem Analysis and Basic Definitions

This section discusses several problem areas which are particularly relevant for DAC in OMSs. For each area, we will (1) describe relevant aspects of the "real world situation" in OMSs or software development environments, (2) define related terminology, (3) introduce basic features of our DAC concept, and (4) summarize the most important problems. These steps are not strictly sequential, because problem analysis and design are - as usual - interleaved.

We start with defining several general notions of DAC (adapted from [DOD83, Hs87, ITS89]).

2.1 Basic Notions of DAC

A (**data**) **granule** is a passive entity which contains or receives information. Access to a granule potentially implies access to the information it contains. Normally, the term 'object' is used instead of 'granule'; however, this would be confusing here since objects are not the only granules in DAC for OMSs.

A **subject** is an active entity, e.g. a person or a device, that causes information to flow among objects or changes the system state. We assume here that the OMS is accessed by executing programs on behalf of a (human) user[4].

An **access mode**, or simply a mode, is a name for a set of OMS operations.

Access Right Determinations. Access right determinations constitute *the state of the object base with regard to access control.* This state is used to compute whether an intended access is permitted (we call this the **evaluation** of the state) and to explain the effect of operations which modify access rights.

An **access right determination** (**ARD** for short) is a quadruple (S,G,M,V), S being a subject, G a granule, M a Mode, and V being a value which indicates whether S shall be allowed to access G using operations in M. An ARD corresponds to an entry in an access control list.

[4] More precisely, we assume that the OMS has an application programming interface and that OMS operations can only be invoked by an application program (viz. a tool of a design environment). When deciding whether an intended access is allowed the OMS needs to know on behalf of which user(s) or subject(s) the access is to be performed. Thus, there must be a mechanism which conveys this information to the OMS. The details of such a mechanism are irrelevant for the concepts presented in this paper; they can vary considerably depending on the implementation of the OMS, the binding between an application and the OMS, and the process model of the underlying operating system. Such a mechanism includes normally means for the identification and authorization of human users. Examples of such mechanisms are *processes* in operating systems or *sessions* in transaction processing systems.

2.2 Data Granules

2.2.1 Structurally Object-Oriented Database Systems

This subsection introduces structurally object-oriented OMSs and complex objects in order to give some intuitive understanding of the data granules occurring in OMSs. We abstract from all details of concrete OMSs which are irrelevant for our DAC concept. Readers familiar with OMSs may skip this subsection.

Basic Concepts. Structurally object-oriented OMSs [Di86] have inherited concepts from database systems and file systems. We assume that their data model is an extension of the entity-relationship-model [Ch76]. Examples of such OMSs are Base/OPEN [BA89], CAIS-A [CAIS88], CWS/OMS [Ha&89], DAMOKLES [DAMO88]. GPI/OMS [GPI89, Pe89], PCTE+/OMS [PCTE+88], PVS [PVS87] and many others.

An object base contains **objects** and **relationships**. A relationship connects objects which play a "role" in this relationship. Objects and/or relationships have **attributes.**[5]

Complex Objects. A distinguishing feature of structurally object-oriented OMSs are **complex objects**. Complex objects allow to directly represent typical hierarchical structures, e.g. module graphs, documents, etc., as one complex object in the OMS. Figure 1 contains an example of a deeply nested complex object which might represent the data of a complete development project.

[5]Objects, relationships and attributed are typed. Typing is not relevant for access controls for object instances, but, of course, for access controls on the type level (see section 7.1).

In general, a complex object consists of a set of attributes, a set of direct components and a set of internal relationships. We will refer to the set of attributes as "**root node**". Components are again (complex) objects. Internal relationships of an object are relationships between this object and its direct and indirect components or between components of this object.

An **atomic object** is just a special case of a complex object, namely one without components or internal relationships. We will therefore simply refer to "objects" in the following.

Data Manipulation. We assume here that OMSs rely mainly on *navigation* for locating objects. Navigation in OMSs is similar to navigation in hierarchical file systems and completely different from navigation in network databases[6].

The OMS provides operations to copy, delete, versionize, lock, move, etc. arbitrary complex objects. Only the attributes of the object, viz. the root node, can be accessed directly; accessing attributes of a component object requires typically to navigate to this component object, e.g. via pathnames of objects.

Shared Objects. Two complex objects can share a component; such a component is called a **shared object**. We assume *symmetrical* sharing, that is none of the outer granules stands out from the others[7].

[6]For example, there is no concept of owner/member records [NDL87].

[7] An example for symmetrical sharing in figure 1 is the object module_hierarchy. The designers and developers can both write this object.

Asymmetrical sharing means that the way in which a shared object can be accessed depends on the outer granule from which is it accessed. Typical examples for asymmetrical sharing are library elements, e.g. the object 'library_module' in figure 1.

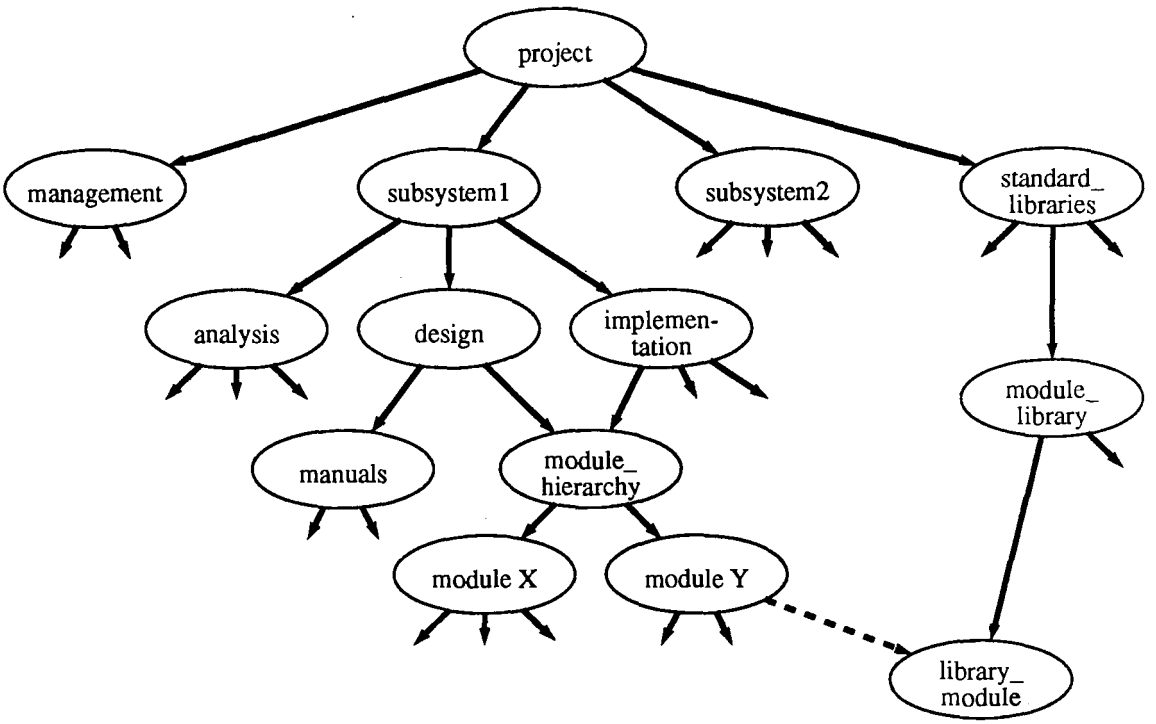

Figure 1: Nested complex objects (Ellipses represent objects, bold arrows "component-of" relationships. Attributes are not shown. Note that module_hierarchy is a *shared* object.)

The overall component-of structure in an OMS is acyclic, but not necessarily a tree.

A write right on the complex object module_hierarchy, which is granted to the designers of subsystem1, should not allow to write this library object, but only allow to write all non-library components of 'module_hierarchy'. This could be called a *second class* component-of relationship between both objects. Second-class components are not treated as part of the complex object in some operations.

A write right on the complex object module_library, which is granted to the administrator of the library, must, of course, allow to write all library modules. This could be called a *first class* component-of relationship between both objects.

First and second class components of an object should be treated differently in several operations on complex objects, e.g. copy, delete, versionize, archive, lock, move to another segment, etc. (They should therefore be distinguished by the data model.)

In PVS [PVS87] for example, each object except the root object has exactly one first class superobject and arbitrarily many second class superobjects. PRODAT [BaBK88] allows to distinguish normal components, which are first class, and library element components, which are second class. PCTE and GPI/OMS [GPI89] provides only first class component-of relationships. One could also define that a first class component automatically becomes a second class component as soon as it is shared (this seems to be proposed in [DiHP88]); this approach does, however, not adequately reflect the desired access rights in both above examples.

With regard to access rights, *second class components of an object should not be regarded as components at all*, i.e. an access right on a complex object should not imply any access rights on a second class component. A required access right on a second class component, which is almost always a read right, should be granted independently from rights on the superobject. Therefore, we will not discuss second class components and asymmetrical sharing any further here. Note, however, that the ability to distinguish first and second class components is a very useful feature of a data model.

In all, there is a hierarchy of nested, overlapping complex objects.

2.2.2 General Definitions

Objects are not the only granules in our DAC concept. Therefore, we introduce general definitions to express the nesting of granules.

We call a granule G2 an **inner granule** (or **component**) of granule G1 if it is contained in granule G1; conversely, G1 is called an **outer granule** of G2. The nesting structure of granules is, of course, acyclic[8]. For a granule G, **PARTS(G)** denotes the set of all direct and indirect inner granules of G, including G itself.

A granule is called **shared** if it is inner granule of two outer granules which are not inner/outer granule of each other.

2.2.3 Basic Features of Our DAC Concept

The following are granules[9]:

- complex objects
- root nodes of objects
- relationships

The set of attributes of a relationship needs not to be a granule because the access modes for the "pure" relationship and the attributes of a relationship are disjoint (see section 6).

Although it would be useful in some cases if single attributes were granules, too, this would introduce too much processing and storage overhead[10]. If different attributes shall have different ARDs then other data structures must be used: these attributes must, e.g., be allocated at suitably placed component objects, which can have individual ARDs.

The "inner granule" structure is as follows: an object contains its root node, its component objects and its inner relationships. Root nodes and relationships do not have inner granules.

2.2.4 Main Problems

Nesting and Intersection of Objects. An access right for a complex object must be valid for all its components. For example, a read right for a complex object must imply that the whole object, including all components, can actually be read. This means that an access right for a complex object implies **implicit access rights** for all components.

An access right for a complex object should remain valid even if the set of components of the complex object changes.

The most important problem for access controls in OMSs is that complex objects can overlap. Note that a shared object is contained in two or more complex objects which are not component of each other. Thus, several implicit access rights, which might be *contradicting*, could apply for a shared object.

Ownership. Complex objects cannot always have an owner: It must be possible that components of a complex object are exclusively owned by different subjects. Thus it must be possible that a complex object does not have an owner. However, one must be able to reach again a state in

[8] In some OMSs, the whole object base is also a granule, in fact the "outermost" granule.

[9] This set of granules refers to our assumptions about the data model (see section 2.2.1), which abstracted from many details of concrete data models. A concrete data model may have additional features which lead to further granules. For example, if the OMS supports versioning of objects then single versions and version sets should be granules.

[10] Conceptually, it would be no problem to have single attributes as granules.

which one can modify access rights.

Conversely, an owner of a complex object very often wants to enforce to remain owner of the complete complex object, even when other subjects are allowed to change the structure of the complex object, e.g. to insert new components[11]. This problem occurs also in UNIX-like directory structures: assume user A owns a directory and user B inserts a new file into this directory. Then user A is no longer owner of the directory and all its "components".

2.3 Subjects

We make certain assumptions about how projects which use an OMS based environment are organized into groups. These assumptions and the resulting problems will now be explained.

2.3.1 Nested Working Groups

Working groups in a project are formed according to a repeated division of the overall task of the project into smaller tasks. Work may be divided

- *quantitatively*, e.g. a system is divided into subsystems which are developed independently, or
- *qualitatively*, e.g. according to usual roles in a project (analyst, designer, programmer, manager, technical writer etc.).

An example of a nested group structure is given in figure 2. This example has a tree structure. In general, there can be a partial order of groups. In our example, the design/review groups and quality_assurance might have a common subgroup, or the implementers of all subsystems, etc.

[11] In fact, similar examples can be found for other access modes.

We assume that the subgroups of a group contribute to the overall task of this group. Note that groups are entities of their own right, they are not identified by their current set of members.

2.3.2 General Definitions

Users and **groups** are entities managed by the OMS. They can be subjects in ARDs. The set of groups has an acyclic, transitive "subgroup-of" structure and users or programs are "member-of" groups.

For a subject S, let **MEMBERS(S)** denote the set {S} if S is a user or program and the set of users or programs which are member of S or any of the transitive subgroups of S if S is a group.

Note that there are now two interpretations of the term 'group': a "real world group" is a set of people, a "technical group in the OMS" is a technical feature of the access controls in the OMS. We assume that there are suitable means (identification and authentication procedures) which enforce that a technical group in the OMS and a real world group correctly relate to each other. The "technical" definition applies in the context of all discussions about DAC within the OMS.

The same remarks apply to the terms 'subgroup' and 'user'.

2.3.3 Basic Features of Our DAC Concept

Any group-oriented DAC must offer the following general basic features (which will be assumed in the rest of this paper):

- Groups can be directly represented in the OMS, that is groups are a technical feature of the DAC of the OMS.
- Groups can be the subject of access right determinations.

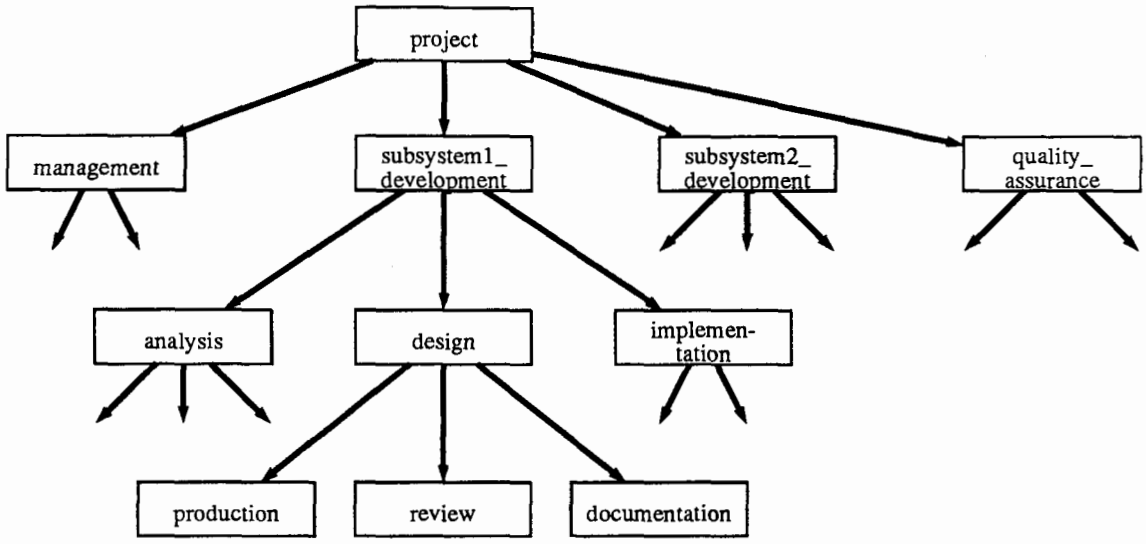

Figure 2: A group structure

- The structure of subgroups is explicitly represented in the OMS and exploited in the evaluation of access right determinations.
- Arbitrarily many access right determinations (with different subjects) for one object are possible.
- A user can be a member of arbitrarily many groups.

In addition to the above general basic features, our DAC concept has the following additional features:

There is an additional type of subjects: programs. A **program** is an object with an attribute of type string or long field which is executable.

There is one predefined group WORLD. The "subgroup of" structure between groups is acyclic and has one root, the group WORLD.

A user or program can be a member of several groups[12]. Each user is member in at least one group[13]. One can enforce pure "user groups", i.e. groups which can have only users as member and only user groups as subgroups. The same holds for pure "program groups"[14].

2.3.4 Main Problems

Activation of Groups. A very important question is whether a user should be able to activate[15] *several groups* (in which she/he is a member) *at the same time* and to exploit their rights:

- If this is possible then several problems arise; the most important one is: Different groups can correspond to different roles which may be mutually exclusive, for example the producers and reviewers of a document. Generally, activating more than one group contradicts directly the least privilege

[12] All subjects, "subgroup of" relationships, and "member of" relationships are represented in the OMS as objects or relationships of certain predefined type.

[13] As a consequence, each user is direct or indirect member of the group WORLD.

[14] These restrictions on membership do, however, not affect our concepts and are therefore not discussed any further.

[15] We assume that the rights of a group can only be exploited if this group has been "activated", e.g. through an explicit command of a user.

principle, and is therefore unacceptable in security-critical environments. As a consequence of these problems, it is often argued that the activation of more than one group should not be possible at all.

- If only one group can be activated at a time then a technical group must be (explicitly) given *all* rights required by this group and it is not possible to factor out rights which are common to several groups. This causes an *unacceptable overhead* in terms of maintenance effort and storage overhead.

We conclude that it must not be allowed to activate *arbitrary* groups at the same time, but that means are required which allow a *controlled* activation of several groups.

Combination of ARD Values. Activated groups can have different values in their ARDs for the same object and mode. Thus, a rule must be developed about how to combine these values.

2.4 Access Modes

There are only minor problems for DAC in OMSs due to access modes: due to the complexity of the data model, the number of access modes must be higher than in other systems. The full list of modes in our DAC concept will be presented in section 6.

2.4.1 General Definitions

Each granule G is assumed to have one particular type **TYPE(G)**. The set of generic OMS operations applicable to a granule depends on its type[16]. For example, there are different operations for objects and relationships. Let **OPS(T)** denote the set of operations applicable to granules of type T.

An **access mode**, or simply a **mode**, is a name for a set of OMS operations. More precisely, an access mode M associates with each type T a set of operations **M(T)**, which is a subset of OPS(T). For example, mode 'read' associates different operations with complex objects and with root nodes (see section 6).

2.5 Distribution

Design environments are typically based on workstations and servers which are connected by a local area network. We assume that the object base is distributed over the network. This requires that the object base is partitioned into several **segments** which can be independently stored, e.g. in a file or a "raw volume". Moreover we assume that certain types of volumes, e.g. tapes, floppy discs or even local hard discs, can be temporarily dismounted.

Finally, we assume that a complex object can be distributed over several sites, that is the root node and components can reside on different sites.

2.5.1 Main Problems

It can happen in distributed systems that single sites are non-operational (e.g. switched off) or unreachable (e.g. due to a network failure) or that a volume which contains a segment is dismounted. A very important design goal is therefore to remain *resilient against the unavailability of segments.* It should be possible to per-

[16] If an OMS operation involves several granules which are not inner/outer granule of each other then we regard each granule to be accessed separately. For example, the creation of a relationship between objects A and B implies an access to both A and B. In such cases, suitable access rights must be available on all involved granules to perform this operation.

form sensible work on a site with the segments reachable at this site[17]. Dependencies on data stored in other segments must be strictly avoided. (Communication delays are another reason for this design policy.) As a consequence, objects and their ARDs must be stored in the same segment.

The above remarks and our assumption that a complex object can have components on different sites imply that the access controls must be designed in such a way that it is possible to decide whether an intended access to an object is allowed by using only ARDs of this object, and without using ARDs of superobjects or subobjects (which might be stored in a different segment).

3 Access Right Determinations

ARDs, which have already been introduced in section 2.1, are a central notion of our DAC concept. This section presents a refined definition and related design decisions.

Access Units. With the above definitions of granules, subjects and modes, we are now able to formally specify the notion "access" and, more importantly, sets of accesses.

Let s be a user or program, let g be a granule, and let o be an OMS operation in OPS(TYPE(g)). Then **(s,g,o)** denotes an **elementary access**, i.e. granule g is accessed through operation o by, or on behalf of, subject s.

Let S be an arbitrary subject, G be an arbitrary granule stored in the OMS and M be an access mode as defined by the DAC mechanism of the OMS (the set of access modes is normally static, but the following does not depend on this assumption). Then the **access unit (S,G,M)** denotes the following set of elementary accesses:

$$\{ (s,g,o) \mid s \in \text{MEMBERS}(S), \; g \in \text{PARTS}(G), \; o \in \text{M}(\text{TYPE}(g))\}$$

Access Right Determinations. Access right determinations constitute *the state of the object base with regard to access control.* The state is defined to be a function **access** which maps the set of current access units onto a set of access **values**. In other words, for each triple (S,G,M), *exactly one* ARD (S,G,M,V) is valid in the OMS.

Two possible values and their meanings are:

access(S,G,M) = '+': the accesses of this unit are allowed.

access(S,G,M) = '-': the accesses of this unit are not allowed.

Further values will be defined below.

We use the notation **access(S,G,M) := V** to express that the state is changed such that access(S,G,M) = V holds after the change.

Ownership. Modification of ARDs for a granule G (i.e. access(S,G,M) := V) is also considered to be an operation on G; it is covered by one specific access mode called **'control'**. Subjects which have permission

[17] This requirement is the main reason why a distributed OMS which allows to freely move objects between segments (e.g. PCTE) does not have relational query facilities which would allow to treat all instances of an object type as a base relation: Instances of the object type can normally be created on, or moved to, any workstation. Consequently, a query must be executed on all (!) workstations and cannot terminate if only one workstation or volume is unavailable. In medium to large environments with 10 - 100 or even more sites with local disc, this is untolerable.

to modify ARDs for a granule G are also called **owners** of G.

We assume (like PCTE) the "laissez-faire" approach to access permissions (see [Do&85]), i.e. we allow several equal owners of a granule.[18]

4 Subjects and the Activation of Groups

This section gives only a short description of the features of our concept which are related to subjects. A more detailed description can be found in [Ke90].

Group Paradigms. A distinguishing feature of group-oriented DAC is that the subgroup structure is explicitly maintained and exploited. This, however, leads to the question: *what is the semantics of the subgroup structure?* A **group paradigm** is a set of assumptions and rules about why and how, given a real world group structure, technical groups and subgroups in the OMS are formed, which rights are given to them, and how these rights are combined (s. [Do&85]). Our concept supports two paradigms:

Under the **rights package paradigm**, a group corresponds to a set (a "package") of rights which shall be given to several users. The main reason for supporting the rights package paradigm is that it allows to *efficiently manage access rights.* A subgroup of a group G has less members, which have potentially more common rights than the whole group. Thus, *subgroups have more rights than supergroups.* This is to be achieved by letting the subgroups automatically and *implicitly* inherit the rights of their supergroups.

In order to exploit the inheritance of rights of supergroups (and in view of the goal to efficiently manage access rights) rights should be granted according to the **delta rule**: a group is given the rights needed by its members *except* those rights already granted to one of its supergroups.

Under the **task paradigm**, a group corresponds to a task which it shall solve. The group is given the rights required to solve this task. Only users which can act in the name of the group, and which we call **administrators**, are members of the group under the task paradigm. A subgroup has *less* rights than its supergroups since it deals only with a subtask.

Realization of the Rights Package Paradigm. Accesses to the object base are performed by executed programs, i.e. **processes**[19]. The following subjects are **active** for a process:

- the user on behalf of whom the process runs;
- *one explicitly activated group in which this user is a member*, and all direct and indirect supergroups of this group;
- all groups in which the executed program is member, and all direct and indirect supergroups of these groups.

Informally, the rights of all active groups are "added". In the simplest case, the process can perform an access if at least one subject is allowed to do this. A precise definition how the rights of the active groups are combined is given below.

[18] In our concept, one can impose a total order on the owners of a granule through ownership of outer granules. This may require to introduce dummy outer granules. Details will be explained in section 5.

[19] Processes are not necessarily understood as operating system processes here (see also footnote 4).

Realization of the Task Paradigm. There are several ways to realize the task paradigm. Due to space considerations, we will present only one of them (a second one can be found in [Ke90]):

- to each group G, a subset **A(G)** of **administrators of G** is associated. This set of users is not a technical group of its own right. Administrators, being members of G, inherit rights from the supergroups of G.
- In addition, administrators "inherit upwards" from all subgroups of G (but not from the users which are members of the subgroups). In other words, whenever a subgroup S of G is subject of an access rights determination then members of A(G) can exploit this right.

This solution does not cause much overhead: There are straightforward ways to represent the subset A(G). The "upwards inheritance" can be implemented as follows: if a member of A(G) explicitly activates group G then all subgroups of G are activated implicitly. ARDs of several active subjects are combined in the same way as specified above, with the following exception: ARDs for subgroups of G with value - are not considered.

A New ARD Value. The rights package paradigm leads to situations in which several groups are active. This causes a problem if there are only two ARD values. Assume the following situation:

- group S1 has a certain right, that is access(S1,G,M) = '+' for some granule G and mode M;
- S2 is a subgroup of S1.

According to the delta rule, S2 should not have this right again, that is access(S2,G,M) = '+' should *not* hold. However, access(S2,G,M) = '-' would have to interpreted in the sense that S2 *must not* be able to perform operations of M on G; this is quite contrary to what was intended in this situation.

A solution to this problem would be an ARD with a value which *neither allows nor denies* the accesses of an access unit ((S2,G,M) in our example). We call such an ARD value **undefined** (? for short).

It will later turn out that one undefined value is not sufficient (due to the nesting of granules). We will defer this discussion and assume for the rest of this section that granules do not overlap.

Evaluation of ARDs. We are now able to specify how the ARDs of several subjects are combined. A process is allowed to perform operation o on a granule G iff

- there is a mode M such that o $\in$ M and such that there is an active subject S such that access(S,G,M) = '+', and
- there is *no* mode M such that o $\in$ M and such that there is an active subject S such that access(S,G,M) = '-'.

In other words, ARDs of active subjects are combined according to the logic shown in table 1.

	+	?	-
+	+	+	-
?	+	?	-
-	-	-	-

Table 1: Combination of ARD values

It is not necessary to consider ARDs for outer granules of G due to reasons which will become clear in section 5.

5 Granules

This section explains important aspects of our concept which are related to granules.

The main problems addressed are the overlapping of complex objects and the efficient implementation of the evaluation of ARDs.

5.1 ARDs for Nested Granules

Derived vs. Dominating ARDs. There are two basic design choices concerning the nature of ARDs of complex granules:

The ARDs are **derived** from the ARDs of the components of this granule. Whenever an ARD of a component is changed the ARD of the whole granule can change as a side-effect. The obvious problem is that such side-effects may be unwanted. A typical example is the ownership problem mentioned in section 2.2.4. With derived ARDs, *it is impossible to enforce consistent access rights for all components of a granule.*

ARDs of a granule **dominate** (i.e. take precedence over) ARDs of components.

We adopt this approach because it removes the problems mentioned above.

Additional ARD Values. We come back to the problem that several groups might be activated at the same time and that they might have different ADR values for the same object and mode.

Assume that groups S1 and S2 are active for a process and that

access(S1,G,M) = '+', and
access(S2,G,M) = '?'.

We have to define the result of the *combination* of the values + and ?.

Assume further that G is a complex object. Now we have to distinguish two cases:

- if there is *no* inner granule G1 of G such that access(S2,G1,M) = '-' then the process is allowed to execute operations in M on G, that is the combination of the values + and ? is +.
- Otherwise, the process is not allowed to execute operations in M on G, that is the result of the combination is -.

We conclude that, *in case of complex granules, the ARD value ? does not contain enough information to compute its combination with +.* As a result, it would be necessary to *scan all inner granules of G* for a relevant denied ARD. This is unacceptable because of the overhead, distribution and other reasons (see section 2.5.1).

This problem leads us to introduce two new ARD values instead of ?, namely

?+ (or "**positive undefined**") and

?- (or "**negative undefined**").

Both indicate that no statement is made whether accesses of a *whole* access unit (S,G,M) shall be allowed or denied. The meaning of both values differs as follows:

?+ indicates that there is *no* inner granule G1 of G such that access(S,G1,M) = '-' or '?-'.

?- does not impose any such restriction, that is there can be zero, one or more inner granules G1 of G such that access(S,G1,M) = '-' or '?-'[20].

The value ?- is only meaningful for granules which can have inner granules, that is only for objects. The value ?- is therefore *only applicable to ARDs concerning objects.*

The evaluation rule remains the same as specified above with the addition that the value ?+ is treated like ? and that the value ?- is treated like -. The combination logic of all four values is shown in table 2.

[20]Both values allow a situation in which there is no inner granule with denied ARD. The difference between both values is that ?- allows to insert an inner granule with denied ARD, whereas ?+ does not allow to do this. Note that ?+ and ?- are in some sense similar to intention locks [Ko83].

	+	?+	?-	-
+	+	+	-	-
?+	+	?+	-	-
?-	-	-	-	-
-	-	-	-	-

Table 2: Combination of 4 ARD values

5.2 Consistency Rule for Inner Granules

A new problem caused by dominating ARDs is that, given a shared granule G, it may have two outer granules which are not inner/outer granule of each other and which have with contradicting ARDs. This situation is semantically inconsistent and must be avoided. Therefore, we introduce the following consistency rule:

If G2 is an inner granule of G1 then

- *access(S,G1,M)* = '+' *implies*
  access(S,G2,M) = '+',
- *access(S,G1,M)* = '?+' *implies*
  access(S,G2,M) = '+' *or* '?+',
- *access(S,G1,M)* = '-' *implies*
  access(S,G2,M) = '-'.

There is no such rule for the value ?-.

Any attempt to set ARDs such that the consistency rule would be broken is rejected.

The consistency rule offers several very important advantages, both with regard to the clarity and implementability of our concept:

Conceptual Aspects. The consistency rule prevents complicated situations which are difficult to interpret and to understand by users and which are likely to lead to erroneous results when users change ARDs.

In fact, if there are any implicit ARDs for a granule then their value is the same like the value of the explicit ARD[21]. Consequently, *implicit ARDs need not be considered at all in the evaluation of ARDs!*

Without the consistency rule, the notion of an implicit ARD must be part of the concept[22] and there needs to be a complicated definition how ARDs are evaluated.

Implementation Aspects. We assume that all ARDs for an object are stored in an **access control list**[23] (**ACL**) for this object. The important point is that the set of all ARDs of an object can be retrieved and changed very efficiently.[24]

We assume (like PCTE) that the root node of a complex object and a component object can reside on different sites (or volumes). Because of the reasons mentioned in section 2.5.1, it must be possible to evaluate the ARDs of an object O without accessing the ACL of an outer or inner granule of O which resides on another segment.

This problem gives another strong motivation for the consistency rule for inner granules: without this rule, implicit ARDs would be relevant in the ARD evaluation. The ACL of all outer granules whose root node is stored on another volume would have to be available; thus they would have to be *copied* into the segment in which O is stored, at least in the case of all granules which have *direct* outer granules on a different segment (note that this may even apply for relationships). This would lead

[21] Note: an ARD with value ?+ or ?- for a complex object does not imply any implicit granted or denied ARDs for the inner granules.

[22] Note that we have introduced this notion only for a discussion of general problems and design alternatives.

[23] An ACL is often understood as an *ordered* set of ARDs; in our model, it is an unordered set.

[24] It is not strictly necessary to store an object and its ACL on the same disc page. However, in a distributed OMS (or an OMS with removable volumes) both must reside on the same site (or the same volume); see section 2.5.1.

FERNUNIVERSITÄT HAGER UNIVERSITÄTSBIBLIOTHEK

to a substantial storage and maintenance overhead.

If a complete complex object is stored within one volume - which is the normal case - then one can *save substantial amounts of storage space*: ARDs for the complex object need not to be repeated in the ACL of the components. The price for this is that the evaluation of the ARDs of a component object requires to scan the ACLs of the outer granules. Moreover, changes of ARDs become more difficult to implement.

However, access rights differ typically only on the level of rather large objects, e.g. whole document versions, in typical tools within a software development environment. Objects below this level do typically not have additional ARDs; in other words, *no ARDs at all need to be stored for the large number of small objects.* The boundary between large-grain and small-grain objects can be determined statically within a schema (e.g. by adopting the two-tier database approach of ECLIPSE [CaA87]). Ideally, however, the OMS should dynamically optimize the space/time trade-off in the management of ACLs.

Finally, it is very important that the evaluation of ARDs is fast, in particular for complex objects. This gives another motivation for the consistency rule for inner granules: without this rule, it would be necessary to *scan all inner granules* when determining whether an intended access is allowed.

5.3 Operations

Basically, there is an operation `SetARD` to set an ARD and another operation `GetACL` to query all ARDs of an object.

Propagation. A consequence of the consistency rule is that two kinds of propagation in `SetARD` become necessary:

- If access(S,G,M) is set to + or - then this change must must be propagated to all *inner* granules. More precisely, if access(S,G,M) := V, V $\in$ {+,-}, then access(S,G',M) := V for all G' $\in$ PARTS(G).

  This rule applies also for the value ?+ with the exception that ?+ is not propagated to an inner granule G' if access(S,G',M) = '+'

- If access(S,G,M) is set to ?- then this change must must be propagated to all *outer* granules.

  If access(S,G,M) is set to ?+ then this change must must be propagated to all outer granules except those where access(S,G,M) = '?-'.

The above propagation rules apply *conceptually.* An implementation of the concept should actually be optimized along the lines indicated in the previous section. The resulting performance reduction can in any case be tolerated because ARD modifications are much less frequent than evaluations and are not time-critical.

Propagation into inner granules means that the old value of access(S,G',M) is replaced by the new one. The consistency rule for inner granules applies here: the operation fails if the rule would be broken. An example of such a situation is:

- G' is a component of both G and G1 (i.e. G' is a shared object),
- access(S,G1,M) = '-' and, thus, access(S,G',M) = '-', and
- assume the operation access(S,G,M) := '+' shall be executed.

This operation fails because the result would break the consistency rule.

Propagation to outer granules could occur as an inadvertent side-effect. Therefore, `SetARD` must have a parameter which specifies whether propagation shall occur or not. If propagation shall not occur, the operation fails if propagation would be necessary.

If access(S,G,M) is set to `?-` or `?+` and if G has inner granules then the ARDs of inner granules remain unchanged. If propagation to inner granules is desired then this can be specified by another parameter of `SetARD`.

Changes of the Component Structure. There are operations which make a granule G' a component of another granule G. G' can be "freshly created", for example a new component or a new internal relationship of a complex object is created; in this case we assume G' to have an initial set of ARDs determined by the creating process. G' can also exist already, e.g. an existing object is made a component of another object, in particular in order to share a component between two complex objects.

One effect of these operations is that all ARDs of G with value `+`, `?+` or `-` are propagated into G', in the same way as explained above.

The right to add or remove components of an object is controlled by an own access mode which is different from the mode which controls writing of the attributes (see next section). The operation which makes an object O a component of another object requires to be owner of O.

6 Access Modes

This section gives a summary of the aspects of our concept which are related to access modes.

There are 9 access modes. Table 3 shows for each access mode and each granule type whether the resulting set of operations is empty (by -) or non-empty (by a number). A short description of each operation set is given below.

access mode	complex object	root node	relationship
read	1	2	2
write	-	3	3
delete	4	-	5
append	-	6	6
execute	-	7	7
navigate	-	-	8
mod_comp	-	9	-
mod_rel	-	10	-
control	11	12	12

Table 3: Operation sets for access modes and granule types

1: copy the object

2: read the values of the attributes

3: modify the values of the attributes

4: delete the object

5: delete the relationship

6: append to the values of the attributes (applies only to long fields and strings)

7: execute the values of the attributes (applies only to long fields and strings)

8: navigate along the relationship

9: insert or delete direct components of the object

10: append a relationship to the object

11: change ARDs for this granule, make the object a component of another object

12: change ARDs for this granule

Note that an ARD for an object can imply ARDs for its inner granules which may

have a different type. For example, the append right on an object implies that there is also an append right on the root node which enables to append to all attributes of this object which have the type long field or string.

There are *no inclusions* of access modes. For example, the set of operations associated with 'write' right does not include the set of operations associated with 'read'. Inclusions lead to considerable complexity and additional special concepts. If desired they should be maintained by a tool which is used to modify ACLs.

7 Discussion

7.1 Extensions

There is an important extension of our approach which has not been presented here due to space considerations: There are also *access controls on the type level*, in addition to the controls on the instance level. For example, it is possible to grant only the group 'managers' the right to read and write objects of type 'management_report'. In general, subjects can be allowed or denied to access user-defined types of objects, attributes or relationships. This concept mainly serves as a "filtering mechanism" which allows to filter out parts of a complex object which belong to a specific role. Type level access controls allow to realize important access control features known from conventional database systems, viz. account level privileges and restrictions of the visibility and accessibility of fields or relations in the same way like external schemata. Types have owners like ordinary objects, only owners can grant or revoke type rights. Finally, inheritance hierarchies of object types are supported.

7.2 Other Approaches

Although the importance of object- and group-oriented access controls in OMSs has been emphasized at many occasions, there are actually only very few attempts to develop concepts to solve the related problems.

Some proposals cannot sensibly be compared with our proposal because they refer to a substantially different data model (even though it is also called "object-oriented"). For example, some object-oriented DBS are extensions of conventional relational DBSs and have kept an SQL-like query language and data-dependent views as main DAC mechanism.

[FeGS89] for example presents a DAC concept for such "relational" OMSs. It mainly addresses type-level access controls and does not deal with hierarchical complex objects, sharing, or distribution.

In the following, we will only discuss DAC concepts which are compatible with our assumptions about the data model of the OMS.

[CAIS88] and [PCTE+88] present access control mechanisms which support nested groups, but they do not support nested complex objects and the task paradigm.

[KiBG89] presents a mechanism which supports complex objects as granules, but without a consistency rule as presented here. As a consequence, a highly complicated scheme for implicit ARDs on component objects needs to be introduced. Nesting of groups is *not* supported. It would be very difficult to do this without a consistency rule for inner granules. By he same reasons, it would be difficult to extend this approach on distributed OMSs.

An object management system with advanced access control features is PVS [PVS87]. The main difference between our

concepts and PVS is that PVS is based on *asymmetric sharing* (see footnote 7), i.e. each shared object has one "main" superobject. Symmetrical sharing, which we have chosen, offers advantages over asymmetric sharing in many applications. Asymmetric sharing simplifies many problems; e.g. it allows to use simple rules for combining "implicit" ARDs of a shared object. Consequently, PVS does not have a consistency rule for ARDs of inner granules. Moreover, PVS has only two ARD values, which correspond to + and ?+.

A completely different approach to group-oriented access controls in an OMS is taken in DAMOKLES [DAMO88]. DAMOKLES separates the complete database into segments called 'database'. Each 'database' is owned by one user or user group. User groups have a hierarchical (i.e. partially ordered) structure. A user or group can arbitrarily access its own 'databases' and under certain restrictions the 'databases' of super- and subgroups. In other words, all objects within one 'database' have the same ARDs. This allows to simplify access controls considerably. If an object shall be accessed by several groups then it may become necessary to copy the object into several 'databases'; this is the most important disadvantage of this approach. Copies can be produced using the commands copy or check-out. The copies can be changed independently (with some exceptions for check-out copies); thus it is not always possible to simulate an object with n ARDs by copies of this object in n different 'databases'.

Relation to Group Transactions. The problem of supporting nested working groups which use a design environment has also been addressed by another concept, namely design and group transactions (e.g. in [BaKK85, Kl&85]). One of the features of such transactions is that they realize nested private databases for nested working groups. Objects are exchanged between these databases via check-out and check-in operations. The privacy aspect of group transactions can easily, and should, be implemented with group-oriented DAC [Ke89]. Moreover, if an OMS provides DAC then group transactions are only practicable if the DAC supports the task paradigm, because check-out and check-in usually include copying or versioning of complex objects.

Acknowledgements

The concepts presented in this paper are an extension and evolution of the concepts developed in the German PCTE Initiative [GPI89, Pe89] which are themselves an evolution of the concepts of PCTE+ issue 3 [PCTE+88]. Moreover, the author has gained many insights from discussions with other members of the ECMA Technical Committee 33 (PCTE) and its technical working group TGEP. The author would like to acknowledge the contribution of all members of these groups to his work; particular thanks are due to R. Minot, E. Petry and M. Simon.

References

[BA89] Base/OPEN Database Management System, CDDL and NQL Reference Manuals; TeleLOGIC Sundsvall AB; 1989

[BaBK88] Batz, T.; Baumann, P.; Köhler, D.: A data model supporting system engineering; p.178-185 in: Proc. COMPSAC; 1988

[BaKK85] Bancilhon, F.; Kim, W.; Korth, H.F.: A model of CAD transac-

tions; p.25-33 in: Proc. VLDB 85; 1985

[CaA87] Cartmell, J; Alderson, A.: The ECLIPSE two-tier database interface; in: Proc. ESEC 87, Strasbourg; 1987

[CAIS88] Common Ada Programming Support Environment (APSE) Interface Set (CAIS), Revision A; DoD-STD-1838A; 1988/05

[Ch76] Chen, P.P.: The entity relationship model: towards a unified view of data; ACM ToDS 1:1, p.9-36; 1976/03

[DAMO88] DAMOKLES - Database Management System for Design Applications, Reference Manual, version 2.0; FZI, Karlsruhe; 1988/03

[Da83] Date, C.J.: An introduction to database systems, vol. 1 & 2; Addison-Wesley; 1983

[De82] Denning, D.E.: Cryptography and data security; Addison-Wesley; 1982

[Di86] Dittrich K.R.: Object-Oriented Database Systems: The Notion and the Issues; p.2-4 in: Proc. ACM / IEEE Int.l Workshop on Object-Oriented Database Systems, IEEE Catalog Nr. 86TH0161-0; 1986

[DiHP88] Dittrich, K.R.; Härtig, M.; Pfefferle, H.: Discretionary access control in structurally object-oriented database systems; in: Proc. IFIP WG 11:3 Workshop on Database Security, Kingston, Ontario, Canada; North-Holland; 1988

[DOD83] Department of Defense trusted computer system evaluation criteria; DOD document CSC-STD-001-83; 1983/08

[Do&85] Downs, D.D.; Rub, J.R.; Kung, K.C.; Jordan, C.S.: Issues in discretionary access control; p.208-218 in: Proc. 1985 Symp. on Security and Privacy, Oakland, April 22-24, 1985; 1985

[EURAC89] Requirements and Design criteria for Tool Support Interface, Version 4; Report, IEPG TA 13; 1989

[FeGS89] Fernandez, E.B.; Gudes, E.; Song, H.: A security model for object-oriented databases; p.110-115 in: Proc. 1989 Symp. on Security and Privacy, Oakland; 1989

[FeSW81] Fernandez, E.B.; Summers, R.C.; Wood, C.: Database security and integrity; Addison Wesley; 1981

[GPI89] German PCTE Initiative: Introduction to the specifications of the GPI-OMS-Data-Model; GPI; 1989

[GPI89a] German PCTE Initiative: Requirements for the enhancement of PCTE/OMS, Version 2.0; GPI; 1989/03

[GrS87] Greif, I.; Sarin, S.: Data sharing in group work; ACM TOIS 5:2, p.187-211; 1987/04

[Ha&89] Haabma, J.; et al.: The NMP-CADLAB Framework; in: Proc. IFIP TC10/WG10.5 Working Conference on Very Large Scale Integration, Munich; 1989/08

[Hs87] Hsiao, D.K.: Database security; Naval Postgraduate School, Monterey, California, NPS52-87-048; 1987/11

[ITS89] German Information Security Agency: IT Security Criteria (Criteria for the Evaluation of Trustworthiness of Infor-

mation Technology Systems); Bundesanzeiger Verlag; 1989

[Ke88] Kelter, U.: Requirements on access control mechanisms in object management systems with autonomous user groups (in German); SWT Memo 31, Dep. Computer Science, University of Dortmund, ISSN 0933-7725; 1988/12

[Ke89] Kelter, U.: Gruppen-Transaktionen vs. gruppenorientierte Zugriffsrechte; p.287-300 in: Proc. GI Jahrestagung 1989, Springer Verlag; 1989/10 (english version available as: SWT Memo 37, Dep. of Computer Science, STL, University of Dortmund, ISSN 0933-7725)

[Ke90] Kelter, U.: Group paradigms in discretionary access controls for object management systems; to appear in: Proc. Ada Europe International Workshop on Environments, Chinon, September 1989; LNiCS, Springer Verlag; 1990

[KiBG89] Kim, W.; Bertino, E.; Gara, J.F.: Composite objects revisited; p. 337-347 in: Proc. SIGMOD 89; 1989/02

[Kl&85] Klahold, P.; Schlageter, G.; Unland, R.; Wilkes, W.: A transaction model supporting complex applications in integrated information systems; p.388-401 in: Proc. SIGMOD 85; 1985

[Ko83] Korth, H.F.: Locking primitives in a database system; JACM 30:1, p.55-79; 1983/01

[NDL87] ISO/TC97/SC21/WG3: Database Language NDL (ISO Standard 8907-1987(E)); ISO; 1987

[PCTE87] PCTE Project Team: PCTE: a basis for a portable tool environment; p.53-71 in: Proc. ESPRIT'86, Results and Achievements; Elsevier Science Publ.; 1987

[PCTE88] PCTE Interface Control Group: PCTE - A Basis for a Portable Common Tool Environment, Functional Specifications, Version 1.5; 1988/11/15

[PCTE+88] PCTE+ Functional Specification, Issue 3; IEPG TA-13; 1988/10/28

[Pe89] Petry, E.: Access control in a standard database for software development environments (in German); p.171-175 in: Proc. BTW89, Springer Verlag, Informatik Fachberichte 204; 1989

[PVS87] System PVS from its users' point of view, Version PVS/6 (in German); Softlab GmbH; 1987