Access control policies and languages

Sabrina De Capitani di Vimercati*, Sara Foresti
and Pierangela Samarati

DTI, Università degli Studi di Milano,
26013 Crema, Italy
E-mail: decapita@dti.unimi.it E-mail: foresti@dti.unimi.it
E-mail: samarati@dti.unimi.it
Website: http://www.dti.unimi.it/decapita
Website: http://www.dti.unimi.it/samarati
*Corresponding author

Sushil Jajodia

George Mason University,
Fairfax, VA 22030-4444, USA
E-mail: jajodia@gmu.edu
Website: http://csis.gmu.edu/faculty/jajodia.html

Abstract: Access control is the process of mediating every request to data and services maintained by a system and determining whether the request should be granted or denied. Expressiveness and flexibility are top requirements for an access control system together with, and usually in conflict with, simplicity and efficiency. In this paper, we discuss the main desiderata for access control systems and illustrate the main characteristics of access control solutions.

Keywords: authorisation hierarchies; positive and negative authorisations; attribute-based access control.

Reference to this paper should be made as follows: De Capitani di Vimercati, S., Foresti, S., Samarati, P. and Jajodia, S. (2007) 'Access control policies and languages', *Int. J. Computational Science and Engineering*, Vol. 3, No. 2, pp.94–102.

Biographical notes: Sabrina De Capitani di Vimercati is an Associate Professor at the Department of Information Technology of the University of Milan. She received her Laurea and PhD Degrees both in Computer Science from the University of Milan in 1996 and 2001, respectively. Her research interests are in the area of information security, databases, and information systems. She has been an International Fellow in the Computer Science Laboratory at SRI, CA, USA. She is co-recipient of the ACM-PODS'99 Best Newcomer Paper Award.

Sara Foresti received the Laurea Degree in Computer Science from the University of Milan, Italy in 2005. From April 2005, she has been a Research Collaborator at the Information Technology Department, University of Milan, Italy. Her research interests are in the area of data security and privacy.

Pierangela Samarati is a Professor at the Department of Information Technology of the University of Milan. Her main research interests are in data and application security, information system security, access control policies, models and systems, and information protection in general. She has participated in several projects involving different aspects of information protection. On these topics she has published more than 100 refereed technical papers in international journals and conferences. She is co-recipient of the ACM-PODS'99 Best Newcomer Paper Award.

Sushil Jajodia is University Professor, BDM International Professor of Information Technology, and the Director of Center for Secure Information Systems at the George Mason University, Fairfax, Virginia. His research interests include information security, temporal databases, and replicated databases. He has authored five books, edited 22 books, and published more than 250 technical papers in the refereed journals and conference proceedings.

1 Introduction

One of the most important features of today's systems is the protection of their resources (i.e., data and services) against unauthorised disclosure (*secrecy*) and intentional or accidental unauthorised changes (*integrity*), while at the same time ensuring their accessibility by authorised users whenever needed (*no denials-of-service*) (Samarati and De Capitani di Vimercati, 2001). Considerable effort is being devoted to addressing various aspects of secrecy, integrity, and availability. Although, historically, confidentiality has received the most attention, probably because of its importance in military and government applications.

One of the main security services to achieve data protection is access control. Access control is the act of ensuring that a user accesses only what she is authorised to and no more. Significant research has focused on achieving more expressive and powerful *access control systems*. The development of an access control system requires the definition of the regulations according to which access is to be controlled and their implementation as functions executable by a computer system. This development process is usually carried out with a multi-phase approach based on the concepts of *security policy*, *security model* and *security mechanism*. A policy defines the (high-level) rules according to which access control must be regulated. A policy is then accompanied by a language for the specification of the rules. An access control model provides a *formal* representation of the access control security policy and its working. The formalisation allows the proof of properties on the security provided by the access control system being designed (Landwehr, 1981). A security mechanism defines the low level (software and hardware) functions that implement the controls imposed by the policy and formally stated in the model.

The traditional access control models used for describing the enforcement of confidentiality are based on the definition of access control rules, called *authorisations*, which are of the form ⟨subject, object, operation⟩. These authorisations specify which operations can be performed on objects by which subjects. However, in today's systems the definition of an access control model is complicated by the need to formally represent complex policies, where access decisions depend on the application of different rules coming, for example, from law practices, and organisational regulations. A security policy must then combine all the different regulations to be enforced (Wijesekera and Jajodia, 2003) and, in addition, must consider all possible additional threats due to the use of computer systems. Given the complexity of the scenario, the simple authorisation triple ⟨subject, object, operation⟩ is no more sufficient.

The remainder of this paper is organised as follows. Section 2 discusses the main features supported by modern access control policies and models. Section 3 presents recent approaches in the area of access control languages. Section 4 introduces recent solutions basing the access control decisions on the evaluation of users' attributes rather than on their identity. Finally, Section 5 concludes the paper.

2 Policies and models

The access control service provided by the computer system should be expressive and flexible enough to accommodate all the different requirements that may need to be expressed, while at the same time be simple both in terms of use (so that specifications can be kept under control) and implementation (so to allow for its verification). In the following, we discuss the main features that an access control service should support.

2.1 Conditions and supports of abstractions

Even early approaches to authorisation specifications allowed *conditions* to be associated with authorisations to restrict their validity. Conditions can make the authorisation validity dependent on the satisfaction of some system predicates (*system-dependent* conditions) like the time or location of access. For instance, a condition can be associated with the bank-clerks' authorisation to access accounts, restricting its application only from machines within the bank building and in working hours. Conditions can also constrain access depending on the content of objects on which the authorisation is defined (*content-dependent* conditions). Content-dependent conditions can be used simply as a way to determine whether or not an access to the object should be granted or as way to restrict the portion of the object that can be accessed (e.g., a subset of the tuples in a relation). This latter option is useful when the authorisation object has a coarser granularity than the one supported by the data model (Date, 1995). Other possible conditions that can be enforced can make an access decision depend on accesses previously executed (*history dependent* conditions).

Another feature usually supported by today systems is the management of abstractions (groups of subjects and objects) in the authorisation specification. Even early approaches supported the specification and use within authorisations of *user groups* (e.g., Employees, Programmers, Consultants). Groups can be nested and need not be disjoint. Figure 1 illustrates an example of user-group hierarchy. Support of groups greatly simplifies management of authorisations, since a single authorisation granted to a group can be enjoyed by all its members. Later efforts moved to the support of groups on all the elements of the authorisation triple (i.e., subject, object, and operation), where, typically, groups are abstractions hierarchically organised. For instance, in an operating system the hierarchy can reflect the logical file system tree structure, while in an object-oriented system it can reflect the class (is-a) hierarchy. Figure 2 illustrates an example of object hierarchy. Even operations can be organised hierarchically, where the hierarchy may reflect an implication of privileges (e.g., write is more powerful than read (Rabitti et al., 1991)) or a grouping of sets of privileges (e.g., a 'writing privileges' group can be defined containing write, append, and undo (Shen and Dewan, 1992)). These hierarchical relationships can be exploited

- to support preconditions on accesses (e.g., in Unix a subject needs the execute privilege on a directory to access the files within it), or
- to support authorisation implication, that is, authorisations specified on an abstraction apply to all its members.

Figure 1 An example of user-group hierarchy

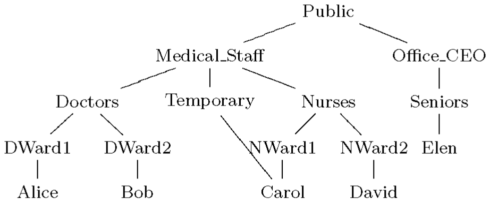
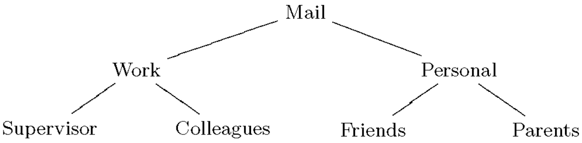


Figure 2 An example of object hierarchy



Support of abstractions with implications provides a short hand way to specify authorisations, clearly simplifying authorisation management. As a matter of fact, in most situations the ability to execute privileges depends on the membership of users into groups or objects into collections: translating these requirements into basic triples of the form ⟨user, object, operation⟩ that then have to be singularly managed is a considerable administrative burden, and makes it difficult to maintain both satisfactory security and administrative efficiency.

2.2 Positive and negative authorisations

Although there are cases where abstractions can work just fine, many will be the cases where exceptions (i.e., authorisations applicable to all members of a group but few) will need to be supported. This observation has brought to the combined support of both *positive* and *negative* authorisations. Traditionally, positive and negative authorisations have been used in mutual exclusion corresponding to two classical approaches to access control, namely:

- *Closed policy.* Authorisations specify permissions for an access. The closed policy allows an access if there exists a positive authorisation for it, and denies it otherwise.
- *Open policy.* (Negative) Authorisations specify denials for an access. The open policy denies an access if there exists a negative authorisation for it, and allows it otherwise.

The open policy has usually found application only in those scenarios where the need for protection is not strong and by default access is to be granted. Most systems adopt the closed policy, which, denying access by default, ensures better protection; cases where information is public by default are enforced with a positive authorisation on the root of the subject hierarchy (e.g., Public).

The combined use of positive and negative authorisations has been adopted in recent approaches as a convenient way to support exceptions. To illustrate, suppose we wish to grant an authorisation to all members of a group composed of 1000 users, except to one specific member Alice. In a closed policy approach, we would have to express the above requirement by specifying a positive authorisation for each member of the group except Alice.[1] However, if we combine positive and negative authorisations we can specify the same requirement by granting a positive authorisation to the group and a negative authorisation to Alice.

The combined use of positive and negative authorisations brings now to the problem of how the two specifications should be treated:

- What if for an access no authorisation is specified? (*incompleteness*)
- What if for an access there are both a negative and a positive authorisation? (*inconsistency*)

Completeness can be easily achieved by assuming that one of either the open or closed policy operates as a *default*, and accordingly access is granted or denied if no authorisation is found for it. Note that the alternative of explicitly requiring completeness of the authorisations is too heavy and complicates administration.

Conflict resolution is a more complex matter and does not usually have a unique answer (Jajodia et al., 2001b; Lunt, 1988). Rather, different decision criteria could be adopted, each applicable in specific situations, corresponding to different policies that can be implemented. Examples of different conflict resolution policies are given below.

Denials-take-precedence. Negative authorisations are always adopted when a conflict occurs (it satisfies the 'fail safe principle'). In other words, the principle says that if we have one reason to authorise an access, and another to deny it, then we deny it.

Most-specific-takes-precedence. A natural and straightforward policy is the one stating that "the most specific authorisation should be the one that prevails"; after all this is what we had in mind when we introduced negative authorisations in the first place (our example about Alice). Although the most-specific-takes-precedence principle is intuitive and natural and likely to fit in many situations, it is not enough. As a matter of fact, even if we adopt the argument that the most specific authorisation always wins (and this may not always be the case) it is not always clear what more specific is:

- What if two authorisations are specified on non-disjoint, but non-hierarchically related groups (e.g., NWard1 and Temporary in Figure 1)?
- What if for two authorisations the most specific relationship appears reversed over different domains? For instance, consider authorisations (Doctors, read+, Mail) and (Medical_Staff, read–, Personal); the first has a more specific subject, while the second has a more specific object (see Figures 1 and 2).

Most-specific-along-a-path-takes-precedence. This policy considers an authorisation specified on an element *x* as overriding an authorisation specified on a more general element *y* only for those elements that are members of *y* because of *x*. Intuitively, this policy takes into account the fact that, even in the presence of a more specific authorisation, the more general authorisation can still be applicable because of other paths in the hierarchy. For instance, consider the group hierarchy in Figure 1 and suppose that for an access a negative authorisation is granted to Medical_Staff while a positive authorisation is granted to Nurses. What should we decide for Carol? On the one side, it is true that Nurses is more specific than Medical_Staff; on the other side, however, Carol belongs to Temporary, and for Temporary members the negative authorisation is not overridden. While the most-specific-takes-precedence policy would consider the authorisation granted to Medical_Staff as being overridden for Carol, the most-specific-along-a-path considers both authorisations as applicable to Carol. Intuitively, in the most-specific-along-a-path policy, an authorisation propagates down the hierarchy until overridden by a more specific authorisation (Fernandez et al., 1994).

Priority level. The most specific argument does not always apply. For instance, an organisation may want to be able to state that consultants should not be given access to private projects, no exceptions allowed. However, if the most specific policy is applied, any authorisation explicitly granted to a single consultant will override the denial specified by the organisation. To address situations like this, some approaches proposed adopting explicit priorities; however, these solutions do not appear viable as the authorisation specifications may result not always clear.

Positional. Other approaches (e.g., Shen and Dewan, 1992) proposed making authorisation priority dependent on the *order in which authorisations are listed* (i.e., the authorisation that is encountered first applies). This approach, however, has the drawback that granting an authorisation requires inserting the authorisation in the proper place in the list. Beside the administrative burden put on the administrator (who, essentially, has to explicitly solve the conflicts when deciding the order), specifying authorisations implies explicitly writing the ACL associated with the object, and may impede delegation of administrative privileges.

Grantor- or time-dependent. Other possible ways of defining priorities can make the authorisation's priority dependent on the *time* at which the authorisations was granted (e.g., the more recent authorisations prevails) or on priorities between the *grantors*. For instance, authorisations specified by an employee may be overridden by those specified by her supervisor; the authorisations specified by an object's owner may override those specified by other users to whom the owner has delegated administrative authority.

As it is clear from this discussion, different approaches can be taken to deal with positive and negative authorisations. Also, if it is true that some solutions may appear more natural than others, none of them represents 'the perfect solution'. Whichever approach we take, we will always find one situation for which it does not fit. Also, note that different conflict resolution policies are not mutually exclusive. For instance, one can decide to try solving conflicts with the most-specific-takes-precedence policy first, and apply the denials-take-precedence principle on the remaining conflicts (i.e., conflicting authorisations that are not hierarchically related).

The support of negative authorisations does not come for free, and there is a price to pay in terms of authorisation management and less clarity of the specifications. However, the complications brought by negative authorisations are not due to negative authorisations themselves, but to the different semantics that the presence of permissions and denials can have, that is, to the complexity of the different real world scenarios and requirements that may need to be captured. There is therefore a trade-off between expressiveness and simplicity. For this reason, most current systems adopting negative authorisations for exception support impose specific conflict resolution policies, or support a limited form of conflict resolution (e.g., see the Apache server (http://www.apache.org/docs-2.0/misc/tutorials.html), where authorisations can be positive and negative and an ordering can be specified dictating how negative and positive authorisations are to be interpreted). More recent approaches are moving towards the development of flexible frameworks with the support of multiple conflict resolution and decision policies.

3 Languages for access control

The specification of the access control policies requires the use of a language to specify the access control rules as well as the possible assertions and properties of the different entities of the system (e.g., subjects/objects abstractions or properties) (Samarati and De Capitani di Vimercati, 2001).

We now describe some desiderata that an expressive and powerful access control language should satisfy. We then present a flexible and powerful access control language that addresses these desiderata (Jajodia et al., 2001b).

- An access control language should be simple and expressive. It should be simple to make easy the management task of specifying and maintaining the security specifications. It should be expressive to make it possible to specify in a flexible way different protection requirements that may need to be imposed on different objects.
- An access control language should support access rules (authorisations) to be referred to specific accesses, providing fine-grained reference to the subjects and objects in the system. More precisely, the language should provide support for authorisations specified for groups of users, groups of objects, and possibly even groups of actions.
- Protection requirements may need to depend on the evaluation of some conditions (e.g., system's predicates or conditions that make access dependent on the information being accessed). An access control language should then allow the specification of generic constraints on subjects, objects, and on contextual information.
- An access control language should support the definition of different types of access rules. Traditionally, there are access rules that specify the accesses that should not be allowed and access rules that specify the accesses that should be allowed.
- An access control language should support the definition of administrative policies that regulate the specification of access rules, that is, define who can add, delete, or modify them.

Several of the most recent language designs rely on concepts and techniques from logic, specifically from logic programming: Woo and Lam (1993), Li et al.'s D1LP and RT (Li et al., 2002, 2003; Li and Mitchell, 2003), Jim's SD3 (2001) and DeTreville's Binder (2002). Logic languages are particularly attractive as policy specification languages. One obvious advantage lies in their clean and unambiguous semantics, suitable for implementation validation, as well as formal policy verification. Second, logic languages can be expressive enough to formulate all the policies introduced in the literature. The declarative nature of logic languages yields a good compromise between expressiveness and simplicity. Their high level of abstraction, very close to the natural language formulation of the policies, makes them simpler to use than imperative programming languages. However, security managers are not experts in formal logics, either, so generality is sometimes traded for simplicity. For this reason, some languages do not adopt a first-order syntax, even if the policy language is then interpreted by embedding it into a first-order-logic. One of the major challenges in the definition of a policy language is to provide expressiveness and flexibility while at the same time ensuring easiness of use and therefore applicability. A promising solution in this direction is the proposal by Jajodia et al. (2001b) presented next.

3.1 A flexible authorisation framework

Jajodia et al. (2001b) worked on a proposal for a logic-based language that attempted to balance flexibility and expressiveness on one side, and easy management and performance, on the other. Their language allows the representation of different policies and protection requirements, while at the same time providing understandable specifications, clear semantics, and bearable data complexity. Their proposal for a *Flexible Authorisation Framework* (FAF) corresponds to a polynomial (quadratic) time data complexity fragment of default logic.

The components of a FAF are mainly the following:

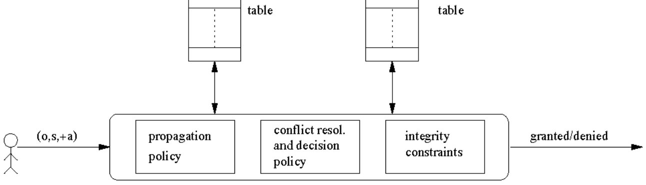
- *data items (objects)*: are the resources that can be accessed by system users; in most realistic systems, data items are organised hierarchically
- *access types*: are the different actions that users try to execute on different data items
- *users and group of users*: the word 'user' always refers to a human being, while a group is a non-empty set of users; in most applications, users and groups are organised into a hierarchy, which typically looks like a directed acyclic graph
- *roles*: are groups of privileges that a user can execute only when playing that specific role; roles too may be organised as a hierarchy
- *administration*: is a policy that regulates who can grant and revoke authorisations.

It is important to notice that groups of users and roles are not the same: groups are sets of people while roles are sets of privileges, furthermore, while roles can be activated and deactivated directly by users, groups belonging is not decided directly by users.

Formally, the *data system* (DS) is a 5-tuple (OTH, UGH, RH, A, Rel) where: OTH is an object-type hierarchy, UGH is a user-group hierarchy, RH is a role hierarchy, A is a set whose elements are called authorisation modes or actions and Rel is a set of elements called relationships. The three hierarchies have not common nodes.

In FAF, policies are divided into different decision stages (as in Figure 3), and the framework structure has the following components.

Figure 3 Functional authorisation architecture



Source: Jajodia et al. (2001b)

- A *history table* whose rows describe the accesses executed.
- An *authorisation table* whose rows are authorisations composed of the triples `(s, o, <sign>a)`, where *s* is the subject, *o* the data item, *a* the action and `<sign>` may be + if the action is allowed and – if it is denied. This is the set of explicitly specified authorisations.
- The *propagation policy* specifies how to obtain new derived authorisations from the explicit authorisation table. Typically, derived authorisations are obtained according to hierarchy-based derivation. However, derivation rules are not restricted to this particular form of derivation.
- The *conflict resolution policy* describes how possible conflicts between the (explicit and/or derived) authorisations should be solved.
- A *decision policy* defines the response that should be returned to each access request. In case of conflicts or gaps (i.e., some access is neither authorised nor denied), the decision policy determines the answer. In many systems, decisions assume either the open or the closed form (by default, access is granted or denied, respectively).
- A set of *integrity constraints* that may impose restrictions on the content and output of the other components; integrity rules are then used to individuate errors in the hierarchies or in the explicitly specified authorisations or for implementing duty separation.

The Authorisation Specification Logic language (ASL) is a logic language that is used to encode the system security needs. The predicates used to express the policy are the following, where *s*, *o* and *a* (representing the subject, object, and action of the authorisation, respectively) may be both constant or variable.

- `cando`(*o*, *s*, ±*a*) represents authorisations explicitly inserted by the security administrator. Each of them represents that the administrator wishes to allow, or deny (depending on the sign associated with the action), subject *s* to do action *a* on object *o*.
- `dercando`(*o*, *s*, ±*a*) represents authorisations derived by the system using logic program rules. `dercando` rules typically include in their body `cando` and `cando` literals (which represent the explicit or derived authorisations being propagated).
- `do`(*o*, *s*, ±*a*) definitively represents the accesses that must be granted or denied, enforcing conflict resolution and access decision policies. `do` rules can include in their body both `cando` and `dercando` literals expressing the presence/absence of authorisations affecting the decision.
- `done`(*o*, *s*, *r*, *a*, *t*) represents the fact that *s*, playing role *r*, performed action *a* over resource *o* at time *t*. This predicate is used to keep track of the history of accesses in the system.
- `error` represents the violation of an integrity constraint.

In addition, the language has a set of predicates for representing hierarchical relationships (`hie`-predicates), and additional application-specific predicates, called `rel`-predicates. Hierarchical predicates represent hierarchical relationships within the different components of the system (objects, subjects, or roles). These predicates can state if element *a* is a direct on indirect descendant of element *b* for the specified hierarchy. For instance, predicate `in`(*s*, *s′*ASH) where *s* and *s′* are subjects (i.e., users, groups, or roles) and ASH is the authorisation subject hierarchy obtained combining UGH and RH, evaluates whether *s* is a subgroup of *s′* in ASH. Application-specific predicates, instead, capture the possible different relationships, existing between the elements of the data system, that may need to be taken into account by the access control system. Examples of `rel`-predicates are `owner(user, object)`, which models ownership of objects by users, or `supervisor(user1, user2)`, which models responsibilities and control within the organisational structure.

The language for expressing authorisations is therefore based on few predefined predicates for the specification of authorisation rules (`cando`), the propagation policy (`dercando`), and the decision/conflict resolution policy (`do`). The structure of authorisation specifications guarantees stratification and hence, stable model uniqueness and PTIME computability. Policies are then expressed by a restricted class of stratified and function-free normal logic programs, called *authorisation specifications*. The semantics of authorisation specifications is the stable model semantics (Gelfond and Lifschitz, 1988).

While simple, the language proves quite expressive: administrators can specify, via the logic rules, any policy for propagating authorisations and resolving conflicts. Figure 4 illustrates the rules for propagating authorisations along a subject hierarchy ASH according to the commonly used overriding policies, including: *no propagation* (authorisations are not propagated along the hierarchy), *no overriding* (authorisations are propagated along the hierarchy, maintaining also more than a rule for each node), *most specific overrides*[2] (if two authorisations are inherited by a node, the one coming from the lowest parent is kept), *path overrides* (the label attached to a node *n* overrides a contradicting label of a supernode *n* for all the subnodes of *n* only for the paths passing from *n*). Figure 5 illustrates the conflict resolution/decision policies including: *no-conflict* (conflicts are considered errors), *denials-take-precedence* (negative authorisations prevail over positive ones), *permissions-take-precedence* (positive authorisations prevail over negative ones), and *nothing-takes-precedence* (the conflict remains unsolved). Some forms of conflict resolutions can be expressed within the propagation policy, as in the case of overriding (also known as *most-specific-takes-precedence*).

Figure 4 An example of propagation policies in ASL

Nopropagation

$$\begin{array}{lcl} \mathtt{dercando}(o,s,+a) & \leftarrow & \mathtt{cando}(o,s,+a). \\ \mathtt{dercando}(o,s,-a) & \leftarrow & \mathtt{cando}(o,s,-a). \end{array}$$

Nooverriding

$$\begin{array}{lcl} \mathtt{dercando}(o,s,+a) & \leftarrow & \mathtt{cando}(o,s',+a)\&\ \mathtt{in}(s,s',\mathsf{ASH}). \\ \mathtt{dercando}(o,s,-a) & \leftarrow & \mathtt{cando}(o,s',-a)\&\ \mathtt{in}(s,s',\mathsf{ASH}). \end{array}$$

Mostspecificoverrides

$$\begin{array}{lcl} \mathtt{dercando}(o,s,+a) & \leftarrow & \mathtt{cando}(o,s',+a)\& \\ & & \neg\mathtt{over}_{\mathsf{AS}}(s,o,s',+a)\&\ \mathtt{in}(s,s',\mathsf{ASH}). \\ \mathtt{dercando}(o,s,-a) & \leftarrow & \mathtt{cando}(o,s',-a)\& \\ & & \neg\mathtt{over}_{\mathsf{AS}}(s,o,s',-a)\&\ \mathtt{in}(s,s',\mathsf{ASH}). \\ \mathtt{over}_{\mathsf{AS}}(s,o,s',+a) & \leftarrow & \mathtt{cando}(o,s'',-a)\&\ \mathtt{in}(s,s'',\mathsf{ASH})\& \\ & & \mathtt{in}(s'',s',\mathsf{ASH})\&\ s''\neq s'. \\ \mathtt{over}_{\mathsf{AS}}(s,o,s',-a) & \leftarrow & \mathtt{cando}(o,s'',+a)\&\ \mathtt{in}(s,s'',\mathsf{ASH})\& \\ & & \mathtt{in}(s'',s',\mathsf{ASH})\&\ s''\neq s'. \end{array}$$

Pathoverrides

$$\begin{array}{lcl} \mathtt{dercando}(o,s,+a) & \leftarrow & \mathtt{cando}(o,s,+a). \\ \mathtt{dercando}(o,s,-a) & \leftarrow & \mathtt{cando}(o,s,-a). \\ \mathtt{dercando}(o,s,+a) & \leftarrow & \mathtt{dercando}(o,s',+a)\ \& \\ & & \neg\mathtt{cando}(o,s,-a)\ \&\ \mathtt{dirin}(s,s'). \\ \mathtt{dercando}(o,s,-a) & \leftarrow & \mathtt{dercando}(o,s',-a)\ \& \\ & & \neg\mathtt{cando}(o,s,+a)\&\ \mathtt{dirin}(s,s'). \end{array}$$

Figure 5 Conflict resolution/decision policies in ASL

CONFLICT	DECISION	RULES
No conflict	**open**	$\mathtt{error} \leftarrow \mathtt{dercando}(o,s,+a)\&\ \mathtt{dercando}(o,s,-a).$ $\mathtt{do}(o,s,+a) \leftarrow \neg\mathtt{dercando}(o,s,-a).$
No conflict	**closed**	$\mathtt{error} \leftarrow \mathtt{dercando}(o,s,+a)\&\ \mathtt{dercando}(o,s,-a).$ $\mathtt{do}(o,s,+a) \leftarrow \mathtt{dercando}(o,s,+a)\&\ \neg\mathtt{dercando}(o,s,-a).$
Denials take p.	**open**	$\mathtt{do}(o,s,+a) \leftarrow \neg\mathtt{dercando}(o,s,-a).$
Denials take p.	**closed**	$\mathtt{do}(o,s,+a) \leftarrow \mathtt{dercando}(o,s,+a)\&\ \neg\mathtt{dercando}(o,s,-a).$
Permissions take p.	**open**	$\mathtt{do}(o,s,+a) \leftarrow \mathtt{dercando}(o,s,+a).$ $\mathtt{do}(o,s,+a) \leftarrow \neg\mathtt{dercando}(o,s,-a).$
Permissions take p.	**closed**	$\mathtt{do}(o,s,+a) \leftarrow \mathtt{dercando}(o,s,+a).$
Nothing takes p.	**open**	$\mathtt{do}(o,s,+a) \leftarrow \neg\mathtt{dercando}(o,s,-a).$
Nothing takes p.	**closed**	$\mathtt{do}(o,s,+a) \leftarrow \mathtt{dercando}(o,s,+a)\&\ \neg\mathtt{dercando}(o,s,-a).$
Additional closure rule		$\mathtt{do}(o,s,-a) \leftarrow \neg\mathtt{do}(o,s,+a).$

Authorisation specifications are stated as logic rules defined over the predicates explained above. To ensure stratifiability, the format of the rules is restricted as illustrated in Figure 6. Note that the adopted strata reflect the logical ordering of the decision stages illustrated in Figure 3.

Each of the predicates here described is used within a specific component of the FAF architecture. The history table has only done rules; the authorisation table has only cando rules; the propagation policy contains mostly dercando rules; conflict resolution and decision policy have only positive do rules (i.e., with sign + for *a*) and an additional rule `do(o, s, -a)` ← ¬`do(o, s, +a)`, that guarantees completeness of the policy. In the end, the integrity module has only error rules.

Figure 6 Rule composition and stratification of FAF

Stratum	Predicate	Rules defining predicate

0	`hie-predicates`
	`rel-predicates`
	`done`
		Base relations.
		Base relations.
		Base relation.

1	`cando`	Body may contain `done`, `hie-` and `rel`-literals.
2	`dercando`	Body may contain `cando`, `dercando`, `done`, `hie-`, and `rel-` literals. Occurrences of `dercando` literals must be positive.
3	`do`	When head is of the form $\mathtt{do}(\_,\_,+a)$ body may contain `cando`, `dercando`, `done`, `hie-` and `rel-` literals.
4	`do`	When head is of the form $\mathtt{do}(o,s,-a)$ body contains just one literal $\neg\mathtt{do}(o,s,+a)$.
5	`error`	Body may contain `do`, `cando`, `dercando`, `done`, `hie-`, and `rel-` literals.

Jajodia et al. (2001b) present a materialisation technique for producing, storing, and updating the stable model of the policy. The model is computed on the initial specifications and updated with incremental maintenance strategies.

Note that the clean identification and separation of the four decision stages can be regarded as a basis for a policy specification methodology. In this sense, the choice of a precise ontology and other syntactic restrictions (such as those illustrated in Figure 6) may assist security managers in formulating their policies.

4 Attribute-based specifications

In an open system like the Internet, the different parties (clients and servers) that interact with each other to offer services are usually strangers. They have no preexisting relationship and are not in the same security domain. Therefore, on the one side the server may not have all the information it needs to decide whether or not an access should be granted. On the other side, however, the client may not know which information she needs to present to a (possibly just encountered) server to get access. All this requires a new way of enforcing the access control process, which cannot be assumed anymore to operate with a given prior knowledge and return a yes/no access decision. Rather, the access control process should be able to operate without a priori knowledge of the party requesting access and return the information of the requisites that it requires be satisfied for the access to be allowed (Bettini et al., 2002; Jajodia et al., 2001a). Also, the traditional "identity-based access control models", where subjects and objects are usually identified by unique names, are not appropriate in this setting. Instead, attributes other than identity are useful in determining the party's trustworthiness. In this context, access restrictions to the data/services should be expressed by policies that specify the properties (attributes) that a requester should enjoy to gain access to the data/services. Some proposals have been developed that use *digital certificates*. Traditionally, the widely adopted digital certificate has been the *identity certificate*. An identity certificate is an electronic document used to recognise an individual, a server, or some other entity, and to connect that identity with a public key (Blaze et al., 1996, 1998;

Chu et al., 1997). More recent research and development efforts have resulted in a second kind of digital certificate, the *attribute certificate* (Farrell and Housley, 2002) that can be used for supporting attribute-based access control. An attribute certificate has a structure similar to an identity certificate but contains attributes that specify access control information associated with the certificate holder (e.g., group membership, role, security clearance). One of the most important aspects that attribute-based access control policies should support is the ability to specify accesses to a *collection* of services based on a *collection* of attributes. In this context, logic programming provides a convenient, expressive, and well-understood framework in which to work with authorisation policy. Wang et al. (2004) propose a framework that models an attribute-based access control system using logic programming with set constraints of a computable set theory. More precisely, the set theory used in this approach is CLP($\mathcal{SET}$), the *hereditarily finite* and computable set theory developed by Dovier et al. (2000). Here, sets are constructed out of a finite universe by applying operators such as $\cap$, $\cup$ and so on. A policy can refer to both attributes and services, and a two sorted first order language with set variables is then used. The terms are constructed in the usual way by means of variables and functions. Also, the approach supports two kinds of predicates: those used to specify the computation domain and those used to specify its sub-domain of constraints. To reduce the runtime inefficiency of constrained logic programs, which is due to the backtracking through program clauses, two techniques are used. The first technique consists in transforming any attribute-based access control policy into one with less backtracking but the same semantics. The second technique consists in materialising predicate instances accessed repeatedly.

Bonatti and Samarati (2002) propose a uniform framework for regulating service access and information disclosure in an open, distributed network system like the web. Access regulations are specified as logical rules, where some predicates are explicitly identified. Attribute certificates are modelled as *credential expressions* of the form `credential_name(attribute_name`$_1$ `= value_term`$_1$`)`, …, `attribute_name`$_n$ `= value_ term`$_n$, where `credential_name` is the attribute credential name, `attribute_name`$_i$ is the attribute name, and `value_term`$_i$ is either a ground value or a variable. Besides credentials, the proposal also allows to reason about declarations (i.e., unsigned statements) and user-profiles that the server can maintain and exploit for taking the access decision. Communication of requisites to be satisfied by the requester is based on a filtering and renaming process applied on the server's policy, which exploits partial evaluation techniques in logic programs.

Although attribute-based access control polices allow the specifications of access control rules with reference to generic attributes or properties of the involved parties, they do not fully exploit the semantic power and reasoning capabilities of emerging web applications. The next step in the development of expressive and powerful access control models and policies should then be the support of access control rules based on the rich ontology-based metadata associated with both the subjects accessing the resources and the resources themselves (Damiani et al., 2004).

5 Conclusions

Access control models, policies, and languages are constantly under development to obtain frameworks flexible and expressive enough so as to handle the specification and enforcement of security requirements of many emerging applications and real-world scenarios. In this paper, we presented the main features that modern access control models and policies should support and discussed recent proposals in the area of access control languages.

References

Bettini, C., Jajodia, S., Wang, S. and Wijesekera, D. (2002) 'Provisions and obligations in policy rule management and security applications', *Proc. 28th International Conference on Very Large Data Bases*, Hong Kong, China, August, pp.351–372.

Blaze, M., Feigenbaum, J. and Lacy, J. (1996) 'Decentralized trust management', *Proc. 1996 IEEE Symposiumon Security and Privacy*, Oakland, CA, USA, May, pp.164–173.

Blaze, M., Feigenbaum, J., Ioannidis, J. and Keromytis, A. (1998) 'The role of trust management in distributed systems security', *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*, Springer-Verlag, LNCS State-of-the- Art series, pp.185–210.

Bonatti, P. and Samarati, P. (2002) 'A unified framework for regulating access and information release on the web', *Journal of Computer Security*, Vol. 10, No. 3, pp.241–272.

Chu, Y-H., Feigenbaum, J., LaMacchia, B., Resnick, P. and Strauss, M. (1997) 'Referee: trust management for web applications', *World Wide Web Journal*, Vol. 2, No. 3, pp.706–734.

Damiani, E., De Capitani di Vimercati, S., Fugazza, C. and Samarati, P. (2004) 'Extending policy languages to the semantic web', *Proc. International Conference on Web Engineering*, Munich, Germany, July, pp.330–343.

Date, C. (1995) *An Introduction to Database Systems*, 6th ed., Addison-Wesley, Boston, MA, USA.

DeTreville, J. (2002) 'Binder, a logic-based security language', *Proc. 2001 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May, pp.105–113.

Dovier, A., Piazza, C., Pontelli, E. and Rossi, G. (2000) 'Sets and constraints logic programming', *ACM Transactions of Programming Languages and Systems*, Vol. 22, No. 5, September, pp.861–931.

Farrell, S. and Housley, R. (2002) 'An internet attribute certificate profile for authorization', *RFC 3281*, April.

Fernandez, E., Gudes, E. and Song, H. (1994) 'A model for evaluation and administration of security in object-oriented databases', *IEEE Transaction on Knowledge and Data Engineering*, Vol. 6, No. 2, pp.275–292.

Gelfond, M. and Lifschitz, V. (1988) 'The stable model semantics for logic programming', *Proc. 5th International Conference and Symposium on Logic Programming*, The MIT Press, Cambridge, Massachusetts, pp.1070–1080.

Jajodia, S., Kudo, M. and Subrahmanian, V. (2001a) 'Provisional authorizations', in Ghosh, A. (Ed.): *E-Commerce Security and Privacy*, Kluwer Academic Publishers, Boston, pp.133–159.

Jajodia, S., Samarati, P., Sapino, M. and Subrahmanian, V. (2001b) 'Flexible support for multiple access control policies', *ACM Transactions on Database Systems*, Vol. 26, No. 2, June, pp.214–260.

Jim, T. (2001) 'SD3: a trust management system with certified evaluation', *Proc. 2001 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May, pp.106–115.

Landwehr, C. (1981) 'Formal models for computer security', *ACM Computing Surveys*, Vol. 13, No. 3, pp.247–278.

Li, N. and Mitchell, J. (2003) 'Datalog with constraints: a foundation for trust-management languages', *Proc. Fifth International Symposium on Practical Aspects of Declarative Languages (PADL 2003)*, New Orleans, LA, USA, January, pp.58–73.

Li, N., Mitchell, J. and Winsborough, W. (2002) 'Design of a role-based trust-management framework', *Proc. IEEE Symposium on Security and Privacy*, May, Oakland, CA, USA, pp.114–130.

Li, N., Grosof, B. and Feigenbaum, J. (2003) 'Delegation logic: a logic-based approach to distributed authorization', *ACM Transactions on Information and System Security*, Vol. 6, No. 1, February, pp.128–171.

Lunt, T. (1988) 'Access control policies: some unanswered questions', *IEEE Computer Security Foundations Workshop II*, Franconia, NH, June, pp.227–245.

Rabitti, F., Bertino, E., Kim, W. and Woelk, D. (1991) 'A model of authorization for next-generation database systems', *ACM TODS*, Vol. 16, No. 1, March, pp.89–131.

Samarati, P. and De Capitani di Vimercati, S. (2001) 'Access control: policies, models, and mechanisms', in Focardi, R. and Gorrieri, R. (Eds.): *Foundations of Security Analysis and Design*, LNCS 2171, Springer-Verlag, Berlin, Heidelberg, New York, pp.137–196.

Shen, H. and Dewan, P. (1992) 'Access control for collaborative environments', *Proc. Int. Conf. on Computer Supported Cooperative Work*, Toronto, Ontario, Canada, November, pp.51–58.

Wang, L., Wijesekera, D. and Jajodia, S. (2004) 'A logic-based framework for attribute based access control', *Proc. 2004 ACM Workshop on Formal Methods in Security Engineering*, October, Washington DC, USA.

Wijesekera, D. and Jajodia, S. (2003) 'A propositional policy algebra for access control', *ACM Transactions on Information and System Security*, Vol. 6, No. 2, May, pp.286–325.

Woo, T. and Lam, S. (1993) 'Authorization in distributes system: a new approach', *Journal of Computer Security*, Vol. 2, Nos. 2–3, pp.107–136.

Notes

[1] In an open policy scenario, the dual example of all users, but a few, who have to be denied an access can be considered.

[2] Note that this requires the use of an additional predicate (over) to evaluate whether an authorisation specified for subject s' can propagate to a subgroup s along any of the paths connecting them.

Website

Apache http server version 2.0, http://www.apache.org/docs-2.0/misc/tutorials.html.