Computers & Security, 13 (1994) 437-450



# Discretionary access control by means of usage conditions

Eike Born and Helmut Stiegler

*Siemens Nixdorf Informationssysteme AG, Otto-Hahn-Ring 6, D-81739 München, Germany*

A data processing environment allows users to create data objects of very different types. As a part of the management of these objects, the data processing system must support user-defined access control. In this paper we propose a general concept and user interface for user-defined access control and discuss a number of applications. The concept regards access control information as valuable information in its own right and consequently places this information, nowadays mostly specified within and as an integral part of meta-information either of objects or subjects, in objects of its own, called usage conditions. Each of these objects contains access control information corresponding to a task of the real world and formulated with general attributes. Access control for an object is implemented by references to appropriate usage conditions. Access to the object is granted if granted by at least one usage condition. Thus, by setting such references in an *m*-to-*n* manner, a restricted set of usage conditions is sufficient to implement even complex access control conditions.

*Keywords:* Access control information, Access relation attributes, Discretionary access control, Permission sets, Usage conditions, User interface.

## Introduction

A data processing system offers data objects of different types, mostly representing irreplaceable values to the user. Hence the system has to offer protection for these objects against unauthorized usage, which may lead to a loss of confidentiality, integrity or availability of data. Any access to data objects can either be controlled by fixed predefined rules ('mandatory' access control), or will be defined by users of data objects as the owner or the creator, according to their needs ('discretionary' or 'user-defined' access control). In this paper we concentrate on the latter kind of access control.

The systems usually discussed for user-defined access control are based on access control lists (ACLs) or capabilities, attaching the access information to the corresponding objects or subjects (see [1] or [2]). As is discussed, for example, in refs. [3–5], most solutions of this type lack generality, uniformity, standard logic and object class independence and do not satisfy the requirements of easy maintenance, precision and invariance. But also in connection with advanced concepts of access control in distributed systems as proposed in [6] or [7], and with rule-based mandatory access control as introduced in [8], [9], [10] or [11], there is still a need for a general and uniform concept of specifying and managing access control information.

In this paper we propose a concept of user-defined access control, satisfying most of the requirements above. We regard access control information as a value in its own right, and place this information into objects of its own, called usage conditions. Each of these objects contains access control information corresponding to a task of the real world and formulated with general attributes. Access

0167-4048/94/$7.00 

control for an object is implemented by references to appropriate usage conditions. Access to an object is granted if granted by at least one referred usage condition. Thus, by setting references in an *m*-to-*n* manner, a restricted set of usage conditions is sufficient to implement even complex access control conditions.

In terms of intended applications and benefits, usage conditions compete with the concept of roles. Roles are often introduced as a remedy for the problems mentioned above, in particular for easy maintenance and precision of access rights (see for instance [12]). A role comprises all access grants necessary for a certain duty or function of a subject. It therefore introduces an intermediate layer between subjects and objects, comparable to usage conditions. In fact, roles are an improvement, for instance, if the overall number of roles is much less than the number of users, and if roles are more static than the population of users. But, in general, roles call for centralized management comparable to user management, and hence only partly solve the requirements of easy and user-defined maintenance.

The concept of usage conditions comprises two different approaches to roles. If roles are supported by the system via a function reporting the role of a user, entrustment with a role can be used within usage conditions as can any other attribute of a user. Otherwise, roles can be introduced by usage conditions with appropriate user attributes, which can be used globally in parallel with usage conditions controlled by individual users. But usage conditions are more powerful than roles. They may refer not only to attributes of subjects but also to those of objects and of the context of an access. In addition they can be hierarchically defined and recursively applied.

In Section 1 of this paper we discuss basic requirements for an access control system, and list drawbacks of existing solutions. Section 2 introduces the proposed concept. Section 3 is devoted to the corresponding user interface, which provides support for the concept in a very convenient way. A basic example of how to specify and apply usage conditions is contained in Section 4. Problems arising in establishing discretionary access control by usage conditions in a data processing environment are discussed in Section 5, while Section 6 addresses some implementation problems and experiences. Finally, Section 7 contains a conclusion and an outlook on further investigations of access control systems.

## 1. Requirements and problems encountered

In a computing system, protection of data objects against misuse is primarily the responsibility of the owner of a data object, who is defined explicitly or implicitly by this very right. It is well known that the complexity inherent in specification, maintenance and application of access control information requires a degree of diligence that owners of objects are usually not able or willing to support, and hence requires particular support by the computing system. The design of a system for user-defined access control should thus meet two general requirements: specification of access control information should allow implementation of simple access structures in a simple way as well as very complex access structures (perhaps in a way that is not as simple); maintenance of access control information should allow the user to handle the commonly encountered dynamics of this information.

These general requirements lead to more specific demands. First of all, specifying access control information should obviously be the same for objects to be protected from all object classes. Secondly, specifying access control should be possible with respect to different access modes of an object. It should also be possible to refer to many different conditions for specifying access control, such as attributes of the subject as group membership or privileges, attributes of the object as location, name or value-dependent characteristics, or attributes of the access situation itself as access mode or time of access, and logical com-

binations thereof. Finally, a discretionary access control facility must meet the requirements of widely accepted standards of secure computing systems such as [13] or [14].

Most of the proposed solutions for user-defined access control based on access control lists or capabilities (see, for example, [1] or [2]) do not fulfil these requirements. Primarily, problems arise from the specification of access control information. Access mode dependent specification introduces object class dependence and hence a rather high complexity. In order to allow for formulation of positive and negative entries for access control conditions at the same time, with the same mechanisms and with approximately the same effort, many concepts support both positive and negative expressions such as 'user A is allowed' as well as 'user B is excluded'. This introduces a kind of three-valued logic, which is one additional source of complexity. Finally, user interfaces often lack functional flexibility, which restricts specifiable sets of usable access conditions. For instance, no complete logic is available, or the set of usable attributes is incomplete (e.g. 'on Monday' must be expressed as '7.5.; 14.5.; 21.5.; ...' or 'system administrator' must be expressed as 'Watson; John; Brown').

Additional drawbacks originate from poor support for management of access control information. Experience shows that data corruption is due mainly to inconsistencies between access control information and the surrounding world (e.g. entries for users already removed from the system). Either references from relevant users, objects and their attributes to access control information are missing, or changes of these attributes are made ignoring such references, for instance because their interpretation with respect to the impact on access control requires additional semantic information such that it cannot be automated.

Finally, deficiencies of existing access control facilities must be seen in their diversity and their integration into existing systems. Different systems support different mechanisms, expect different information and evaluate this information in a system-specific way. The meaning of access control information thus depends on the implementation and is therefore hard to understand. Also, many proposed concepts do not allow an embedding of existing access control facilities without problems, thus limiting their practical value [4, 9].

## 2. The concept of usage conditions

In order to cope with all the requirements and to overcome the drawbacks listed above, we consider access control information not as part of meta-information of either objects or subjects, but as valuable information in its own right. Consequently this information should be placed into objects of its own. To this end we propose:

- the introduction of an object class 'usage condition', the objects thereof containing access control information from one real world task to be executed;

- specification of access control information therein, based on whatever attributes the user may need;

- specification of access control information in well-known set logic, and representation and combination of this information in disjunctive normal form; and

- object class and access mode independent application of usage conditions by defining references from pairs (object, access mode) to usage conditions in order to specify the protection required.

A usage condition contains all access control information necessary for successful execution of a certain task. The object-independent part of this information is formulated within the usage condition by means of attributes, supported in evaluation by the system. The information relating to objects and access modes is specified by references from these pairs to the usage conditions. Different usage

conditions for one pair, 'object, access mode', are combined logically as an or-combination. Figure 1 illustrates the relationship of objects of class 'usage condition' to objects of different classes to be protected.

### 2.1. Why 'usage condition'?

Access control is intended to make objects accessible and usable for a set of users to be specified with the help of the access control system as precisely as possible. In this context the usage of an object is required by a certain task or process or piece of work for which the object is needed, as for instance designing a machine, managing payment lists or developing software. Such a task implies conditions of usage. On the one hand it defines objects to be necessarily used for successful completion of the task; on the other hand it defines subjects that are concerned with this task and should therefore be able to use the specified objects. Hence we choose the name 'usage condition' for units of access control information corresponding to a task to be executed and specifying the 'conditions of usage' for the necessary objects.

### 2.2. Support of attributes for specifying access control information

One may argue that, for specifying access control information for a data object, it is sufficient to enumerate all subjects (by name) to which access to the object is granted. In the case of many subjects, or if one wants to exclude only a few subjects from access, this will obviously be inconvenient. But in fact such an enumeration is impossible whenever access control information changes in time. Subjects would have to be enumerated with respect to conditions known not at the moment of specification but only at the very moment of access, as privileges (that may be revoked) or certain access times. Hence, referring direct to attribute values is necessary for the precise specification of access control information.

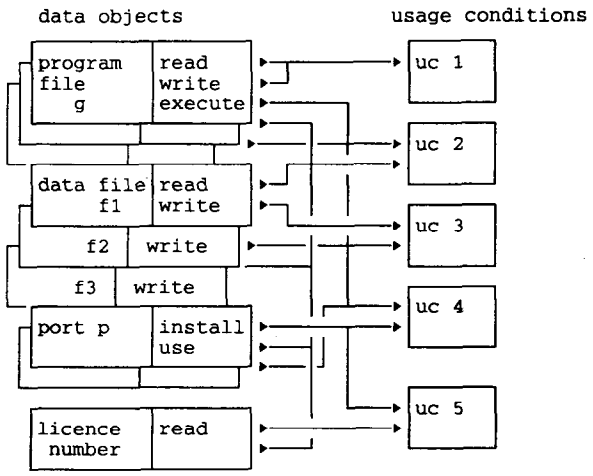


Fig. 1. Relationship between protected objects and usage conditions.

### 2.3. Combination of usage conditions

In general, a data object may be necessary for more than one task. Hence it must be possible for (an access mode of) an object to refer to several usage conditions. This combination will not lead to unintended or incompatible access-granting conditions if and only if the usage conditions are specified independently, are kept independently, and each of them grants access independently of the others. The last independence is ensured only if a combination of usage conditions is evaluated as follows: access to an object is granted if it is granted by at least one of the referred usage conditions. In this sense the usage conditions have to be combined via a logical 'or'.

Furthermore, the purpose of access control information should primarily be seen not in excluding persons from access to objects but in making objects available to a precisely limited population. Hence access control information should naturally describe the population to which access is granted, and not the users to whom the access is denied. This positive specification of access control information corresponds to a union of all subpopulations admitted to access, and hence also suggests combination of usage conditions by an 'or' instead of an 'and'.

Due to the complexity of access control information, there may be usage conditions with overlapping specifications, or usage conditions included in another in the sense that one usage condition

grants access in at least every situation that another does. If these usage conditions are not easily discernible, reshaping of a single usage condition may not have the expected effect on the overall access control information for an object, according to the proposed combination rule. But access control basically concerns information on tasks to be executed. And the proposed concept requires that each usage condition should represent an invariant formulation and documentation of such an independent task. Therefore overlapping is by chance only. A single usage condition can and must be reshaped independently of these incidental situations, such that it remains a precise description of the access control information of an independent task to be executed.

### 2.4. Access control information structure in a usage condition

The access control information of a usage condition can be expressed in natural language as one fairly complex logical formula connecting all attributes of interest by logical 'or', 'and' and 'not', representing the set of all subjects to whom access is actually granted. To obtain a normalized form of such an expression, it is perfectly natural to apply the above combination rule of different usage conditions internally for a single usage condition as well. This leads to the well-known disjunctive normal form of set theory (see, for example, [15]). In disjunctive normal form, a usage condition consists of the union ('or'-combination) of more basic sets; we shall call the latter sets 'permission sets'. A single permission set is the intersection of elementary access conditions. These conditions are chosen according to the convenience of the user but also in order to comply with the demand for precision (see Section 2). All accessible attributes should be supported for their specification.

As an example, elementary access conditions may be 'name of user = John', 'name of user = Ed', 'access-time = 8 a.m. to 5 p.m.' and 'used terminal = tty1 or tty3', where only the first two are elementary in the sense of natural language. Two permission sets may be

- PS1 = (name of user = John) and (access time = 8 a.m. to 5 p.m.),
- PS2 = (name of user = Ed) and (used terminal = tty1 or tty3),

such that the usage condition 'U = PS1 or PS2' grants access to John between 8 a.m. and 5 p.m. from every terminal, and to Ed at any time, but only from terminal tty1 or tty3.

Such a specification rule is complete in the sense of coping with any access-granting condition to be raised [16]. Prominent examples of ACLs can obviously be handled easily. In particular, (positive) enumerations of users remain the same. Simple cases, that cover the bulk of access control specifications we are dealing with today, remain simple. Complex cases, however, may require more complex specifications. The user interface, described in detail in Section 3, supports convenient formulation of access control information within this structure.

### 2.5. Referring to usage conditions

Access control information contains a great variety of permissions and restrictions with respect to many different objects and subjects. It is subject to dynamic modifications. The descriptions of access rights, for instance, vary at least every time a subject or an object is created or destroyed. One observes that creation or deletion of a data object occurs more frequently and is done more easily than the introduction or deletion of a registered subject. Hence information concerning objects displays greater dynamics than that concerning subjects. This suggests the storage of constant information (concerning subjects and tasks to be executed) within the usage conditions, and the specification of dynamically changing information (concerning objects) by referencing only. This supports a convenient initialization and frequent update of access control information by setting appropriate references, and copes with changes in the object world without modification of the usage conditions.

In addition, the need for additional access information arises primarily whenever a new object is created and the owner or creator of the object is asked to supply it. Thus user-defined access control information will be specified to set up object protection rather than subject access granting. This requires comfortable initialization and update of access control information with respect to objects as well.

Hence, in what follows, we adopt an 'ACL-like' solution for the access control concept, asking for references from objects to usage conditions, instead of a 'capability-like' solution with references from subjects to the access control information. Nevertheless, all proposed mechanisms work as well for the latter concept (by formulating access control information for objects within the usage conditions, and defining references from subjects to usage conditions).

#### 2.6. Access-mode-specific references

The objects necessary for executing a certain task may be of different classes, which are defined by the functions ('methods') applicable to these objects. The methods are in fact the access modes offered by the object class. As usage conditions (and the access control information they represent) are to be kept independent of objects, access modes in particular must not appear in their specification. Nevertheless, a task to be carried out may need only specific access modes for each object. Hence we propose to connect the access control information represented by a usage condition not to the complete object but to an access mode of an object only.

## 3. The user interface for usage conditions

The user interface for usage conditions has to cope with the two essential tasks of specifying and administering access control information, crucial for adequate protection of data objects according to the discussion in Section 1. The object class 'usage condition' is introduced as a subclass of the class 'data object' with the typical operations:

- `create`
- `modify`
- `show`
- `delete`
- `evaluate`.

The operations `create`, `modify`, `show` and `delete` are accessible to the user for explicit creation, modification, display and deletion of a usage condition, as for data objects of other classes. The operation `evaluate` is characteristic for the class usage condition. It is invoked only implicitly in the moment of an access demand and leads to an evaluation of the concerned usage conditions.

Via commands supported by the various object classes, references can be set from each access mode of an object to usage conditions. Finally, additional commands for different purposes can be designed on the base of the concept of usage conditions.

Creation and modification of a usage condition seem not to be a problem of an appropriate command interface but one of the complexity of a user's intention. Nevertheless, the user interface for the above methods has to cope with harmonizing the natural language user's intention and the formal language system's representation of a usage condition. Hence we shall discuss these commands in some detail. To this end we have agreed upon the following notation. Assignment of values to operands is indicated by '='. The slash '/' is used for separating possible operand values, and 'list possible' means that it is possible to enumerate a list of values for the preceding operand.

#### 3.1. The 'create' operation

The operation `create-usage-condition` defines the name and content of a usage condition. It follows the disjunctive normal form and should support references to existing usage conditions to allow for a recursive structure of usage conditions. The complete specification shows an or-combination of usage conditions already specified and of permission sets yet to be specified (see Section 2.4). The latter consist of the intersection of elementary

access conditions described by values of the different attributes, either as the set itself or as its complement. For certain single-attribute-valued elementary access conditions the corresponding command may therefore read

```
create-usage-condition  name = ⟨name⟩
                        usage-conditions = none/⟨uc-name⟩ list possible,
                        permission-sets = none/
                            (user-admission = all/⟨user-id⟩,
                             user-exception = no/⟨user-id⟩,
                             group-admission = all/⟨group-id⟩,
                             group-exception = no/⟨group-id⟩,
                             prog-admission = all/⟨program-id⟩,
                             prog-exception = no/⟨program-id⟩,
                             ...
                            ), list possible
```

This operation creates a usage condition of a certain name (first operand) as the union of usage conditions already specified (second operand), and of a list of permission sets to be specified within the third operand. The different operand names of the permission set specification concern the different attributes, as user name, group name, or program name. For each attribute, the subname `admission` indicates that the permission set is a subset of the set specified by the following attribute value. The subname `exception` indicates that the complement of the permission set with respect to this attribute is specified. All operands are connected by an 'and' in the sense that an access is granted if it satisfies all specified conditions. One may specify several permission sets in this way. The complete usage condition is built as the union of these permission sets (and the named usage conditions).

For each attribute, it should be possible to specify logical expressions of different values instead of a single attribute value. This supports specification of more complex permission sets in one step according to the needs of the user. Then, of course, the user is responsible for the redundancy and clarity of the specified permission sets. For internal representation and display, these permission sets may nevertheless be transformed by the system—perhaps on request of the user—into a completely solved disjunctive normal form, consisting of elementary access conditions only.

### 3.2. The 'modify' operation

For an unambiguous modification of a usage condition, a command mode and an interactive mode should both be supported. A command mode is feasible only if the current state of a usage condition is completely known. Then a general search-and-replace mechanism based on plain term rewriting can be applied:

```
modify-usage-condition  name=⟨name⟩,
                        search-expression=⟨xxxx⟩,
                        term-to-be-replaced=⟨yy⟩,
                        replacing-term=⟨zz⟩,
```

where ⟨xxxx⟩ is any part of the named usage condition, ⟨yy⟩ has to be a part of ⟨xxxx⟩ and will be replaced by the term ⟨zz⟩. The command will be rejected if a replacement of ⟨yy⟩ by ⟨zz⟩ leads to syntactical inconsistencies. Additional modifying commands (such as `exclude-user` or `reinclude-user`, see Section 5.4) can be designed in this way in accordance with fixed parts of the usage conditions.

For an interactive modification, which in particular applies if the current state of the usage condition is not exactly known, editing of a usage condition must be preferred. It should be based on a normalized and completely solved representation of the usage condition. Referred usage conditions are displayed by name only. The different permission sets can be shown explicitly, for instance within different columns, whereas the rows refer to single attributes, as shown in Fig. 2. In this form, a function for displaying redundancy can also be incorporated easily. Such a form can in fact also be used to create a usage condition in a guided mode.

### 3.3. The 'delete' operation

The `delete` command specifies the name of the usage condition which will then be replaced by the empty set in all referred objects and usage conditions. The designer of the operating system or the security policy may decide whether deletion of a usage condition is allowed only if there are no references from objects to it, or whether the usage condition will be replaced by the empty set (no granting at all) in all relevant objects, possibly at the same time triggering a message to the owners of all these objects to enable them to specify the access control information anew.

**edit usage condition** *name*

**referred usage conditions:** *name, name, ...*

| **permission sets:** / **attributes** | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| **user-admission** | user1 | all | user3 | | | |
| **user-exception** | user5 | user2 | none | | | |
| **group-admission** | group1 group3 | group3 | all | | | |
| **prog-admission** | prog1 | all | prog3 | | | |
| ... | | | | | | |

Fig. 2. A layout for usage condition editing.

## 4. A basic application example

Let us give a simple example of how to specify and apply usage conditions. The file 'salary data' of salary data for the employees of a company has to be accessible for fixing the salary by the salary department and for paying the salary by the cashier. In the corresponding usage conditions 'salary fixing' and 'salary payment', all persons, groups, times and in particular programs necessary for the corresponding tasks are specified, as is described in Section 3. Payments by the cashier will be recorded in a file named 'total payment' that has to be accessible for the quarterly balancing by the accounts department by means of the usage condition 'perform balancing'. Finally, there may be a database application called 'salary management' necessary to support both files 'salary data' and 'total payment'. A usage condition 'program maintenance' comprises all users and conditions necessary to maintain this program but not to execute it directly.

The access control information of the system under consideration will be represented by at least four usage conditions: 'salary fixing', 'salary payment', 'perform balancing' and 'program maintenance'. The usage conditions 'salary fixing' and 'perform balancing', for instance, can be specified as follows:

```
create-usage-condition  name = salary fixing,
                        usage-conditions = controlling,
                        permission-sets =
                            (user-admission = Brown, Ellis, Jackson,
                             group-admission = salary depmt. 1,
                             terminal-admission = tty sd1 - sd10,
                             time-admission = working hours)
create-usage-condition  name = perform balancing,
                        usage-conditions = controlling,
                        permission-sets =
                            (user-admission = all,
                             group-admission = accounts depmt.,
                             time-admission = last 3 working days each quarter, working hours,
                             prog-admission = salary management)
```

where the usage condition 'controlling' ensures access to all payment transactions for the company headquarters' controlling department and may be specified elsewhere. A suitable way to specify access to the application and files under consideration by references to the appropriate usage conditions can be as follows. The corresponding reference between data objects and usage condition is displayed in Fig. 3.

```
set-application-access  application name = salary management,
                        read-access = program maintenance,
                        write-access = program maintenance,
                        execute-access = salary fixing, salary payment, perform balancing,
```

```
set-file-access  filename = salary data,
                 read-access = salary payment,
                 write-access = salary fixing
set-file-access  filename = total payment,
                 read-access = perform balancing,
                 write-access = salary payment
```

Already, within this simple example, different access modes for the same object require not only different but also disjoint sets of permissions, a single usage condition can be used for different objects, and different usage conditions are necessary to formulate the access control information for one particular access to an object.

## 5. Implementation of usage conditions in a data processing system

Obviously it is no trivial task to establish usage conditions as a complete and consistent discretionary access control system from scratch, and in particular within an existing data processing environment. In what follows we discuss some of the corresponding problems.

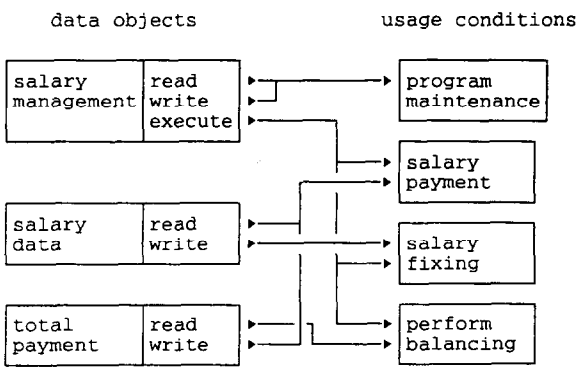


Fig. 3. Example for referring to usage conditions.

### 5.1. Access control for usage conditions

Usage conditions are data objects, like files or messages. Hence an access control system should also be able to control access to usage conditions with respect to the access modes 'modify' and 'evaluate'. Simple access control can be based on an ownership for usage conditions. Every owner of protectable objects may create or obtain usage conditions in order to establish necessary access control. He/she becomes owner of these usage conditions and may modify and use them exclusively.

Obviously owner-based access control for usage conditions does not allow for (modifying or) referring to usage conditions owned by other subjects. This is necessary, for instance, if usage conditions, globally defined and maintained by an administrator, are to be used for the access control of data objects of different owners in the sense of official distribution lists. The simplest way to obtain proper discretionary access control is thus to introduce at least two different types of usage conditions by means of access scopes. Usage conditions marked as 'globally accessible' can be referred to (but perhaps not modified) by all subjects, whereas other usage conditions can be referred to by their owners only. Of course, a finer granularity or hierarchy of access scopes may be chosen.

If even more specifically controlled access to usage conditions is desired, this can again be done exclusively by means of usage conditions. The

chain of usage conditions arising in the specification of access conditions by means of other usage conditions must obviously be finished by a loop to ensure its finiteness. Access to a usage condition is then controlled by a usage condition already in the chain (see Fig. 4). This loop may consist of only one usage condition, which has already been proposed in [9].

### 5.2. Backward references

For the automatic management of access control information (see Section 1) one must know which objects refer to a usage condition. Deletion of a usage condition, for instance, should be allowed only if there is no reference to it from any object (see 3.3). Or modification of a usage condition triggers the sending of corresponding messages to the owners of all concerned objects. This requires backward references from usage conditions to the referring objects.

Such a backward reference must be supplied by the object class of a protected object on setting up the reference. It consists of object class (universally unique), object characteristics such as name and access mode, and perhaps other attributes and comments. The only necessary restriction for the object-class-dependent information is the support of a common convention allowing the display of backward link information by an object-class-dependent `show-referring-objects` operation on usage conditions.

### 5.3. Embedding of the UNIX ACL

To see how integration of existing access control facilities works for the proposed concept (see Section 1), take the UNIX® ACL. It specifies the granting of access rights to a data object with respect to the access modes 'read', 'write' and 'execute' for the owner, one specified group and for all 'other' users. Evaluation of this information regards the subject specifications as disjoint sets of users and grants access to a user who belongs to at least one of the specified sets. For each of the three access modes above, this ACL, by its evaluation, represents access control information within three well-defined permission sets. The information can thus be transferred perfectly into three usage conditions as it is. In this way the UNIX ACL integrates perfectly into the concept of usage conditions. A corresponding extension of the UNIX ACL has already been proposed, without a precise justification, in [5].

### 5.4. Security standards and the concept of usage conditions

Usage conditions are a well-tailored means of realizing discretionary access control. They meet *per se* intentions and most of the requirements of widely accepted security standards of information technology such as, for example, [13] or [14]. One remark must be added concerning the requirement of single users from access to a data object [13] (class C2, B3 and beyond).

Such an exclusion from access to a specified object (and not from a task) is not well suited to the

®UNIX is a registered trademark of UNIX System Laboratories Inc.

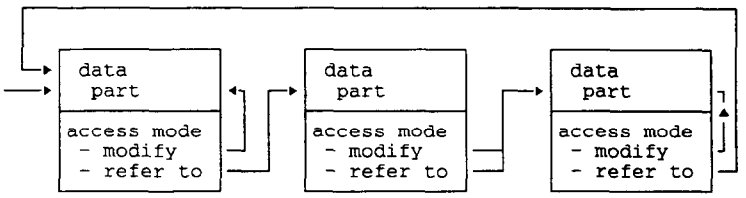


Fig. 4. Example of a loop of references for access control for usage conditions.

purpose of access control, which is to make objects accessible primarily to execute a certain task, and not for particular persons incidentally entrusted with this task (see Section 2). Nevertheless, such exclusion of a single user can be obtained, according to Section 3.2, by a particular command `exclude-user`, which introduces the access-excluding condition to all permission sets of all usage conditions referring to the object, and all usage conditions referred to therein. This obviously requires backward links to the referring objects (see Section 5.2).

If a relevant permission set already contains the corresponding access-granting condition, it is then in fact the empty set. It would be very easy to recognize and delete such redundant permission sets automatically. An interesting feature for implementation is to defer deletion until the user requests it explicitly. Such deferred deletion (and the resulting temporary redundant set specification) supports re-inclusion after exclusion as a transaction-like undo-feature (see Section 3.2).

### 5.5. Support of the Clark–Wilson security model

Clark and Wilson [17] proposed a semiformal model for secure data processing in the commercial world. It is based mainly on splitting up tasks and responsibilities between different persons. Users carry out their subtasks by means of well-defined transaction programs which preserve the integrity of data. The subtasks may be independent or may depend on the completion of the task as a whole.

The emphasis of the Clark–Wilson model on tasks and subtasks of the real world meets perfectly with the task-specific definition of usage conditions, as far as owner-controlled access rights are concerned. For each subtask of the Clark–Wilson model one can define a usage condition which includes all users entrusted with accessing the corresponding data. The encapsulation of data access by integrity-preserving transaction programs will then be enforced via the `prog-admission` attribute in addition to the user names (see Section 3.1).

## 6. Experience from an implementation

A particular, slightly restricted kind of usage condition for user-defined access control was implemented under the name 'GUARDS' in the BS2000® mainframe operating system of Siemens Nixdorf AG in extension of a UNIX-like basic ACL mechanism, in order to meet access control requirements of secure operating systems of [13], class B1. Protectable object classes include data files, ports, library elements, but also semaphores, for instance. The restriction concerns the formal specification of the usage conditions. Nevertheless, a large class of attributes for specifying access control conditions is supported, among which are user-name, group-membership, privileges, chip-card-ID, time of access, program-name, task type (batch, dialogue, ...), calendar functions, and terminal-ID. In addition, user- or administrator-defined attributes are supported by the system.

Two problems proved to be most crucial for the project. Access control information is frequently accessed, in particular for reading, and hence requires short access times. Thus optimal storage would be in the form of (main memory) caching. For large file systems, in particular, caching of the complete access control information is impossible. Hence the access control information of a data object is divided into one short part of most frequent access (and of fixed length), which is cached. The other part of the access control information is stored separately without the need for short access times. Such a division of access control information meets perfectly well with the disjunctive normal-form representation of usage conditions. They can easily be divided on the level of permission sets, as these are combined by an 'or' and so are evaluated independently.

Additional difficulties arose in integrating the concept of access control into data archiving. Data objects, formerly properly protected by usage conditions, no longer fit into an actual access control

®BS2000 is a registered trademark of Siemens Nixdorf Informationssysteme AG.

environment when they are brought back to the system after being stored in tape archives for a long time. It was decided in such a situation not to support the former access control for these objects, but to restrict access after import to the owner only, who then has to redefine proper access control.

## 7. Conclusion

We propose a general concept for implementing user-defined access control and have discussed different application aspects. Access control information is considered therein in the form of independently named data objects. It is completely separated from data objects and subjects and is managed within objects of a new class, called usage conditions. Each usage condition contains a unit of access control information, corresponding to a single user task to be executed and specifying the access conditions to resources necessary for a successful completion of this task.

The concept allows for specifying the access control information of a task once as a usage condition and then making references from all objects necessary to execute this task to the usage condition. Hence the number of usage conditions will be much smaller than that of the objects to be protected. As one object may also be necessary for more than one task, it may refer to several usage conditions. The access control information of these usage conditions remains independent, as the usage conditions are evaluated as a logical or-combination. The support of such $m$-to-$n$ references between usage conditions and objects allows for the specification of very fine-grained security policies with an overall number of usage conditions to be formulated and managed which is minimal with respect to the need of the users.

The combination rule for different usage conditions suggests disjunctive normal-form representation and specification of access control information also within each usage condition. It may not always be the shortest and most convenient structure, in particular for complex access control patterns. But it is based completely on set theory and so copes with every access control condition that can be formulated unambiguously. Finally, the proposed user interface supports our concept for specification and management of access control information in a highly comfortable way.

An extension of the proposed concept, still to be discussed, may be to remove even subject names from the usage conditions. The usage condition then represents solely a certain task of the real world. Subjects actually entrusted with this task are assigned to the usage condition by appropriate references, as is done for all necessary objects. This may be a further advantage for particular complex access control situations. But such an extraction of subject names from usage conditions would necessarily prohibit their management by ordinary users, and tends towards mandatory instead of discretionary access control. This exceeds the chosen scope of our investigation.

## References


[1] D. Dening, *Cryptography and Data Security*, Addison Wesley, Reading, MA, 1982.

[2] C.E. Landwehr, Formal models for computer security, *ACM Comput. Surv.*, 13 (1981) 247–278.

[3] M.D. Abrams, L.J. LaPadula and I.M. Olson, Building generalized access control on UNIX. *Proc. USENIX Workshop UNIX Secur., Portland, OR, Aug. 1990*, pp. 65–69.

[4] T. Lunt, Access controls: some unanswered questions, *Comput. Secur.*, 8 (1989) 43–54.

[5] H. Strack, Extended access control in UNIX System V-ACLs and context, *Proc. USENIX Workshop UNIX Secur., Portland, OR, Aug. 1990*, pp. 87–101.

[6] M. Abadi, M. Burrows, B. Lampson and G. Plotkin, *A Calculus for Access Control in Distributed Systems*, Digital Research Center, Palo Alto, CA, Feb. 1991.

[7] M. Satyanarayanan, The influence of scale on distributed file system design, *IEEE Trans. Softw. Eng.*, 18 (1992) 1–8.

[8] R.W. Baldwin, Naming and grouping privileges to simplify management in large databases, *Proc. IEEE Comput. Soc. Symp. Res. Secur. Priv., Oakland, CA, May 1990*, pp. 116–132.

[9] M. Gasser, A. Goldstein, C. Kaufman and B. Lampson, The Digital distributed system security architecture, *Proc. 12th Natl Comput. Secur. Conf., Baltimore, OH, 1989*, pp. 305–319.

[10] C.J. McCollum, J.R. Messing and L. Notargiacomo, Beyond the pale of MAC and DAC–Defining new forms of access control, *Proc. IEEE Comput. Soc. Symp. Secur. Priv., Oakland, CA, May 1990*, pp. 190–200.

[11] D.R. Wichers, D.M. Cook, R.A. Olsson, J. Crossley, P. Kerchen, K.N. Levitt and L.R. Lo, PACL's: an access control list approach to anti-viral security, *Proc. USENIX Workshop UNIX Secur., Portland, OR, Aug. 1990*, pp. 71–82.

[12] C.A. Ellis, S.J. Gibbs and G.L. Rein, Groupware–some issues and experiences, *Comm. ACM*, 34 (1991) 38–58.

[13] *Trusted Computer Systems Evaluation Criteria*, Department of Defense 5200.28-STD, Washington, DC, 1985.

[14] *Information Technology Security Evaluation Criteria, Version 1.2*, Office for official publications of the European Community, Luxembourg, June 1991.

[15] K. Kuratowski and A. Mostowski, *Set Theory*, North Holland, Amsterdam - New York - Oxford, 1976.

[16] H.G. Stiegler, A structure for access control lists, *Softw. Prac. Exper.*, 9 (1979) 813–819.

[17] D.D. Clark and D.R. Wilson, A comparison of commercial and military computer security policies, *Proc. IEEE Comput. Soc. Symp. Secur. Priv., 1987*, pp. 184–194.