A Logical Framework for Reasoning about Access Control Models

ELISA BERTINO
DSI, Università di Milano
BARBARA CATANIA
DISI, Università di Genova
ELENA FERRARI
DSCFM, Università dell'Insubria
and
PAOLO PERLASCA
DSI, Università di Milano

The increased awareness of the importance of data protection has made access control a relevant component of current data management systems. Moreover, emerging applications and data models call for flexible and expressive access control models. This has led to an extensive research activity that has resulted in the definition of a variety of access control models that differ greatly with respect to the access control policies they support. Thus, the need arises for developing tools for reasoning about the characteristics of these models. These tools should support users in the tasks of model specification, analysis of model properties, and authorization management. For example, they must be able to identify inconsistencies in the model specification and must support the administrator in comparing the expressive power of different models. In this paper, we make a first step in this direction by proposing a formal framework for reasoning about access control models. The framework we propose is based on a logical formalism and is general enough to model discretionary, mandatory, and role-based access control models. Each instance of the proposed framework corresponds to a C-Datalog program, interpreted according to a stable model semantics. In the paper, besides giving the syntax and the formal semantics of our framework, we show some examples of its application. Additionally, we present a number of dimensions along which access control models can be analyzed and compared. For each dimension, we show decidability results and we present some examples of its application.

Categories and Subject Descriptors: H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*

Author's addresses: E. Bertino and P. Perlasca, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, via Comelico 39/41, 20135 Milano, Italy; email: {bertino, perlasca}@dsi.unimi.it; B. Catania, Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova, via Dodecaneso 35, 16146 Genova, Italy; email: catania@disi.unige.it; E. Ferrari, Dipartimento di Scienze Chimiche, Fisiche e Matematiche, Università dell'Insubria, via Valleggio 11, 22100 Como, Italy; email: Elena.Ferrari@uninsubria.it.

General Terms: Security

Additional Key Words and Phrases: Access control framework, access control models analysis, logic programming

1. INTRODUCTION

Access control is usually performed against a set of authorizations stated by security administrators or users according to some security policies [Castano et al. 1995]. An authorization in general is specified on the basis of three parameters $\langle o, s, p \rangle$. This triple specifies that subject s is authorized to exercise privilege p on object o. Access control models based on the $\langle o, s, p \rangle$ paradigm are appropriate for traditional database environments and applications but are not adequate to meet the requirements of new database models and emerging applications that call for more flexible and expressive access control models. Over the last decade this need has resulted in various extensions to traditional data access control models that can be classified into three main categories:

— Extensions of access control models used in conventional relational database systems. Recent research proposals have extended the capabilities of these models with a variety of features, such as negative authorizations [Bertino et al. 1997], role-based and task-based authorizations [Sandhu et al. 1996], and temporal authorizations [Bertino et al. 1998]. Moreover, *multipolicy authorization models* [Bertino et al. 2000] have been proposed, that is, models able to support a variety of access control policies.

— Development of access control models for advanced DBMSs, like object-oriented [Fernandez et al. 1994; Millen and Lunt 1992; Rabitti et al. 1991], object-relational, and active DBMSs [Haas et al. 1990]. These DBMSs are characterized by data models that are richer than the relational model. Therefore, authorization models developed for relational DBMSs must be properly extended to deal with the additional modeling concepts contained in such advanced data models. For instance, in object-oriented data models, classes can be organized in hierarchies and can contain not only data but also methods performing operations on data. All these features must be taken into account when defining an access control model for object-oriented databases.

— Development of access control models for advanced applications, such as applications in the context of the World Wide Web (WWW) [Samarati et al. 1996; Winslett et al. 1997], Digital Libraries (DLs) [Adam et al. 2002], Workflow Management Systems (WFMSs) [Atluri and Huang 2000; Bertino et al. 1999; Thomas and Sandhu 1997], and Data Warehousing Systems. These advanced application environments are characterized by new access control requirements, such as for instance the support for a flexible subject qualification, authorization constraints, content-dependent access control, and new access privileges. All these requirements cannot be adequately supported by current access control mechanisms, which are tailored to few, specific policies. In most cases, either the organization is forced to adopt the specific policy

built-in into the access control mechanism at hand, or access control policies must be implemented as application programs. Both situations are clearly unacceptable.

These research efforts have resulted in the definition of a variety of access control models, that greatly differ with respect to their basic components and the access control policies they support. Thus, an important issue is how to evaluate and compare the *expressive power* of all those models. We believe that this is a crucial requirement since sometimes it is difficult to understand if new requirements and applications really need the definition of another access control model or the models already defined could be enough. The term expressivity is a broad term which may have different meanings and may take into account different aspects of an access control model. In this paper, along the lines pointed out by Sandhu [1992b], by expressive power we mean the ability of the models to support a variety of access control policies. In other words, the expressivity of an access control model is a measure of the range of policies it can support.

Based on the previous considerations, we believe that there is a strong need for a framework that makes it possible to reason about the expressive power and the features of access control models and to compare them on a formal basis. The framework must be flexible enough to accommodate both traditional and advanced access control models. The goal of this paper is thus not to present yet another access control model but to make a step towards the development of a formal foundation for access control models. We propose a logical framework able to model discretionary, mandatory, and role-based access control models. Each instance of the framework is a logical program, composed of C-Datalog rules [Greco et al. 1992], representing a specific instance of an access control model, for a given application domain. The framework is flexible enough to represent objects, subjects, privileges, possibly organized into hierarchies, and sessions, as well as positive and negative authorizations. The framework allows one to specify both explicit and derived authorizations, that is, authorizations that can be inferred from those explicitly specified. Additionally, the framework allows the specification of constraints on basic components of the framework. An important and innovative characteristic of our framework is that it does not impose any specific conflict resolution policy to deal with the simultaneous presence of both a positive and a negative authorization for the same object, subject, and privilege. Rather, it supports the specification of arbitrary conflict resolution policies and provides a semantics that is parameterized with respect to such a policy.

In the paper, besides giving the syntax and the formal semantics of our framework, we demonstrate its applicability by showing how some access control models can be mapped onto it. Moreover, we define a set of dimensions for the analysis of access control models and for comparing their expressive power, and we show how the analysis along these dimensions can be performed using our framework. Additionally, we present decidability results for these dimensions. The proposed dimensions can be taken as the basis for developing advanced tools for the specification and the analysis of access control models. For

example, they must be able to identify inconsistencies in the model specification and must support the administrator in comparing the expressive power of the models.

The remainder of the paper is organized as follows. Section 2 surveys related work, whereas Section 3 provides a possible scenario for the proposed framework and analysis dimensions. Section 4 introduces the basic components of our framework, whereas Section 5 gives the formal semantics. In Section 6 we show some examples of the framework applicability. Section 7 presents the proposed dimensions for the analysis of access control models. Section 8 concludes the paper. Appendix A shows how a C-Datalog program can be transformed into an equivalent Datalog program, whereas formal proofs of the proposed results are reported in Appendix B.

2. RELATED WORK

Although no comprehensive solution to the problem of comparing and analyzing access control models has been proposed so far, recent researchers have recognized the importance of this issue and some proposals have been presented [Jaeger and Tidswell 2001; Jajodia et al. 1997; 2001; Koch et al. 2000; 2001]. Those proposals mainly differ in the kind of approach (graph-based vs. logical) they use for representing access control models, policies, and constraints. *Graph-based approaches* rely on the use of graph transformations [Rozenberg 1997], whereas *logical approaches* are based on the use of logic programming [Lloyd 1987]. In *graph-based approaches*, access control models are modeled through graphs whose state changes upon the application of graph transformations. Graph transformations are techniques for modeling the evolution of graph structures according to a graph grammar[1] and are used for modeling the basic components of access control models and constraints. In *logical approaches*, such components are expressed through logic programs. According to the semantics chosen for these programs, the set of authorizations entailed by a given access control model instance corresponds to a certain set of facts contained in the models of the constructed logic program. By representing access control models as logic programs, the problem of comparing and analyzing access control models is reduced to the problem of comparing and analyzing logic programs. The advantage of this equivalence is that one can exploit the theoretical results obtained in logic programming for reasoning about access control models.

Although the two approaches have almost the same expressive power, they are complementary with respect to the purpose of use. In particular, a logical approach provides a precise mathematical foundation for reasoning about access control models. However, logical programs are not intuitive to most users. A graph-based approach, by contrast, provides user-friendly notation and, as such, it could be used as an effective tool for representing logical programs. Therefore, we believe that the main usage of the graph representation is to help in the specification, design, and presentation of access control policies and

[1]A graph grammar can be viewed as a generalization of the formal language theory based on string grammars, and of the theory of rewriting terms using trees [Rozenberg 1997].

constraints, rather than as a pure computational model. By contrast, a logical representation allows one to use a declarative approach for representing access control models and for computing the entailed set of access authorizations. Moreover, a logical approach is better supported than a graph-based approach by a wide variety of techniques and environments for computation.

Several languages, tools, and applications exist for both approaches. Among the languages and tools based on a graph representation, we recall PROGRES [Ehrig et al. 1999; Schurr 1991], DACTL [Glauert et al. 1991], and AGG [Ehrig et al. 1999; AGG]. Those tools provide graph editors for managing graphs and interpreters for supporting graph transformations. In general, all those tools provide several functions such as the generation and the management of different graph views, of different sections of a given graph, and of different representations of a graph. The interpreter applies rules for graph transformations and, in most cases, the user can choose to activate, either interactively or automatically, these rules. For the logic programming approach, Prolog is the most widely used language and several Prolog implementations have been developed. We recall, among the others: Strawberry Prolog [STRAWBERRY PROLOG], a 32-bit Prolog compiler supporting object oriented programming; ECLiPSe [ECLiPSe] (ECLiPSe Common Logic Programming System), a Prolog based system whose aim is to serve as a platform for integrating various Logic Programming extensions, in particular Constraint Logic Programming (CLP); XSB [XSB], a Prolog based system that extends standard Prolog with an implementation of OLDT[2] and HiLog terms (higher-order programming in which predicate symbols can be variable or structured). All those tools provide interfaces towards C, C++, or Java environments, and various DBMSs, such as Oracle, through ODBC or JDBC. From the side of specific logic programming database systems we recall, among the others, CORAL [CORAL] developed at the University of Wisconsin-Madison.

Among the graph-based approaches to the specification of access control models, we recall the one by Koch, Mancini, and Parisi-Presicce [Koch et al. 2001]. The framework they propose is expressive enough to model classical access control policies—mandatory, discretionary, and role-based access control policies—as well as constraints on the basic components of access control models. However, the goal of this framework is different from ours in that its goal is mainly to provide a tool for policy integration and evolution. By contrast, the problem of analyzing and comparing the expressive power of existing access control models is not considered. In a previous work [Koch et al. 2000], Koch, Mancini and Parisi-Presicce provide a formalization of role-based access control models using graph transformations which allows one to specify and verify consistency requirements. The goal of that work is different from ours since the framework they propose is focused on role-based access control models only and, in particular, on the issues of administration and revocation of user-role assignments. By contrast, we do not focus on a specific access control

[2]OLDT is a resolution strategy that avoids redundant computation by remembering subcomputations and reusing their results to respond to later requests.

model, rather we propose a general framework for representing a large variety of access control models.

The use of graph models has also been proposed by Jaeger and Tidswell [2001] as an approach to simplify the specification and the verification of safety via constraints, that is, with expressions able to specify the safety requirements of any access control configuration. An access control configuration is safe if does not happen that unexpected rights are granted to subjects not possessing the requirements needed for exercising those rights. In Jaeger and Tidswell [2001], constraints are expressed by using a small set of operators on graph nodes. The constraint language is expressive enough to model the most widely-used classes of constraints, such as static and dynamic separation of duty. Some preliminary results on the complexity of safety verification are also presented, which demonstrate the applicability of the approach. However, this work does not address the problem of comparing the expressive power of access control models and analyzing their characteristics.

For the logical formalism approach, Jajodia et al. [1997; 2001] have proposed a logical language for specifying authorization rules and have shown how this language can express several discretionary access control policies. Programs that can be written in this language are a subset of stratified Datalog programs. Thus, each program generates a unique set of authorizations. Moreover, the set of predicates that can be used in those programs is fixed and specifically conceived to express traditional discretionary access control models. By contrast, in this paper we propose a more general formalism able to model a variety of authorization specifications without syntactic restrictions, like stratification. Each program can therefore generate more than one set of authorizations (one for each stable model of the program). Hence, we do not restrict ourselves to the consideration of programs having a unique model, like stratified Datalog programs. Rather we allow a multiplicity of models to be associated with a given program. The administrator can then choose one of those sets, depending on the security requirements of the considered application domain. Our framework supports the usage of user-defined predicates. The framework is thus able to accommodate new and emerging models without requiring any extension. Our framework is also different from the one by Jajodia et al. [1997; 2001] with respect to the adopted conflict resolution policy. In the framework by Jajodia et al., when a conflicting authorization is discarded, other authorizations that may have been derived from it are not necessarily discarded. From our point of view this approach is not always correct. Thus, in our framework, we compute derived authorizations after resolving conflicts. Additionally, we propose a set of dimensions for the analysis of access control models and show how our framework can be used to compare and analyze existing access control models. These aspects are not addressed by the work of Jajodia et al.

3. MOTIVATING EXAMPLE

Suppose that an organization plans to develop an access control system. To this purpose, the security administrator (hereafter denoted by SA) must identify

which data have to be protected, how data can be accessed, and who can access those data under which privileges. These activities can be performed in three main steps, described in the following.

I. During the first step, the SA must choose an access control policy. For example, the SA may decide to adopt a discretionary policy because of its flexibility.

II. During the second step, the SA must specify an access control model compliant with the chosen discretionary policy. To this purpose, the SA must specify: (i) relevant domain components (for example, groups and/or roles); (ii) a set of rules specifying how domain components are hierarchically organized (for example, role inheritance hierarchies); (iii) a set of rules specifying how conditional authorizations are automatically derived by the system, starting from the authorizations explicitly specified; (iv) a set of rules specifying the integrity constraints the generated authorizations must satisfy. If the chosen model supports both positive and negative authorizations, the SA must also specify a mechanism to deal with conflicts that can possibly arise in the set of entailed authorizations.

We may assume that, in the access control model specification process, the SA is assisted by a tool, providing a GUI for entering the model characteristics and translating the specified model in an internal formal (e.g., the logical specification). It is also reasonable to assume that a library of already-specified access control models exists in the system. Thus, the SA, besides the specification of a new access control model, can alternatively select an access control model already contained in the library.

After the access control model has been specified or selected, the SA may be interested in determining whether the library contains other access control models, supporting similar features with respect to the ones supported by the chosen model. For example, if the SA selects a hierarchical RBAC model supporting both positive and negative authorizations, he/she may be interested in determining which other RBAC models with similar characteristics exist in the library. Such models can differ for example with respect to how they propagate authorizations through the role hierarchy. After selecting a certain set of models, the SA, in order to make his/her final choice, can evaluate the expressive power of the chosen models, determining for example the relationships existing between the authorizations entailed by the selected models. The SA may also be interested in specifying some instance examples then analyzing how the sets of entailed authorizations, computed for the specified instances, are related.

III. Once the model has been specified, the SA can start the analysis of the specified model, to identify possible weaknesses and interesting properties. For example, he/she can check whether constraints are well defined, that is, they admit the existence of at least one model instance, or examine the dependencies existing among authorizations. Based on the analysis results, the model can be refined, for example adding or deleting some integrity constraints.

In order to illustrate an example of the proposed scenario, suppose that, during step I, the SA chooses a discretionary policy. To support the chosen policy,

during step II, the SA may specify a hierarchical RBAC model, called *Model*1 in the following, propagating authorizations from roles to users playing those roles and from roles to subroles. The library may contain other access models with similar characteristics. For example, suppose that the library contains a hierarchical RBAC model, called *Model*2, that allows the SA to specify for each role whether it is temporarily not usable by the users authorized to play it. Moreover, suppose that for each role and privilege it is possible to specify in which direction (upward or downward) the privilege has to be propagated through the role hierarchy. *Model*2 is shown to the SA, since it is structurally similar to the one he/she has specified. At this time, in order to make the right choice, the SA may be interested in comparing the expressive power of *Model*1 and *Model*2. In this case, the system could determine that for each instance of *Model*1 there exists an instance of *Model*2 entailing the same set of authorizations. Based on this result, the SA can for example select *Model*2, since it is more expressive than *Model*1. *Model*2 can then be analyzed during step III, in order to identify useful properties. For example, the SA may be interested in determining whether it is possible to generate some negative authorizations for a given role starting from a positive authorization for the same role. Based on the result of the analysis, *Model*2 can be refined, for example, inserting an integrity constraint specifying that this situation must be avoided.

With respect to the previous scenario, the aim of this paper is to present the formal foundations for developing a tool able to support all the above-mentioned tasks. In particular, as we will clarify in the following sections, the proposed logical framework can be used for the internal representation of access control models in a common and formal format, that makes possible the subsequent analysis phase. As far as analysis is concerned, we present in Section 7 two classes of dimensions, that can serve as a basis for developing analysis tools. In particular, we propose a set of *inter-model* properties (see Section 7.1) that serve to make a comparative evaluation of access control models and thus represent the basis to make the more appropriate choice for the considered environment. An important issue concerns devising appropriated criteria for such choice. Some relevant quantitative criteria for each choice concerns the cost of administration and authorization checking operations and the complexity of the mapping on a given authorization policy on a given authorization model. Examples of inter-model properties are related to the *structural equivalence/containment* of two models, that is, whether they are built from the same or a similar set of building blocks, independently from the set of authorizations they entail. Besides structural equivalence, another fundamental issue is *access equivalence/containment*, that is, whether two models entail the same set of authorizations. By evaluating both access and structural equivalence, the SA can for example discover that the library contains an access control model equivalent to the one he/she has defined, but able to express the security requirements in a more compact way (for instance, by specifying fewer authorizations) or in a way that makes the management of administrative operations more efficient.

By contrast, for the analysis of a single access control model we propose the use of *intra-model* properties (see Section 7.2). In particular, we have devised two main intra-model properties: *reachability*, which allows the SA to identify

whether an authorization can be derived from a given access control model, and can thus be the basis for safety verification, and *consistency*, which allows the SA to determine whether the specified constraints can be satisfied, or whether they specify inconsistent conditions.

4. THE FORMAL FRAMEWORK

In the following, we first introduce the basic elements of the framework. Then, we introduce a logical language by which these basic components can be formally represented. Finally, we show how the introduced components can be described by the proposed logical language.

4.1 The Basic Components

The framework we propose has been developed with the goal of being as general as possible. Therefore, it contains a set of primitive "building blocks" upon which all other necessary concepts can be constructed. Then, based on the model to be represented, some of these building blocks are selected and composed together.

The framework supports the representation of four basic components—subjects, objects, privileges, and sessions—and the representation of arbitrary authorization rules. Additionally, the framework supports the specification of constraints on basic components of the framework. Subjects, objects, privileges, and sessions are characterized by a variable number of *attributes*. Those attributes model properties that are relevant in the specification of access control policies. Examples of properties are the name of a subject or an object access class. The various components of our framework are presented below.

Subjects. Authorizations are granted to *subjects*. A subject can be either a *user*, a *process*, a *group*, or a *role*. A user is a human being for whom authorizations must be expressed. Groups consist of sets of users and can be hierarchically organized into a group-subgroup hierarchy, according to a partially ordered relationship (denoted by $\prec_G$). Roles represent functions within a given organization. Roles can be hierarchically organized into a role-subrole hierarchy according to a partially ordered relationship (denoted by $\prec_R$). A process is the execution of a program on behalf of a specific user.

Objects. *Objects* are the resources to be protected. Objects can be hierarchically organized into a part-of hierarchy according to a partially-ordered relationship (denoted by $\prec_O$).

Privileges. *Privileges* represent the access modes subjects can exercise on the objects in the system. Some interactions may exist between privileges, specifying that a privilege is stronger than some others. For this reason, privileges can also be hierarchically organized into a privilege hierarchy according to a partially ordered relationship (denoted by $\prec_P$).

Sessions. A *Session* is a particular instance of a connection of a user to the system.

Authorization rules. Authorizations on the basic components can be specified through *authorization rules*. Authorization rules can exploit subjects,

objects, privileges and sessions attributes for the derivation of either positive or negative authorizations. A positive authorization establishes that a subject is authorized for a given privilege on a given object, whereas a negative authorization establishes that a subject is denied access to a given object under a given privilege. The specification of authorization rules can rely on the usage of user-defined predicates.

Constraint rules. Constraints on the components of the system can be specified through *constraint rules*. In general, constraint rules specify conditions that cannot be violated by the components of the system. Constraints can be classified into static and dynamic: static constraints can be checked statically, that is, without taking into account the execution state of the system, whereas dynamic constraints can be checked only by taking into account the execution state of the system. Constraints can involve subjects, objects, privileges, sessions and their hierarchical organization.

To formally represent the building blocks of our framework, we use C-Datalog [Greco et al. 1992], which is an object-oriented extension of Datalog, supporting all features required to model subjects, objects, privileges, sessions, their associated attributes, and authorization and constraint rules. More precisely, C-Datalog is an extension of Datalog [Ullman 1989] that includes a number of object-oriented constructs. In particular, it provides a framework for representing both classical object-oriented concepts, such as classes, objects, inheritance, that we use to represent subjects, objects, privileges, and sessions, as well as typical logic-based concepts, such as deductive rules, used to represent authorization and constraint rules. Object-oriented concepts like classes, objects, and inheritance provide the suitable background needed for fully characterizing each basic component of our framework: class attributes model relevant properties for the specification of access control policies whereas inheritance between classes allows classes to be defined in terms of other classes. This feature provides the ability to specialize the behaviour of classes starting from a set of common elements provided by their superclasses. For instance, a user, a process, a group, and a role can be defined as subclasses of a common superclass subject. On the other hand, the usage of a superclass "subject" allows us to easily refer to any subject in the specification of authorizations.

4.2 A Brief Introduction to C-Datalog

In C-Datalog, logical rules are defined against a given *C-Datalog schema*, representing the structure of the objects existing in the system. In the following, we briefly describe the basic concepts of the C-Datalog data model, then we present the C-Datalog Language.

C-Datalog data model. The C-Datalog data model is based on the following concepts.

(1) **Class and relation names**. The model provides the representation of two different sets of entities: (i) *class names* (denoted by K), representing

names of classes in the usual object-oriented terminology, and (ii) *derived relation names* (also called *predicates* in what follows) (denoted by R), representing predicates defined by logical rules. In the following, we denote $K \cup R$ with B. Moreover, the following assumptions hold: (i) $K \cap R = \emptyset$, that is, class and relation names must be disjoint, and (ii) $\{\perp, string, int\} \subseteq K$, that is, we assume that some built-in class names exist.

(2) **Class schema**. A function *Scheme*: $B \rightarrow ES$ assigns a schema to class names, where ES is a set of entity schemas. An entity schema is a finite subset of elements called *attributes* of the form $(name : type)$, where *name* is an attribute name (or label), belonging to an attribute name set A, and $type \in K$. We assume that no entity schema exists having two attributes with the same name. Function *Scheme* must satisfy the following (obvious) conditions: (i) $\forall k \in K, (self : k) \subseteq Scheme(k)$; (ii) $\forall r \in R, (self : r) \not\subset Scheme(r)$; and (iii) $\forall k \in \{\perp, string, int\}, Scheme(k) = \{(self : k)\}$.

(3) **Inheritance**. To specify inheritance between classes, a function $ISA : K \rightarrow 2^K$ is defined, such that $\forall c \in K, ISA(c) = \{k \mid k$ is a direct superclass of $c\}$. Function *Scheme* can be closed with respect to the class hierarchy, obtaining a function $Scheme^*$. $\forall k \in K$, $Scheme^*(k)$ returns the attributes of k and of all its superclasses. Conditions are given to deal with attributes with same name belonging to different superclasses (we do not consider this case in the paper). Note that $\{int,string\} \subseteq ISA(\perp)$ is always valid.

(4) **Object identifiers**. We assume the existence of a set Z of object identifiers (or oids). In the following, object identifiers of non built-in classes are denoted by $\#i$, where i is an integer number.

(5) **Instances**. An instance of an entity $b \in B$ can be defined as a tuple T on the entity schema H of class b, that is, as a subset of $A \times Z$ such that: $\forall (a : k) \in H, \exists! (a : z) \in T$ with $z \in Z^k$. Note that if $b \in K$, instances of b are all the possible objects of class b, in object-oriented terminology. On the other hand, if $b \in R$, instances of b are tuples, in the usual logic programming sense.

Based on the previous notions, a *CD (C-Datalog) Schema* S is defined as a tuple $< B, Scheme, A, ISA, Z >$. A CD Schema must satisfy the usual conditions of object-oriented schemas. For example, the instances of a class must also be instances of all its superclasses, oids are unique, and so on.

C-Datalog language. Given a CD Schema $S =< B, Scheme, A, ISA, Z >$, the C-Datalog logical language is used to define the extensions of the entities of S and to provide query facilities. The language is composed of: predicate symbols, coinciding with set B, constants symbols, coinciding with set Z, argument label symbols, coinciding with set A, and a set of variable names V.

A *CD atom* has the form $p(T)$ where: (i) $p \in B, Scheme^*(p) = \{a_1 : k_1 \ldots, a_n : k_n\}$; (ii) $T = \{a_1 : t_1 \ldots, a_n : t_n\}$, such that for each $(a_j : t_j)$, $j = 1, \ldots, n$, k_j is the type of t_j and one of the following conditions holds: (h) $t_j \in Z$, (hh) $t_j \in V$, (hhh) t_j is an atom. If $t_j \in Z$ or $t_j \in V$, then $(a_j : t_j)$ is a *simple term* otherwise it is a *class term*. Moreover, if $p \in K$, then $p(T)$ is a *class atom* otherwise it is

a *derived relation atom*. Let t_1, t_2 be either constants or variables, then $t_1 \theta t_2$, with $\theta \in \{=, <, >, \leq, \geq, \neq\}$ is a comparison atom. A *CD literal* is an atom $p(T)$ or the negation of an atom *not* $p(T)$.

A *CD rule* l has the form: $H \leftarrow A_1, \ldots, A_n$, where H is a CD atom whose arguments are simple terms, whereas $A_1, \ldots, A_n$ are CD literals or comparison atoms. H is the head of l denoted by *head*(l), whereas $A_1, \ldots, A_n$ is the body of l, denoted by *body*(l). Let l be a CD rule, if *body* (l) $= \emptyset$, then l is a fact; a *ground fact* is a fact with no variables. The variables appearing in the head of a rule must appear also in its body in order to assure typing and referential integrity.

A *CD program* P for S is a set of ground facts for each class name $c \in K$ and a set of CD rules for each relation name $r \in R$.

4.3 Representation of the Basic Components in C-Datalog

Based on the C-Datalog language, in the following we introduce the concept of *Access Control Model Schema* (ACMS for short) and *Access Control Model Instance* (ACMI for short), used to represent an authorization model inside the proposed framework. Informally, an ACMS defines the structural components on which the model is based, whereas an ACMI provides information concerning the component instances, that is, the actual subjects, objects, privileges, sessions, and the authorization and constraint rules used to instantiate the model.

Access Control Model Schema. To represent in C-Datalog the basic components introduced in Section 4.1, we need first to define a CD schema, able to model the following components:

—*Domain component (DC)*. Domain classes represent the structure of basic components of our framework (subjects, objects, privileges, and sessions) and domain instances represent the actual subjects, objects, privileges, and sessions. Instances are represented as a set of facts, which form the *domain component*.

—*Domain structure component (DSC)*. Domain structure information represents relationships existing among the basic components, for example parent-child relationships in basic component hierarchies. This information is expressed through a set of derived relation rules, representing the *Domain Structure Component*. The presence of these kinds of rules depends on the existence of a hierarchical organization of the class instances and/or of their attributes.

—*Authorization component (AC)*. This component contains authorization rules, expressed as a set of derived relation rules.

—*Propagation component (PC)*. This component consists of derived relation rules, called in what follows *propagation rules*, by which additional authorizations can be derived, starting from authorization rules and domain information. For example, a typical propagation rule specifies that an authorization which holds for an object o also holds for all the sub-objects of o. The presence of these kinds of rules depends on the existence of a hierarchical

organization of class instances. Propagated authorizations also maintain information on the *authorization source*, that is, the authorization that has triggered the propagation process. Information on the source of a propagation can be useful for the resolution of conflicts between positive and negative authorizations (see Section 5).

— *Constraint component (CC).* The constraint component is composed of derived relation rules able to express static and dynamic constraints on the basic components.

An ACMS can then be formally defined as follows.

Definition 1 (*Access Control Model Schema*). An Access Control Model Schema is a CD Schema $< B, Scheme, A, ISA, Z >$ such that:

(1) $B = K \cup R$ such that: (i) $K = K_{built\_in} \cup K_{basic}$, where $K_{built\_in} = \{\perp, int, string\}$, and $K_{basic} \subseteq \{user, group, role, process, subject, object, privilege, session\}$. We assume that K_{basic} always contains the class names "subject", "object", "privilege" and at least one class name from those in $\{user, group, role, process\}$; (ii) $R = R_{domain} \cup R_{auth} \cup R_{constraint} \cup R_{user\_def}$. In particular, $R_{domain} \subseteq \{SubG, InSubG, Belong, UserIn, LessR, InLessR, Play, UserPlay, PartOf, InPartOf, LessP, InLessP, ActiveRole\}$, $R_{auth} = \{Auth_d, Auth_p, Auth\}$, $R_{constraint} \subseteq \{ErrorC\}$, and R_{user_def} is a set of user-defined derived relation names.
The aim of derived relation names in R_{domain} is to express the organization of the instances of each class in K and their structural relationships. K and R_{domain} are related as follows:
— if $SubG \in R_{domain}$, then $group \in K_{basic}$ and groups must be hierarchically organized ($InSubG \in R_{domain}$);
— if $Belong \in R_{domain}$, then $\{user, group\} \subseteq K_{basic}$;
— if $UserIn \in R_{domain}$, then $\{user, group\} \subseteq K_{basic}$, $Belong \in R_{domain}$ and groups must be hierarchically organized ($InSubG \in R_{domain}$);
— if $LessR \in R_{domain}$, then $role \in K_{basic}$ and roles must be hierarchically organized ($InLessR \in R_{domain}$);
— if $Play \in R_{domain}$, then $\{user, role\} \subseteq K_{basic}$;
— if $UserPlay \in R_{domain}$, then $\{user, role\} \subseteq K_{basic}$, $Play \in R_{domain}$ and roles must be hierarchically organized ($InLessR \in R_{domain}$);
— if $PartOf \in R_{domain}$, then $object \in K_{basic}$ and objects must be hierarchically organized ($InPartOf \in R_{domain}$);
— if $LessP \in R_{domain}$, then $privilege \in K_{basic}$ and privileges must be hierarchically organized ($InLessP \in R_{domain}$);
— if $ActiveRole \in R_{domain}$, then $\{role, user, session\} \subseteq K_{basic}$.
$R_{constraint}$ contains a derived relation name (that is, $ErrorC$) used to define constraint rules, whereas R_{auth} is composed of three derived relation names used to define authorizations: $Auth_d$ is used to define direct authorizations, that is, authorizations that are not derived through propagation rules, $Auth_p$ is used to define propagated authorizations, and $Auth$ to refer indiscriminately to direct and propagated authorizations. R_{user_def} is a set of user-defined relation names, whose meaning depends on the context.

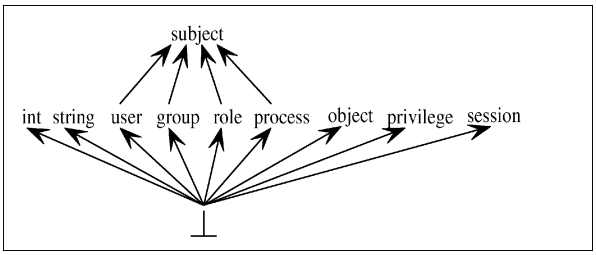


Fig. 1. The ISA hierarchy.

(2) Function *Scheme*, for derived relation names, is defined in the fourth column of Table I.
(3) Function *ISA*, for non built-in class names, is defined as follows: $ISA(user) = ISA(group) = ISA(role) = ISA(process) = \{subject\}$ (see Figure 1). For all the other class names c, $ISA(c) = \emptyset$.
(4) Attribute name set A contains the special name "self" and the attribute names appearing in the schema of each entity name $b \in B$ (see Table I).
(5) A set Z consisting of oids for class entity names.

Table I contains information on the derived relation names composing an ACMS (see Definition 1). In particular, the first column groups predicates according to their type. The second column reports the predicate names. The third column lists constraints that predicates must satisfy whereas the fourth column presents predicate schemas. Finally, the fifth column explains the predicate's meaning.

Access Control Model Instance. An *Access Control Model Instance* is a CD Program constructed over an ACMS, satisfying specific conditions. This program contains definitions for domain classes and instances, domain structure component, authorization component, propagation component, and constraint component. Table II presents rules that can be included in the domain component, depending on the considered schema, Table III presents examples of rules that can be contained in the authorization and propagation components, whereas Table IV presents some examples of rules that can be included in the constraint component, modelling some of the most widely used classes of constraints [Jaeger and Tidswell 2001].

An ACMI can model both *direct* and *propagated* authorizations. Direct authorizations are directly defined by the administrator, without taking into account domain hierarchies. For example, "(A1) Ann is authorized to read document $d1$", is an example of direct authorization. Direct authorizations can in turn be *unconditional* or *conditional*. Authorization A1 is an example of unconditional authorization. By contrast, "(A2) Ann is authorized by Mary to write doc1 if Bob has not been authorized by Mary to write the same document" is an example of a conditional authorization because the authorization for Ann holds providing that another authorization for Bob does not hold. From a C-Datalog point of

Table I. Predicates and their Meaning

Category	Predicate	Constraint	Function Scheme	Meaning
Group	$SubG$	$group \in K_{basic}$ and hierarchical organization of groups	$SubG(G_1:group, G_2:group)$	it represents the child-father relationship in the $\prec_G$ hierarchy; group G_1 is a direct child of G_2
	$InSubG$	$SubG \in R_{domain}$	$InSubG(G_1:group, G_2:group)$	it represents the transitive closure of $SubG$; group G_1 is an indirect child of G_2
	$Belong$	$\{user, group\} \subseteq K_{basic}$	$Belong(U:user, G:group)$	it represents the membership of a user U to a group G
	$UserIn$	$Belong \in R_{domain}$ and hierarchical organization of groups	$UserIn(U:user, G:group)$	it represents the transitive closure of $Belong$; user U is an indirect member of group G
Role	$LessR$	$role \in K_{basic}$ and hierarchical organization of roles	$LessR(R_1:role, R_2:role)$	it represents the child-father relationship in the $\prec_R$ hierarchy; role R_1 is a direct child of R_2
	$InLessR$	$LessR \in R_{domain}$	$InLessR(R_1:role, R_2:role)$	it represents the transitive closure of $LessR$; role R_1 is an indirect child of R_2
	$Play$	$\{user, role\} \subseteq K_{basic}$	$Play(U:user, R:role)$	it represents the authorization of a user U to play a role R
	$UserPlay$	$Play \in R_{domain}$ and hierarchical organization of roles	$UserPlay(U:user, R:role)$	it represents the transitive closure of $Play$
	$ActiveRole$	$\{user, role, session\} \subseteq K_{basic}$	$ActiveRole(U:user, S:session, R:role)$	it represents a role R activated by a user U in a session S
Object	$PartOf$	$object \in K_{basic}$ and hierarchical organization of objects	$PartOf(O_1:object, O_2:subject)$	it represents the child-father relationship in the $\prec_O$ hierarchy; object O_1 is a direct child of O_2
	$InPartOf$	$PartOf \in R_{domain}$	$InPartOf(O_1:object, O_2:subject)$	it represents the transitive closure of $PartOf$; object O_1 is an indirect child of O_2
Privilege	$LessP$	$privilege \in K_{basic}$ and hierarchical organization of privileges	$LessP(P_1:privilege, P_2:privilege)$	it represents the child-father relationship in the $\prec_P$ hierarchy; privilege P_1 is a direct child of P_2
	$InLessP$	$LessP \in R_{domain}$	$InLessP(P_1:privilege, P_2:privilege)$	it represents the transitive closure of $LessP$; privilege P_1 is an indirect child of P_2
Auth.	$Auth_d$		$Auth_d(O:object, S:subject, P:privilege, G:subject, \epsilon:string)$	it represents the granting of a positive or negative direct authorization; subject S is authorized ($\epsilon = +$) or denied ($\epsilon = -$) for a given privilege P on a given object O by grantor G
	$Auth_p$		$Auth_p(O:object, S:subject, P:privilege, G:subject, \epsilon:string, O':object, S':subject, P':privilege)$	it represents the granting of a positive or negative propagated authorization, by considering the authorization source (object O', subject S' and privilege P')
	$Auth$		$Auth(O:object, S:subject, P:privilege, G:subject, \epsilon:string)$	it represents both direct and propagated authorizations
Constraint	$ErrorC$	existence of the elements involved in the constraint	ErrorC()	it represents the violation of a constraint
User-defined				arbitrary predicates

Table II. DSC rules and their Meaning

Category	Id	Rule	Meaning
Group	1a	$InSubG(G_1 : X, G_2 : Y) \leftarrow SubG(G_1 : X, G_2 : Y)$	it captures the direct child-father relation in the $\prec_G$ hierarchy
	1b	$InSubG(G_1 : X, G_2 : Y) \leftarrow SubG(G_1 : X, G_2 : Z), InSubG(G_1 : Z, G_2 : Y)$	it captures the indirect child-father relation in the $\prec_G$ hierarchy
	2a	$UserIn(U : X, G : Y) \leftarrow Belong(U : X, G : Y)$	it captures the direct membership of a user to a group
	2b	$UserIn(U : X, G : Y) \leftarrow Belong(U : X, G : Z), InSubG(G_1 : Z, G_2 : Y)$	it captures the indirect membership of a user to a group
Role	3a	$InLessR(R_1 : X, R_2 : Y) \leftarrow LessR(R_1 : X, R_2 : Y)$	it captures the direct child-father relation in the $\prec_R$ hierarchy
	3b	$InLessR(R_1 : X, R_2 : Y) \leftarrow LessR(R_1 : X, R_2 : Z), InLessR(R_1 : Z, R_2 : Y)$	it captures the indirect child-father relation in the $\prec_R$ hierarchy
	4a	$UserPlay(U : X, R : Y) \leftarrow Play(U : X, R : Y)$	it denotes the roles that a user is explicitly authorized to play
	4b	$UserPlay(U : X, R : Y) \leftarrow Play(U : X, R : Z), InLessR(R_1 : Y, R_2 : Z)$	it denotes the roles that a user is indirectly authorized to play due to the $\prec_R$ hierarchy
Object	5a	$InPartOf(O_1 : X, O_2 : Y) \leftarrow PartOf(O_1 : X, O_2 : Y)$	it captures the direct child-father relation in the $\prec_O$ hierarchy
	5b	$InPartOf(O_1 : X, O_2 : Y) \leftarrow PartOf(O_1 : X, O_2 : Z), InPartOf(O_1 : Z, O_2 : Y)$	it captures the indirect child-father relation in the $\prec_O$ hierarchy
Privilege	6a	$InLessP(P_1 : X, P_2 : Y) \leftarrow LessP(P_1 : X, P_2 : Y)$	it captures the direct child-father relation in the $\prec_P$ hierarchy
	6b	$InLessP(P_1 : X, P_2 : Y) \leftarrow LessP(P_1 : X, P_2 : Z), InLessP(P_1 : Z, P_2 : Y)$	it captures the indirect child-father relation in the $\prec_P$ hierarchy

view, direct authorizations are represented as C-Datalog facts or rules, having predicate $Auth_d$, where d stands for "direct", as head predicate.

On the other hand, propagated authorizations specify how direct authorizations are propagated through domain hierarchies. Propagation can be performed in several ways. For example, authorizations can be propagated from a group to all its subgroups. From a C-Datalog point of view, propagated authorizations can be specified through C-Datalog rules, having $Auth_p$, where p stands for "propagated", as head predicate.

It is important to note that, given a propagated authorization, it is always possible to identify the direct authorization from which the propagation originated. This information is called *propagation source* of the considered propagated authorization. Based on this definition, the propagation source for a direct

Table III. AC and PC Rules and their Meaning

Category	Id	Rule	Meaning
User-defined	1	rules with a user-defined predicate in the head and with predicates contained in $R_{domain} \cup R_{user\_def}$ in the body	the meaning depends on the context
Auth	2	rules with $head = Auth_d$ and with predicates contained in $R \setminus R_{constraint} \setminus \{Auth_p, Auth_d\}$ in the body	they express direct authorizations
Propagation	3	$Auth_p(O:X_1, S:X_2, P:X_3, G:X_4, \epsilon: X_5, O':X_6, S':X_7, P':X_8) \leftarrow Auth_p(O: X_1, S:X_9, P:X_3, G:X_4, \epsilon:X_5, O':X_6, S': X_7, P':X_8), InSubG(G_1:X_2, G_2:X_9)$	it propagates an authorization for a given group to its subgroups
	4	$Auth_p(O:X_1, S:X_2, P:X_3, G:X_4, \epsilon: X_5, O':X_6, S':X_7, P':X_8) \leftarrow Auth_p(O: X_1, S:X_9, P:X_3, G:X_4, \epsilon:X_5, O':X_6, S': X_7, P':X_8), UserIn(U:X_2, G:X_9)$	it propagates an authorization for a given group to users belonging to the group
	5	$Auth_p(O:X_1, S:X_2, P:X_3, G:X_4, \epsilon: +, O':X_5, S':X_6, P':X_7) \leftarrow Auth_p(O: X_1, S:X_8, P:X_3, G:X_4, \epsilon:+, O':X_5, S': X_6, P':X_7), InLessR(R_1:X_8, R_2:X_2)$	it propagates a positive authorization for a given role to roles that are more powerful
	6	$Auth_p(O:X_1, S:X_2, P:X_3, G:X_4, \epsilon: -, O':X_5, S':X_6, P':X_7) \leftarrow Auth_p(O: X_1, S:X_8, P:X_3, G:X_4, \epsilon:-, O':X_5, S': X_6, P':X_7), InLessR(R_1:X_2, R_2:X_8)$	it propagates a negative authorization for a given role to roles that are less powerful
	7	$Auth_p(O:X_1, S:X_2, P:X_3, G:X_4, \epsilon: X_5, O':X_1, S':X_6, P':X_3) \leftarrow Auth_p(O: X_1, S:X_6, P:X_3, G:X_4, \epsilon:X_5, O':X_1, S': X_6, P':X_3), UserPlay(U:X_2, R: X_6), ActiveRole(U:X_2, S:X_7, R:X_6)$	it propagates direct authorizations from each role to users that are authorized to play that role and have activated it;[3] it realizes the activation interpretation of the role hierarchy [Sandhu et al. 2000]
	8	$Auth_p(O:X_1, S:X_2, P:X_3, G:X_4, \epsilon: X_5, O':X_6, S':X_7, P':X_8) \leftarrow Auth_p(O: X_1, S:X_9, P:X_3, G:X_4, \epsilon:X_5, O':X_6, S': X_7, P':X_8), UserPlay(U:X_2, R: X_9), ActiveRole(U:X_2, S:X_{10}, R:X_9)$	it propagates both direct and propagated authorizations from each role to users that are authorized to play that role and have activated it;[4] together with rules 5 and 6 it realizes the inheritance interpretation of the role hierarchy [Sandhu et al. 2000]
	9	$Auth_p(O:X_1, S:X_2, P:X_3, G:X_4, \epsilon: X_5, O':X_6, S':X_7, P':X_8) \leftarrow Auth_p(O: X_9, S:X_2, P:X_3, G:X_4, \epsilon:X_5, O':X_6, S': X_7, P':X_8), InPartOf(O_1:X_1, O_2:X_9)$	it propagates an authorization for a given object to its subobjects
	10	$Auth_p(O:X_1, S:X_2, P:X_3, G:X_4, +, O': X_5, S':X_6, P':X_7) \leftarrow Auth_p(O:X_1, S: X_2, P:X_8, G:X_4, +, O':X_5, S':X_6, P': X_7), InLessP(P_1:X_3, P_2:X_8)$	it propagates a positive authorization for a given privilege to privileges that are less powerful
	11	$Auth_p(O:X_1, S:X_2, P:X_3, G:X_4, -, O': X_5, S':X_6, P':X_7) \leftarrow Auth_p(O:X_1, S: X_2, P:X_8, G:X_4, -, O':X_5, S':X_6, P': X_7), InLessP(P_1:X_8, P_2:X_3)$	it propagates a negative authorization for a given privilege to privileges that are more powerful
	12	$Auth_p(O:X_1, S:X_2, P:X_3, G:X_4, \epsilon: X_5, O':X_1, S':X_2, P':X_3) \leftarrow Auth_d(O: X_1, S:X_2, P:X_3, G:X_4, \epsilon:X_5)$	it adds source to direct authorizations
	13	$Auth(O:X_1, S:X_2, P:X_3, G:X_4, \epsilon:X_5) \leftarrow Auth_p(O:X_1, S:X_2, P:X_3, G:X_4, \epsilon: X_5, O':X_6, S':X_7, P':X_8)$	it removes source from authorizations
Constraint	14	non recursive rules with *ErrorC* in the head and with predicates contained in $R_{domain} \cup R_{auth} \cup R_{user\_def}$ in the body	they express constraints whose meaning depends on the context

Table IV. Examples of CC Rules and their Meaning

Rule	Meaning
$ErrorC() \leftarrow Belong(U : X, G : Y)$	user X cannot be a member of a group Y
$ErrorC() \leftarrow Belong(U : X, G : K), Belong(U : Y, G : K), Belong(U : Z, G : K)$	users X, Y, Z cannot belong to the same group K
$ErrorC() \leftarrow Play(U : X, R : Y)$	user X cannot play role Y
$ErrorC() \leftarrow Play(U : X, R : Y), Play(U : K, R : Y)$	users X, K cannot both play role Y
$ErrorC() \leftarrow Play(U : X, R : Y), Play(U : X, R : K)$	roles Y, K cannot be both assigned to the same user X
$ErrorC() \leftarrow ActiveRole(U : X, S : Y, R : Z), ActiveRole(U : X, S : Y, R : K)$	roles Z, K cannot be both activated by user X within the same session Y
$ErrorC() \leftarrow Auth(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8), Auth(O : X_1, S : X_2, P : X_9, G : X_{10}, \epsilon : X_{11}, O' : X_{12}, S' : X_{13}, P' : X_{14}), role(self : X_2), X_3 \neq X_9$	privileges X_3, X_9 on object X_1 cannot be simultaneously assigned to role X_2
$ErrorC() \leftarrow Auth(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8), Auth(O : X_1, S : X_2, P : X_9, G : X_{10}, \epsilon : X_{11}, O' : X_{12}, S' : X_{13}, P' : X_{14}), user(self : X_2), X_3 \neq X_9$	privileges X_3, X_9 on object X_1 cannot be simultaneously assigned to user X_2

authorization is the authorization itself. From a formal point of view, for each direct authorization $(O : o, S : s, P : p, \epsilon : k)$, we define its source as the triple (o, s, p), whereas for each propagated authorization $(O : o, S : s, P : p, \epsilon : k)$, we define its source as the triple (o', s', p'), such that the following conditions hold: (i) an authorization $(O' : o', S' : s', P' : p', \epsilon : k')$ holds, (ii) $(O : o, S : s, P : p, \epsilon : k)$ has been generated from (o', s', p'), and (iii) no authorization (o'', s'', p'') exists such that $(O' : o', S' : s', P' : p', \epsilon : k')$ has been generated from (o'', s'', p'').

As we will see in Section 5, propagation sources are useful in defining specific conflict resolution functions. For this reason, we assume that rules defining propagated authorizations also compute propagation sources. On the other hand, it is important to note that it could be useful to define conditional direct authorizations depending on some other direct and propagated authorizations. In this case, however, information concerning the propagation source is usually not relevant. For this reason, we introduce an auxiliary predicate *Auth*, representing both propagated and direct authorizations without taking into account authorization sources. The definition of this predicate can therefore be obtained from the definition of predicate $Auth_p$ by projecting out information concerning the source.

Table III presents the rules defining predicate *Auth* (rule 13) and some examples of rules defining predicate $Auth_p$ (rules $3 - 11$ and rule 12). In particular,

[3]Note that the subject of the propagated authorization in the body of the rule is equal to its source subject; this means that the considered propagated authorization has been generated from an explicitly given direct authorization by applying rule 12.

[4]Note that the subject of the propagated authorization in the body of the rule and its source subject are different; this means that the considered propagated authorization can be generated from a generic authorization, that is, from either a direct or a propagated authorization.

rule 12 specifies that a direct authorization is a special case of a propagated authorization, where the source coincides with the authorization itself. Such a rule is always contained in an ACMI since it describes the first step of the propagation process. As additional examples, rule 9 states that an authorization for a given object o is propagated to all the objects that are part of o. Moreover, if s is a group, rule 3 specifies that the authorization is directly propagated to all the subgroups of s, whereas rule 4 specifies that the authorization is propagated to all users that belong to s. By contrast, according to rules 5 and 6, if s is a role, the propagation varies according to the sign of the authorization: if the authorization is positive, it is propagated to all the roles preceding s in the role hierarchy, whereas if it is negative, it is propagated to all the subroles of s. Propagation may also involve privileges in the sense that a positive authorization for a privilege p may imply an analogous authorization for all the subprivileges of p, whereas a negative authorization for a privilege p may imply an analogous authorization for all the privileges preceding p in the privilege hierarchy. This type of propagation is stated by rules 10, 11. Rule 7 specifies the activation interpretation of the role hierarchy [Sandhu et al. 2000], whereas rule 8, together with rules 5 and 6, specifies the inheritance interpretation of the role hierarchy. In fact, rule 7 propagates direct authorizations from each role to users that are authorized to play that role and have activated it, whereas rule 8 propagates direct and propagated authorizations from each role to users that are authorized to play that role and have activated it. Note that the presence of rules $3-11$ depends on the presence of the predicates that define their bodies in the ACMS schema.

Based on the concepts defined above, an instance of our framework can be now defined as follows.

Definition 2 (*Access Control Model Instance*). Let $S =< B, Scheme, ISA, A, Z >$ be an ACMS. An *Access Control Model Instance* (shortly instance) $\mathcal{I}$ over S is a C-Datalog program composed of:

(1) *Domain component*: a definition (that is, a set of facts) for each class name $c \in K_{basic}$.
(2) *Domain structure component*: a definition (that is, a set of facts and/or rules) for each derived relation name $r \in R_{domain}$. The rules must be extracted from those presented in Table II.
(3) *Authorization component*: a definition (that is, a set of facts and rules) for the derived relation name $Auth_d$ and for derived relation names in R_{user_def}. The predicates contained in the body of the rules defining $Auth_d$ must belong to $R \setminus R_{constraint} \setminus \{Auth_p, Auth_d\}$. The predicates contained in the body of the rules defining predicates in R_{user_def} must belong to $R_{domain} \cup R_{user\_def}$.
(4) *Propagation component*: a definition for predicates $Auth_p$, composed of rule 12 and a set of negation-free recursive rules, containing in their body predicates belonging to $R_{domain} \cup \{Auth_p\}$; a definition for predicate $Auth$ obtained by extracting rule 13 from Table III.
(5) *Constraint component*: a non recursive definition for the derived relation name *ErrorC*. The predicates contained in the body of the rules defining *ErrorC* must belong to $R \setminus R_{constraint}$.

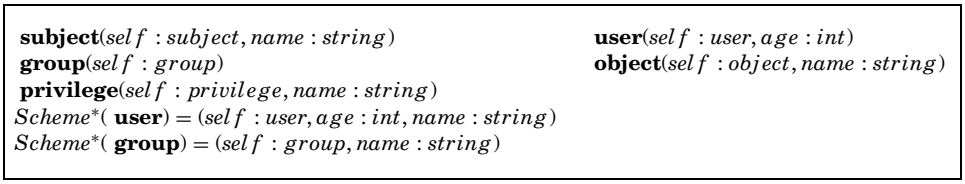


Fig. 2. Schema for classes of Example 1.

Let I be an ACMI. The extensional part of I, called Extensional Access Control Model Instance (EACMI), is composed of: (i) the facts in DC, (ii) the facts in DSC, (iii) the facts for predicate $Auth_d$, and (iv) the facts for user-defined predicates. We require the EACMI be composed of, at least, one subject, one object, and one privilege. The intensional part of I, called Intensional Access Control Model Instance (IACMI), is composed of all the rules in I.

Example 1. Consider an ACMS S defined as follows: (I) $B = K \cup R$, such that: (i) $K = K_{built\_in} \cup K_{basic}$, where $K_{built\_in} = \{\bot, int, string\}$, and $K_{basic} = \{user, group, subject, object, privilege\}$; (ii) $R = R_{domain} \cup R_{auth} \cup R_{constraint} \cup R_{user\_def}$, where $R_{domain} = \{SubG, InSubG, Belong, UserIn, PartOf, InPartOf, LessP, InLessP\}$, $R_{auth} = \{Auth_d, Auth_p, Auth\}$, $R_{constraint} = \{ErrorC\}$, and $R_{user\_def} = \emptyset$; (II) function Scheme for class names in K_{basic} and for derived relation names in $R_{domain} \cup R_{auth}$, presented in Figure 2 and in Table I, respectively; (III) function ISA, presented in Figure 3 (d); (IV) set A, containing the special name "self" and the attribute names appearing in Figure 2; (V) Z containing oids for the elements in K. The following is an instance of S:

(1) DC: it is composed of the following facts:[5]

$group(self : \#1, name : Employees)$
$group(self : \#2, name : Dev)$
$user(self : \#3, name : Ann, age : 30)$
$user(self : \#4, name : Bob, age : 32)$
$user(self : \#5, name : Mary, age : 27)$
$object(self : \#6, name : Pub)$
$object(self : \#7, name : Private)$
$object(self : \#8, name : doc1)$
$object(self : \#9, name : doc2)$
$privilege(self : \#10, name : Read)$
$privilege(self : \#11, name : Write)$

(2) DSC: it is composed of rules 1a, 1b, 2a, 2b, 5a, 5b, 6a, 6b of Table II and the following facts:[6]

$SubG(G_1 : \#2(Dev), G_2 : \#1(Employees))$
$Belong(U : \#3(Ann), G : \#1(Employees))$

[5]We remark that function $Scheme^*$ represents the closure of function $Scheme$ with respect to the class hierarchy and that CD atoms are defined on the basis of $Scheme^*$.
[6]In order to make the examples more readable, after any oid we insert the name of the element the oid refers to.

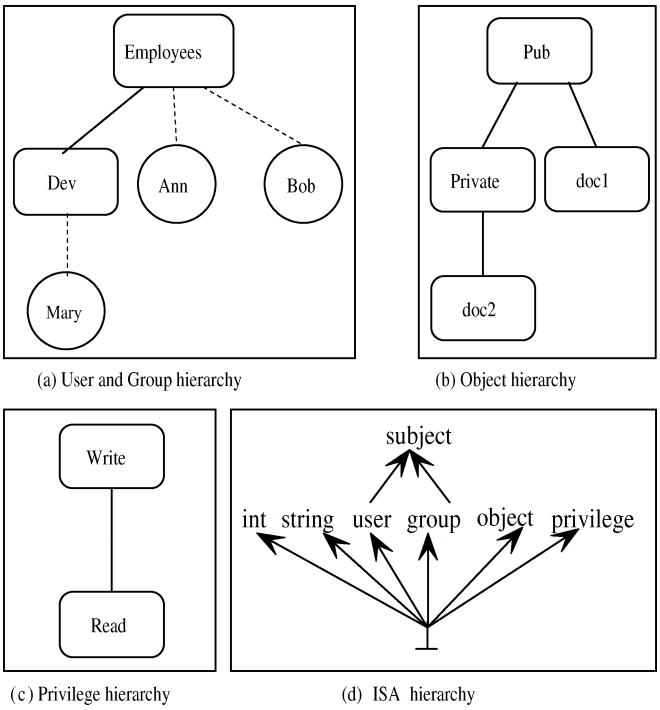


Fig. 3. Hierarchical organization for classes of Example 1.

$Auth_d(O : \#8(doc1), S : \#3(Ann), P : \#11(Write), G : \#5(Mary), \epsilon : +) \leftarrow not\ Auth(O : \#8(doc1), S : \#4(Bob), P : \#11(Write), G : \#5(Mary), \epsilon : +)$
$Auth_d(O : \#8(doc1), S : \#4(Bob), P : \#11(Write), G : \#5(Mary), \epsilon : +) \leftarrow not\ Auth(O : \#8(doc1), S : \#3(Ann), P : \#11(Write), G : \#5(Mary), \epsilon : +)$
$Auth_d(O : \#9(doc2), S : \#2(Dev), P : \#11(Write), G : \#4(Bob), \epsilon : +) \leftarrow$
$Auth_d(O : \#9(doc2), S : \#5(Mary), P : \#11(Write), G : \#3(Ann), \epsilon : -) \leftarrow Auth(O : \#9(doc2), S : \#2(Dev), P : \#11(Write), G : \#4(Bob), \epsilon : +)$

Fig. 4. Authorization rules for Example 1.

$Belong(U : \#4(Bob), G : \#1(Employees))$
$Belong(U : \#5(Mary), G : \#2(Dev))$
$PartOf(O_1 : \#7(Private), O2 : \#6(Pub))$
$PartOf(O_1 : \#8(doc1), O_2 : \#6(Pub))$
$PartOf(O_1 : \#9(doc2), O_2 : \#7(Private))$
$LessP(P_1 : \#10(Read), P_2 : \#11(Write))$

Subject, object, and privilege hierarchies are graphically represented in Figures 3 (a), (b), and (c). We use dashed lines to represent the membership of users to groups, to distinguish it from the group-subgroup relation (denoted with solid lines).

(3) AC: rules in Figure 4. The first rule states that Ann is authorized by Mary to write doc1 if Bob has not been authorized by Mary to write the same document. The second rule states that Bob is authorized by Mary to write

doc1 if Ann has not be authorized by Mary to write the same document. The third rule states that members of group Dev are authorized by Bob to write doc2, whereas the last rule states that Mary is denied by Ann to write doc2 if members of group Dev are authorized by Bob to write the same document.

(4) PC: rules 3-4 and 9-13 of Table III.

(5) CC: it contains the following rules:

$$ErrorC() \leftarrow Belong(U : \#3(Ann), G : \#2(Dev))$$
$$ErrorC() \leftarrow Belong(U : \#3(Ann), G : \#1(Employees))$$

whose meaning is to prevent Ann from being a member of groups Dev and Employees; the first constraint is satisfied whereas the second is violated since Ann belongs to Employees.

Since "user' and "group" are subclasses of "subject",[7] the following set of inherited elements holds:

$subject(self : \#1, name : Employees)$ $subject(self : \#2, name : Dev)$
$subject(self : \#3, name : Ann)$ $subject(self : \#4, name : Bob)$
$subject(self : \#5, name : Mary)$

5. SEMANTICS

In the following, we provide the formal semantics for an ACMI. It has been shown that any C-Datalog program can be transformed into an equivalent Datalog program with negation [Greco et al. 1992]. Given an ACMI $\mathcal{I}$, we denote with $D(\mathcal{I})$ the corresponding Datalog-like program, that we call *Access Control Model Program* (ACMP). The interested reader can find some information concerning this transformation in Appendix A. Moreover, given an ACMS $S =< B, Scheme, ISA, A, Z >$, we denote with $\mathcal{L}(S)$ the logical language over which program $D(\mathcal{I})$ is constructed, where $\mathcal{I}$ is an instance of S, and we call it *Access Control Model Language* (ACML). It is simple to prove that constants in $\mathcal{L}(S)$ coincide with Z and predicates coincide with B. In the following, when S is not relevant or it is clear from the context, we denote $\mathcal{L}(S)$ with $\mathcal{L}$.

The semantics we propose has to cope with conflicting authorizations. More precisely, a conflict arises when a positive and a negative authorization hold for the same subject, object, and privilege. Conflicts have to be solved to determine whether an access should be authorized or not. The proposed semantics supports a *parametric conflict resolution policy* that establishes which authorization prevails possibly exploiting information about the authorization sources. The exact conflict resolution policy depends on the access control model being modeled.

In the following, we first deal with the problem of conflicts and we then present a formal semantics for ACMIs.

5.1 Conflict Management

Let $D(\mathcal{I})$ be an ACMP. With $D(\mathcal{I})_{ground}$ we denote the set of all ground rules of $D(\mathcal{I})$ obtained by replacing each variable appearing in a rule of $D(\mathcal{I})$ with

[7]According to the ISA hierarchy represented in Figure 3 (d).

a constant of the "right" type. Conflicts and conflicting rules are defined as follows.

Definition 3. Let $D(\mathcal{I})$ be an ACMP over $\mathcal{L}$. Two atoms A_1, A_2 are conflicting in $\mathcal{L}$ if $A_1 = Auth_p(O : o, S : s, P : p, G : g', \epsilon : +, O' : o', S' : s', P' : p')$, $A_2 = Auth_p(O : o, S : s, P : p, G : g'', \epsilon : -, O' : o'', S' : s'', P' : p'')$, $o, o', o'', s, s', s'', p, p', p'', g', g'' \in Z$. The pair $(\langle(o, s, p, g', +, o', s', p'), (o, s, p, g'', -, o'', s'', p'')\rangle)$ is a conflict. The set of all conflicts that can be generated over $\mathcal{L}$ is denoted by conflicts($\mathcal{L}$). Let $r_1, r_2 \in D(\mathcal{I})_{ground}$. r_1 and r_2 are conflicting if $head(r_1)$ and $head(r_2)$ are conflicting.

It is well known that there is no unique solution to the problem of conflict management and that several conflict resolution policies can be defined depending on the specific domain [Ferrari and Thuraisingham 2000]. Examples of conflict resolution policies are denials take precedence, most specific authorization takes precedence, and permissions take precedence.

In order to provide a flexible conflict resolution mechanism, a parametric conflict resolution policy is introduced that, for each conflict, specifies how the conflict has to be resolved, possibly also taking into account the authorization sources.

Definition 4 (*Conflict Resolution Policy*). Let $\mathcal{L}$ be an ACML. A conflict resolution policy for $\mathcal{L}$ (denoted by $\mathcal{F}_\mathcal{L}$) is a total function from conflicts($\mathcal{L}$) to $\{+, -\}$.

Given a conflict c, the intended meaning of $\mathcal{F}_\mathcal{L}(c)$ is to choose whether the positive or the negative authorization in c should prevail.

Example 2. Consider the ACMI I presented in Example 1 and, in particular, the following authorizations rules, taken from Figure 4:

AC1. $Auth_d(O : \#9(doc2), S : \#2(Dev), P : \#11(Write), G : \#4(Bob), \epsilon : +) \leftarrow$
AC2. $Auth_d(O : \#9(doc2), S : \#5(Mary), P : \#11(Write), G : \#3(Ann), \epsilon : -) \leftarrow$
$Auth(O : \#9(doc2), S : \#2(Dev), P : \#11(Write), G : \#4(Bob), \epsilon : +)$

AC1 states that members of group Dev are authorized by Bob to write doc2. AC2 states that Mary is denied by Ann to write doc2 if members of group Dev are authorized by Bob to write the same document.

From I, the following facts can be derived:

F1. $Auth_p(O : \#9(doc2), S : \#2(Dev), P : \#11(Write), G : \#4(Bob), \epsilon : +, O' : \#9(doc2), S' : \#2(Dev), P' : \#11(Write))$, from AC1 and rule 12 of Table III;

F2. $Auth_p(O : \#9(doc2), S : \#5(Mary), P : \#11(Write), G : \#4(Bob), \epsilon : +, O' : \#9(doc2), S' : \#2(Dev), P' : \#11(Write))$, from F1 and rule 4 of Table III;

F3. $Auth(O : \#9(doc2), S : \#2(Dev), P : \#11(Write), G : \#4(Bob), \epsilon : +)$, from F1 and rule 13 of Table III;

F4. $Auth_d(O : \#9(doc2), S : \#5(Mary), P : \#11(Write), G : \#3(Ann), \epsilon : -)$, from F3 and AC2;

F5. $Auth_p(O : \#9(doc2), S : \#5(Mary), P : \#11(Write), G : \#3(Ann), \epsilon : -, O' : \#9(doc2), S' : \#5(Mary), P' : \#11(Write))$, from F4 and rule 12 of Table III.

F2 and F5 are conflicting and the pair:

$$c_1 \equiv (\langle(\#9(doc2), \#5(Mary), \#11(Write), \#4(Bob), +, \#9(doc2), \#2(Dev), (\#11(Write)), \#9(doc2), \#5(Mary), \#11(Write), \#3(Ann), -, \#9(doc2), \#5(Mary), \#11(Write))\rangle)$$

is a conflict, thus it is contained into the set of all conflicts $\mathsf{conflicts}(\mathcal{L})$.

Consider a conflict resolution policy $\mathcal{F}_\mathcal{L}$ for $\mathcal{L}$ defined as follows: for each conflict $c \in \mathsf{conflicts}(\mathcal{L})$, $\mathcal{F}_\mathcal{L}(c) = +$. According to this conflict resolution policy, positive authorizations always prevail over negative ones. Thus, in the case of our conflict, this means that authorization $Auth_p(\#9(doc2), \#5(Mary), \#11(Write), \#4(Bob), +, \#9(doc2), \#2(Dev), \#11(Write))$ prevails. As a result, Mary can write doc2.

As an alternative, suppose a conflict resolution policy is defined that takes into account the source of the authorizations involved in a conflict as follows: (i) if the subjects of the sources are ordered by the group hierarchy, the authorization specified for the most specific group is chosen; (ii) if a subject is a user and the other is the group the user belongs to, the user's authorization prevails; (iii) otherwise the negative one prevails. This policy can be formally defined as follows: for each conflict $c \in \mathsf{conflicts}(\mathcal{L})$, $c \equiv (\langle(o, s, p, g', +, o', s', p'), (o, s, p, g'', -, o'', s'', p'')\rangle)$:

$$\mathcal{F}_\mathcal{L}(c) = \begin{cases} + & \begin{cases} \text{if } s' \text{ and } s'' \text{ are groups and } s' \prec_G s'' \text{ or} \\ \text{if } s'' \text{ is a group and } s' \text{ is a user belonging to } s'' \end{cases} \\ - & \text{otherwise} \end{cases}$$

According to the previous conflict resolution policy, since Mary is a user belonging to group Dev, the negative authorization prevails. Thus, Mary is forbidden by Ann to write doc2.

5.2 A Model-Theoretic Semantics for ACMIs

ACMPs are logic programs with (arbitrary) negation. Since we make no restriction on the type of negation, we know, from logic programming, that a single meaning cannot be always assigned to these programs. This means that, in general, an ACMP is associated with different sets of entailed authorizations. The most general semantics for logic programs with negation is the stable model semantics [Ullman 1989]. This semantics assigns to a logic program a number (possibly zero) of alternative models,[8] each representing a set of consistent authorizations that can be possibly assigned to subjects.

In the following, we propose a stable model semantics for ACMIs. Most of the notions we introduce in the following are classical logic programming concepts. However, we need to extend the classical stable model semantics to deal with conflicts.

Before presenting the proposed semantics, some preliminary definitions have to be given. Let $D(\mathcal{I})$ be an ACMP over $\mathcal{L}$. The base $B_{D(\mathcal{I})}$ of $\mathcal{L}$ is the set of all ground atoms that can be constructed from predicate symbols in

[8]Here, the term "model" has a logic programming meaning (see below).

B and constants in Z. A set of ground atoms is consistent if it does not contain any conflicting atom. An interpretation I for $D(\mathcal{I})$ is any consistent subset of $B_{D(\mathcal{I})}$. Let I be an interpretation for $D(\mathcal{I})$, L a ground literal, and $r = H \leftarrow A_1, \ldots, A_n, not\ B_1, \ldots, not\ B_m$ a ground rule. Then, L ($not\ L$) is true with respect to I if $L \in I$ ($L \notin I$); the body of r is true in I if all its ground literals are true in I. An authorization rule $r \in D(\mathcal{I})_{ground}$ is true in I if either its head is true in I or its body is not true in I.

In traditional logic programming, a *model* is simply defined as an interpretation in which all program rules are true. This notion is not sufficient in our context since we have to deal with conflicts and to ensure that the model does not contain conflicting atoms. This is possible by not considering all the rule instances that lead to some conflicts. This notion is formalized by the concept of *discarded rule*.

Definition 5 (*Discarded Rules*). Let $\mathcal{F}_{\mathcal{L}}$ be a conflict resolution policy for an ACML $\mathcal{L}$. Let I be an interpretation for an ACMP $D(\mathcal{I})$ over $\mathcal{L}$, and let r_1 be an authorization rule in $D(\mathcal{I})_{ground}$. Rule r_1 is discarded in I if there exists $r_2 \in D(\mathcal{I})_{ground}$ such that r_1 and r_2 are conflicting, r_2 is true in I, $head(r_1) = Auth_p(O : o, S : s, P : p, G : g, \epsilon : k, O' : o', S' : s', P' : p')$, $head(r_2) = Auth_p(O : o, S : s, P : p, G : g', \epsilon : k', O' : o'', S' : s'', P' : p'')$, $k' \neq k$, and $\mathcal{F}_{\mathcal{L}}(\langle (o, s, p, g, k, o', s', p'), (o, s, p, g', k', o'', s'', p'')\rangle) = k'$.

A discarded rule is therefore an authorization rule generating an authorization a which generates a conflict with the authorization a' generated by another authorization rule, whose body is true in the considered interpretation, and such that the conflict resolution function gives priority to a'.

Based on the previous notion, the concept of truth in an interpretation is replaced by the concept of *satisfaction*, defined as follows.

Definition 6 (*Rule Satisfaction*). Let $D(\mathcal{I})$ be an ACMP over $\mathcal{L}$, and I an interpretation for $D(\mathcal{I})$. An authorization rule $r \in D(\mathcal{I})_{ground}$ is satisfied by I if either r is true in I or it is discarded by I.

A model for an ACMP is now defined as follows.

Definition 7 (*Model*). Let $D(\mathcal{I})$ be an ACMP over $\mathcal{L}$ and M be an interpretation for $D(\mathcal{I})$. M is a model for $D(\mathcal{I})$ if every rule in $D(\mathcal{I})_{ground}$ is satisfied by M. A model is *consistent* if it does not contain *ErrorC* atoms.

The previous definition means that in a model each rule of the ACMP must be either true or discarded due to the chosen conflict resolution policy. By removing discarded rules from a model, we generate a *reduction* of the considered model. By closing this reduced set with respect to the usual logical inference operator, we obtain a *stable model*.

Definition 8 (*Reduction of an Instance*). Let $D(\mathcal{I})$ be an ACMP over $\mathcal{L}$, and I an interpretation for $D(\mathcal{I})$. The reduction of $D(\mathcal{I})$ w.r.t. I, denoted by $R_I(D(\mathcal{I}))$, is the set of rules in $D(\mathcal{I})_{ground}$ that are not discarded in I and whose body is true in I, that is, $R_I(D(\mathcal{I})) = \{r | r \in D(\mathcal{I})_{ground}, r$ is not discarded by I, $body(r)$ is true in $I\}$.

Definition 9 (*Stable Authorization Model*). Let $D(\mathcal{I})$ be an ACMP over $\mathcal{L}$ and M a model for $D(\mathcal{I})$. M is a stable authorization model if $M \equiv T^{\infty}_{R_M(D(\mathcal{I}))}(\emptyset)$, where $T_Y : 2^{B_{D(\mathcal{I})}} \rightarrow 2^{B_{D(\mathcal{I})}}$ is the usual fixpoint operator. A stable authorization model is *consistent* if it does not contain *ErrorC* atoms. Given a stable model M, we use the following notation:
$M^{+}_{Auth} = \{\langle \#o, \#s, \#p, \#g, + \rangle | Auth(O : \#o, S : \#s, P : \#p, G : \#g, \epsilon : +) \in M\}$
$M^{-}_{Auth} = \{\langle \#o, \#s, \#p, \#g, - \rangle | Auth(O : \#o, S : \#s, P : \#p, G : \#g, \epsilon : -) \in M\}$
$M_{Auth} = M^{+}_{Auth} \cup M^{-}_{Auth}$
$M^{+}_{\overline{Auth}} = \{\langle \#o, \#s, \#p, \#g, + \rangle | Auth(O : \#o, S : \#s, P : \#p, G : \#g, \epsilon : +),$
$subject(self : \#s, \ldots) \in M\}$.

Consistent stable models can be used to assign a semantics to access control model instances. In particular, in our framework, the semantics of an instance (that is, of the corresponding ACMP) is not a single stable model but the set of all of its consistent stable models. This approach has the advantage of computing each consistent set of assignments of access authorizations, and thus does not make any assumption on the one to be selected, that depends on the access control model being represented.

Definition 10. Let $D(\mathcal{I})$ be an ACMP over $\mathcal{L}$. The semantics of $D(\mathcal{I})$, denoted by $\mathcal{S}(D(\mathcal{I}))$, is the set of all the consistent stable models of $D(\mathcal{I})$. The generalized semantics of $D(\mathcal{I})$, denoted by $\mathcal{GS}(D(\mathcal{I}))$, is the set of all the stable models of $D(\mathcal{I})$.

Example 3. Consider the following rules, taken from Example 1:

$r1 : Auth_d(O : \#8(doc1), S : \#3(Ann), P : \#11(Write), G : \#5(Mary), \epsilon : +) \leftarrow$
$\quad not\ Auth(O : \#8(doc1), S : \#4(Bob), P : \#11(Write), G : \#5(Mary), \epsilon : +)$
$r2 : Auth_d(O : \#8(doc1), S : \#4(Bob), P : \#11(Write), G : \#5(Mary), \epsilon : +) \leftarrow$
$\quad not\ Auth(O : \#8(doc1), S : \#3(Ann), P : \#11(Write), G : \#5(Mary), \epsilon : +).$

In the set of authorizations entailed by any program containing the previous rules, *Mary* cannot authorize both *Bob* and *Ann* to write *doc*1. Indeed, based on the previous rules, one authorization is entailed only if the other one is not. This means that the program admits two different stable models: in the first, *Mary* authorizes *Bob* to write document *doc1*, in the second *Mary* authorizes *Ann* to write document *doc1*.

6. EXAMPLES OF APPLICATION OF THE PROPOSED FRAMEWORK

To demonstrate the applicability of our framework, in the following we show how two well-known access control models can be seen as instances of our framework. The models we consider are the Bell and La Padula model [Bell and Padula 1975] and the NIST role-based model [Sandhu et al. 1996].

In order to show that these models can be represented in our framework, we show that for each access control model M an access control model instance exists such that its stable models exactly represent the set of access authorizations entailed by M.

Before presenting the examples of the framework usage, we formalize the notion of representability of an access control model in our framework.

Definition 11. Let ac be an access control model. ac is *representable* in our framework if there exists an ACMS S and an IACMI I over S such that, for each instance I_{ac} of ac, there exists an EACMI E over S such that $\mathcal{I} = I \cup E$ agrees with I_{ac}.

I_{ac} agrees with $\mathcal{I}$ if:

— I_{ac} entails the sets of authorizations $A_1, \ldots, A_n$;[9]

— $\mathcal{S}(D(\mathcal{I})) = \{M_1, \ldots, M_n\}$;

— for each A_i, $i = 1, \ldots, n$, there exists a unique M_j, $1 \leq j \leq n$, such that $A_i \equiv M_{j\,auth}$, and vice versa.

Let:

$subject(self : subject, a_1^s : t_1^s, \ldots, a_h^s : t_h^s)$
$object(self : object, a_1^o : t_1^o, \ldots, a_k^o : t_k^o)$
$privilege(self : privilege, a_1^p : t_1^p, \ldots, a_t^p : t_t^p)$

be the class entity schemas for class "subject", "object", and "privilege" in S. In each pair, $a_{<pos>}^{<class>}$ and $t_{<pos>}^{<class>}$ represent the label $(a_{<pos>}^{<class>})$ and the type $(t_{<pos>}^{<class>})$ of the attribute in position $< pos >$ for the class entity $< class >$, respectively. $A_i \equiv M_{j\,auth}$ holds if both the following conditions are satisfied:

— for any tuple $\langle o, s, p, g, \epsilon \rangle \in A_i$, M_j contains facts: $subject(self : \#s, a_1^s : e_1, \ldots, a_h^s : e_h)$, $object(self : \#o, a_1^o : f_1, \ldots, a_k^o : f_k)$, $privilege(self : \#p, a_1^p : g_1, \ldots, a_t^p : g_t)$, $subject(self : \#g, a_1^s : v_1, \ldots, a_h^s : v_h)$, and $M_{j\,auth}$ contains the tuple $\langle \#o, \#s, \#p, \#g, \epsilon \rangle$, where $\#o, \#s, \#p, \#g$ are the unique identifiers for the object o, the subject s, the privilege p, and the grantor g respectively.

— for any tuple $\langle \#o, \#s, \#p, \#g, \epsilon \rangle \in M_{j\,auth}$, such that M_j contains facts: $subject(self : \#s, a_1^s : e_1, \ldots, a_h^s : e_h)$, $object(self : \#o, a_1^o : f_1, \ldots, a_k^o : f_k)$, $privilege(self : \#p, a_1^p : g_1, \ldots, a_t^p : g_t)$, $subject(self : \#g, a_1^s : v_1, \ldots, a_h^s : v_h)$, A_i contains the authorization $\langle o, s, p, g, \epsilon \rangle$.

All the proofs of the results presented in the following sections are reported in Appendix B.

6.1 Bell and La Padula Model

In the Bell and La Padula model [Bell and Padula 1975] subjects, objects, and privileges are non-hierarchical domains. Subjects can be either single users or processes, whereas the privileges considered by the model are: `append`, `write`, `read`, and `execute`.[10] Subjects and objects are assigned an access class. Access classes are partially ordered according to a dominance relationship. Data accesses are regulated by the following two rules:

— **Simple security property**: a subject has a **read** access on an object if its access class dominates the access class of the object;

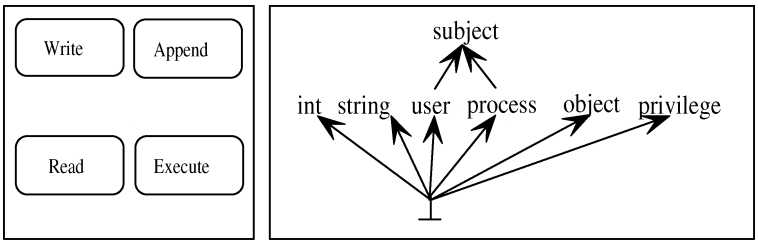
— ***-Property**: a subject has a **write** access on an object if its access class is equal to the access class of the object. By contrast, a subject can exercise the

[9]Each authorization is represented as a tuple $\langle o, s, p, g, \epsilon \rangle$.

[10]For simplicity, we do not consider the `control` privilege, that is, the privilege to extend to another subject one or more of the other privileges.

subject($self$: $subject$, $name$: $string$, $access\_class$: $string$)
user($self$: $user$)
process($self$: $process$, id_{proc} : int)
object($self$: $object$, $name$: $string$, $access\_class$: $string$)
privilege($self$: $privilege$, $name$: $string$)
LessC(C_1 : $string$, C_2 : $string$)
InLessC(C_1 : $string$, C_2 : $string$)
$Scheme^*$(**user**) = ($self$: $user$, $name$: $string$, $access\_class$: $string$)
$Scheme^*$(**process**) = ($self$: $process$, id_{proc} : int, $name$: $string$, $access\_class$: $string$)

Fig. 5. Schema for the Bell and La Padula classes.



(a) Privileges (b) ISA hierarchy

Fig. 6. Privileges and ISA hierarchy for the Bell and La Padula model.

append privilege on an object if its access class is dominated by the access class of the object.

The Bell and La Padula model can be represented in our framework by constructing an ACMS S and an ACMI $\mathcal{I}$ over S as follows.

Access Control Model Schema. To represent the Bell and La Padula model in our framework, we consider an ACMS S consisting of the following components: (I) $B = K \cup R$ such that: (i) $K = K_{built\_in} \cup K_{basic}$, where $K_{built\_in} = \{\bot, int, string\}$, and $K_{basic} = \{subject, user, process, object, privilege\}$; (ii) $R = R_{domain} \cup R_{auth} \cup R_{constraint} \cup R_{user\_def}$, where $R_{domain} = \emptyset$, $R_{auth} = \{Auth_d, Auth_p, Auth\}$, $R_{constraint} = \emptyset$, and $R_{user\_def} = \{LessC, InLessC\}$; (II) function Scheme for the elements in $K_{basic} \cup R_{user\_def}$ and R_{auth}, presented in Figure 5 and Table I, respectively; (III) function ISA, graphically represented in Figure 6 (b); (IV) set A, containing the special name "self" and the attribute names appearing in Figure 5; (V) Z containing oids for the elements of K. Access classes are modeled by associating an attribute $access\_class$ with subjects and objects whereas the hierarchical organization of access classes is expressed by using the user-defined predicates $LessC$ and $InLessC$.

Access Control Model Instance. Let I_{ac} be an instance of the Bell and La Padula access control model; an instance $\mathcal{I}$ over S that agrees with I_{ac} is composed of the following components:

(1) DC: we insert in DC a fact for each subject, user, process, object, and privilege that is defined in I_{ac}. These facts keep track of the subject and object access class.
(2) DSC: empty.

(3) AC: the definition of predicate $Auth_d$ must represent the simple security property and the *-Property. Thus, the following rules must be introduced in AC:
— simple security property:
3a. $Auth_d(O : X, S : Y, P : read, G : \#SA, \epsilon : +) \leftarrow$
$subject(self : Y, name : M, access\_class : K), object(self : X, name : N, access\_class : K)$;
3b. $Auth_d(O : X, S : Y, P : read, G : \#SA, \epsilon : +) \leftarrow$
$subject\ (self : Y, name : M, access\_class : W), object(self : X, name : N, access\_class : K), InLessC(C_1 : K, C_2 : W)$,
where #*SA* denotes the identifier of the Security Administrator.
— *-Property:
3c. $Auth_d(O : X, S : Y, P : append, G : \#SA, \epsilon : +) \leftarrow$
$subject(self : Y,\ name : M, access\_class : K), object(self : X, name : N, access\_class : K)$;
3d. $Auth_d(O : X, S : Y, P : append, G : \#SA, \epsilon : +) \leftarrow$
$subject(self : Y,\ name : M, access\_class : W), object(self : X, name : N, access\_class : K), InLessC(C_1 : W, C_2 : K)$;
3e. $Auth_d(O : X, S : Y, P : write, G : \#SA, \epsilon : +) \leftarrow subject(self : Y, name : M, access\_class : K), object(self : X, name : N, access\_class : K)$.
Predicate *LessC* is defined as follows:
— $\forall c_1, c_2 \in Z$, $LessC(C_1 : c_1, C_2 : c_2) \in \mathcal{I}$ *iff* c_2 dominates c_1 in the access class hierarchy specified by I_{ac}.
Finally, predicate $InLessC$ is defined by the following rules:
3f. $InLessC(C_1 : X, C_2 : Y) \leftarrow LessC(C_1 : X, C_2 : Y)$;
3g. $InLessC(C_1 : X, C_2 : Y) \leftarrow LessC(C_1 : X, C_2 : K), InLessC(C_1 : K, C_2 : Y)$.

(4) PC: since no hierarchies are present, no rules except rule 12 in Table III are extracted for predicate $Auth_p$. On the other hand, predicate *Auth* is defined by extracting rule 13 from the set of rules presented in Table III.

(5) CC: empty.

Note that, for simplicity, in defining the mapping, we have made the assumption that a user can only establish login sessions with his/her access class. However, the mapping can be extended with proper user-defined predicates to relax this assumption and thus allowing a user to establish any login session whose access class is dominated by the access class of the user.

We can state that the Bell and La Padula model is representable in our framework, since it is easy to show that the (unique) model of the instance presented above contains all and only those authorizations that are entailed by the Bell and La Padula rules.

THEOREM 1 (REPRESENTABILITY). *The Bell and La Padula access control model is representable in our framework.*

6.2 The NIST Model

In the following, we show how the general framework for modeling role-based access control (RBAC) models proposed by Sandhu et al. [2000] can be

represented in our framework. The NIST model is defined by four levels of increasing complexity such that each level adds to the previous one new features. These levels are described in the following.

Flat RBAC. *Flat RBAC* is the base level, able to capture the basic classical features of an RBAC model: users acquire permissions from roles; a user can be assigned to many roles and a role can refer to many users (the same holds for the relation existing between permissions and roles); users can simultaneously exercise permissions deriving from different roles. Additionally, Flat RBAC supports user-role review, that is, it must be possible to determine which roles are assigned to a specific user and which are the users authorized to play a specific role.

Hierarchical RBAC. *Hierarchical RBAC* adds to Flat RBAC the support for role hierarchies. Two different interpretations of role hierarchies are supported: the inheritance and the activation interpretation. In the first case, the activation of a role r_i implies the activations of all roles r_j that are less powerful than r_i and thus the inheritance of their permissions whereas, in the second case junior roles must be explicitly activated.

Constrained RBAC. *Constrained RBAC* adds to Hierarchical RBAC the support for separation of duty (SOD) constraints. Separation of duty is the ability to state which roles cannot be simultaneously assigned to the same user (static SOD) or which roles cannot be activated together by the same user (dynamic SOD).

Symmetric RBAC. *Symmetric RBAC* adds to Constrained RBAC support for permission-role review. This is the ability to determine which are the roles to which a particular permission is assigned as well as which are the permissions assigned to a particular role.

The basic components of the NIST model can be formally defined as follows:

— U, R, P, and S represent respectively the sets of users, roles, permissions, and sessions. Each permission is a pair (a, o) and represents a specific access mode a on object o. We thus denote with A and O the sets of access modes and objects, respectively. Thus, $P \subseteq A \times O$. Moreover, let $p \in P$ be a permission, we denote with p_a and p_o the access mode and the object in p, respectively.

— $UA \subseteq U \times R$ and $PA \subseteq P \times R$ represent respectively the user-role and the permission-role assignments. Let $f \in UA$, we denote with f_u and f_r the user and role specified in f. Similarly, let $g \in PA$, we denote with g_p and g_r the permission and role specified in g.

— $RH \subseteq R \times R$ represents the role hierarchy; $\forall r_i, r_j \in R$, $\langle r_i, r_j \rangle \in RH$, if r_j precedes r_i in the role hierarchy.

— $user : S \rightarrow U$ is a function that maps each session onto a single user.

— $roles : S \rightarrow 2^R$ is a function that maps each session onto a set of roles defined as follows: $\forall s \in S$, $roles(s) \subseteq \{r_i \in R | \langle r_i, r_j \rangle \in RH, f \in UA, f_u = user(s), f_r = r_j\}$; session s has the following permissions: $\bigcup_{r_i \in roles(s)} \{p \in P | \langle r_k, r_i \rangle \in RH, g \in PA, g_p = p, g_r = r_k\}$.

— C represents the set of specified constraints.

subject(*self* : *subject*, *name* : *string*)	**user**(*self* : *user*)
role(*self* : *role*)	**object**(*self* : *object*, *name* : *string*)
privilege(*self* : *privilege*, *name* : *string*)	**session**(*self* : *session*, *name* : *string*)
$Scheme^*($ **user**$) = ($*self* : *user*, *name* : *string*$)$	$Scheme^*($ **role**$) = ($*self* : *role*, *name* : *string*$)$

Fig. 7. Schema for NIST RBAC models.

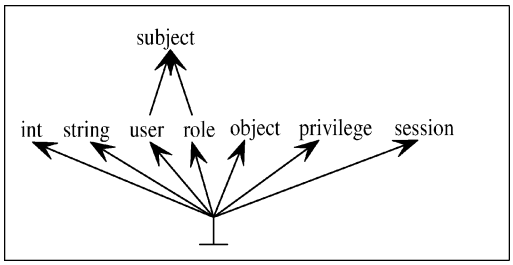


Fig. 8. The ISA hierarchy for NIST RBAC models.

In order to show how each of the above models can be represented by our framework, we show that, for each considered NIST level, an access control model instance exists such that its stable model exactly represents the set of access authorizations entailed by the considered access control model.

For simplicity, in presenting the mapping, differently from the NIST model, we assume that Flat RBAC also supports the notion of session. In Flat RBAC, we assume that during a session all the roles the user is authorized to play are activated. In the other cases, we assume that the user can activate a subset of the roles he/she is authorized to play.

6.2.1 *Flat RBAC.* In Flat RBAC users, roles, and permissions are flat domains; users and permissions are assigned to roles. Permissions are always positive. The ACMS and the ACMI for Flat RBAC can be constructed as follows.

Access Control Model Schema. To represent the Flat RBAC model in our framework, we consider an ACMS S consisting of the following components: (I) $B = K \cup R$, such that: (i) $K = K_{built\_in} \cup K_{basic}$, where $K_{built\_in} = \{\bot, int, string\}$, and $K_{basic} = \{subject, user, role, object, privilege, session\}$; (ii) $R = R_{domain} \cup R_{auth} \cup R_{constraint} \cup R_{user\_def}$, where $R_{domain} = \{Play, UserPlay, ActiveRole\}$, $R_{auth} = \{Auth_d, Auth_p, Auth\}$, $R_{constraint} = R_{user\_def} = \emptyset$; (II) function Scheme for the elements in K_{basic}, $R_{domain} \cup R_{auth} \cup R_{admin} \cup R_{user\_def}$, presented in Figure 7, and Table I; (III) function ISA, graphically represented in Figure 8; (IV) set A, containing the special name "self" and the attribute names appearing in Figure 7; (V) Z containing oids for the elements of K.

Access Control Model Instance. Let I_{ac} be an instance of the Flat RBAC model; an instance $\mathcal{I}$ over S that agrees with I_{ac} is composed of:

(1) DC: we insert in DC: *i)* a fact *user*(*self* : #*i*, *name* : *u*), for each user $u \in U$; *ii)* a fact *role*(*self* : #*i*, *name* : *r*), for each role $r \in R$; *iii)* a fact *session*(*self* : #*i*, *name* : *s*), for each $s \in S$; *iv)* a fact *object*(*self* : #*i*, *name* : *o*), for each

object $o \in O$, and *v)* a fact $privilege(self : \#i, name : a)$, for each access mode $a \in A$, where $\#i \in Z$ denotes a unique identifier for an object i.
Note that, since "user" and "role" are subclasses of "subject",[11] the following set of inherited elements $\{subject(self : \#i, name : u) | user(self : \#i, name : u) \in DC\} \cup \{subject(self : \#j, name : r) | role(self : \#j, name : r) \in DC\}$ holds.

(2) DSC: a definition (set of facts and/or rules) for each derived relation name in R_{domain}. In particular, the following facts and rules are introduced:
—$\forall f \in UA$, we insert the fact: $Play(U : \#f_u, R : \#f_r)$ in $\mathcal{I}$, where $\#f_u$ and $\#f_r$ are the identifiers of f_u and f_r, respectively.
—rule 4a of Table II;
—$\forall s \in S$, we insert in $\mathcal{I}$ the set of facts: $\{ActiveRole(U : \#user(s), S : \#s, R : \#r) \mid r \in roles(s)\}$, where $\#user(s), \#s, \#r$ are the identifiers of $user(s)$, s, and r, respectively.

(3) AC: for each permission-role assignment $g \in PA$, $g_p = (a, o)$, we insert in AC the fact: $Auth_d(O : \#o, S : \#g_r, P : \#a, G : \#SA, \epsilon : +)$, where $\#SA$ denotes the identifier of the Security Administrator , and $\#o$, $\#g_r$, and $\#a$ are the identifiers of o, g_r, and a respectively.

(4) PC: predicate $Auth_p$ is defined by rules 8 and 12 of Table III. Predicate $Auth$ is defined by rule 13 of Table III.

(5) CC: empty.

The following theorem states that the Flat RBAC model is representable in our framework.

THEOREM 2 (REPRESENTABILITY). *The Flat RBAC model is representable in our framework.*

Note that, the user-role review requirement of Flat RBAC can be easily satisfied by our mapping. Indeed, let I_{ac} be an instance of the Flat RBAC model constructed as pointed out above, and let $\mathcal{I}$ be the corresponding ACMI. Let M be the unique stable model of $D(\mathcal{I})$. The roles assigned to a specific user u can be determined by selecting from M all the facts $Play(U : \#u, R : \#r)$ such that $\#u$ is the identifier of u, and by considering the role component of these facts. Similarly, to determine which are the users authorized to play a role r, it is simply necessary to select from M the facts $Play(U : \#s, R : \#r)$ such that $\#r$ is the identifier of role r and taking the user component.

6.2.2 *Hierarchical RBAC.* Hierarchical RBAC model adds to Flat RBAC the support for role hierarchies. Two different interpretations of a role hierarchy are supported, as described before: permission and activation inheritance.

The ACMS and ACMI for Hierarchical RBAC can be constructed as follows.

Access Control Model Schema. To represent the Hierarchical RBAC model in our framework, we consider an ACMS S consisting of of the following components: (I) $B = K \cup R$ such that: (i) $K = K_{built\_in} \cup K_{basic}$, where $K_{built\_in} = \{\bot, int, string\}$, and $K_{basic} = \{subject, user, role, object,$

[11]According to the ISA hierarchy represented in Figure 8.

privilege, *session*}; (ii) $R = R_{domain} \cup R_{auth} \cup R_{constraint} \cup R_{user\_def}$, where $R_{domain} = \{Play, UserPlay, LessR, InLessR, ActiveRole\}$, $R_{auth} = \{Auth_d, Auth_p, Auth\}$, $R_{constraint} = R_{user\_def} = \emptyset$; (II) function Scheme for the elements in K_{basic}, $R_{domain} \cup R_{auth} \cup R_{admin} \cup R_{user\_def}$, presented in Figure 7, and Table I; (III) function ISA, graphically represented in Figure 8; (IV) set A, containing the special name "self" and the attribute names appearing in Figure 7; (V) Z containing oids for the elements of K.

Access Control Model Instance. Let I_{ac} be an instance of the Hierarchical RBAC model; an instance $\mathcal{I}$ over S that agrees with I_{ac} is composed of:

(1) DC: it coincides with the DC component of the Access Control Model Instance presented in Section 6.2.1.
(2) DSC: the following facts and rules must be introduced:
   — $\forall f \in UA$, we insert the fact: $Play(U : \#f_u, R : \#f_r)$ in $\mathcal{I}$, where $\#f_u$ and $\#f_r$ are the identifiers of f_u and f_r, respectively.
   — rules 4a and 4b of Table II;
   — $\forall \#r_i, \#r_j \in Z$, $LessR(R_1 : \#r_i, R_2 : \#r_j) \in \mathcal{I}$ iff $\langle r_i, r_j \rangle \in RH$, and $\#r_i$, $\#r_j$ are the identifiers of r_i and r_j respectively;
   — rules 3a and 3b of Table II representing the transitive closure of $LessR$;
   — $\forall s \in S$, we insert in $\mathcal{I}$ the set of facts $\{ActiveRole(U : \#user(s), S : \#s, R : \#r) \mid r \in roles(s)\}$, where $\#user(s), \#s, \#r$ are the identifiers of $user(s)$, s, and r, respectively.
(3) AC: it coincides with the AC component of the Access Control Model Instance presented in Section 6.2.1.
(4) PC: Predicate $Auth_p$ is defined by:
   — rule 7 of Table III, if I_{ac} supports the activation interpretation of the role hierarchy;
   — rules 5, and 8 of Table III, if I_{ac} supports the inheritance interpretation of the role hierarchy.
   Predicate $Auth$ is defined by rules 12 and 13 of Table III.
(5) CC: empty.

Note that, for simplicity, in the mapping, we have made the assumption that the two interpretations of the role hierarchies are mutually exclusive. The mapping can be easily extended to simultaneously support both interpretations. It is only necessary to maintain information on the session in which an authorization holds.

The following theorem states that the Hierarchical RBAC model is representable in our framework.

THEOREM 3 (REPRESENTABILITY). *The Hierarchical RBAC model is representable in our framework.*

6.2.3 *Constrained RBAC.* The Constrained RBAC model adds to Hierarchical RBAC the support for static and dynamic separation of duties. Thus, the ACMS and ACMI for Constraint RBAC are very similar to those defined for Hierarchical RBAC. The only differences are that in the ACMS of Constraint

RBAC, $R_{constraint} = \{ErrorC\}$, whereas the ACMI contains a non-empty constraint component that maps the constraints expressed in the instance of the Constraint RBAC being modeled. In our framework, static and dynamic separation of duty constraints can be expressed through the following rules:

—$ErrorC() \leftarrow UserPlay(U : X, R : Y), UserPlay(U : X, R : Z), A_1, \ldots, A_n$;
—$ErrorC() \leftarrow ActiveRole(U : X, S : Y, R : Z), ActiveRole(U : X, S : Y, R : K)$, $UserPlay(U : X, R : Z), UserPlay(U : X, R : K), B_1, \ldots, B_m$;

where $A_1, \ldots, A_n, B_1, \ldots, B_m$ are optional atoms, whose predicate must belong to the set of predicates defined in the ACMS, expressing additional conditions on the constraint validity.

6.2.4 *Symmetric RBAC.* Symmetric RBAC does not add any basic component to Constraint RBAC but it adds the support for permission-role review. This implies that it must be possible to easily determine the roles to which a particular permission is assigned and the permissions assigned to a specific role. In our framework, this information can be inferred from the analysis of the model of the generated instance. More precisely, let I_{ac} be an instance of Symmetric RBAC, generated using the same method presented for Constraint RBAC, and let $\mathcal{I}$ be the corresponding ACMI. Let M be the unique stable model of $D(\mathcal{I})$. To determine which are the roles to which a particular permission $p = (a, o)$ is granted, we need to extract from M_{auth} all the tuples $\langle \#o, \#s, \#a \rangle$[12] such that $\#s$ is the identifier of a role, and $\#o$ and $\#a$ are the identifiers of o and a, respectively, and then taking the subject component of these tuples. Analogously, to determine which permissions $p = (a, o)$ are assigned to a specific role r we extract from M_{auth} all the tuples $\langle \#r, \#o, \#a \rangle$, such that $\#r$ is the identifier of role r, and $\#o$ and $\#a$ are the identifiers of o and a, respectively, and we take the privilege and object components of these tuples.

7. COMPARISON DIMENSIONS

In the previous section, we have shown how the formal framework presented in this paper can be used to represent (possibly heterogeneous) access control models into a common formalism. As a result of the mapping, each access control model is expressed by an equivalent logic program. In this section, we use the framework and the result of the mapping as a tool for reasoning about the characteristics of the various access control models and for comparing their expressive power. Since access control models are expressed with a variety of different formalisms, making their comparison very difficult, the proposed formalism allows us to define a uniform context in which models can be homogeneously represented. Due to the nature of our formalism, the problem of access control model analysis and comparison becomes thus a matter of analyzing and comparing logic programs. We can thus exploit formal results proved in the logic programming field and adapt them to our context.

The analysis of access control models can be carried out by considering the properties of the model itself or by comparing the expressive power of distinct

[12]For simplicity, we do not consider the grantor and the sign components of authorizations.

access control models. Therefore, we can distinguish between *intra-model* and *inter-model* properties. Intra-model properties denote characteristics of a single access control model. By contrast, inter-model properties denote characteristics of a model in relation to another model, and are thus a means for comparing two different access control models according to a specified dimension. According to the scenario presented in Section 3, inter-model properties will be used to select, from a possibly existing library, models with similar characteristics with respect to the one the SA has chosen, enabling him/her to refine his/her choice. On the other hand, intra-model properties will help the SA to analyze the chosen model.

In the following, we present some inter- and intra-model properties, by formally defining them and presenting some examples of their applicability. All the proposed definitions and results assume that no conflict resolution policy and constraints are used. We leave to future work the investigation of the same issues in the presence of such a function and constraints.

7.1 Inter-Model Properties

Inter-model properties define the different dimensions that can be used to compare two access control models. The first comparison dimension concerns the modeling capabilities of the models. By modeling capabilities we mean all constructs provided by an authorization model to represent subjects, objects, and privileges. Examples of modeling capabilities include roles, groups, and negative/positive privileges. Another way of comparing two access control models is on the basis of the authorizations they enforce, independently from the possibly generated errors. The last considered comparison dimension concerns consistency of the models. In the following, we introduce these properties, presenting some examples and formally discussing decidability results.

7.1.1 *Structural Subsumption/Equivalence.* Consider two models, one supporting authorizations on groups and the other supporting only the specification of authorizations for single users. These models could entail the same set of user authorizations. However, their expressive power is not the same since the first supports groups, whereas the other does not. These considerations suggest us to consider a dimension, that we call *structural subsumption/equivalence*, that verifies whether two access control models are built from the same set of ACMS basic components.

Two aspects have to be considered when dealing with structural equivalence. The first concerns the components contained in the ACMS for the considered access control models. For example, if an access control model deals with groups, and therefore requires class names *group* and *user*, whereas another access control model only deals with users, thus requiring only class name *user*, the two access control models are not structurally equivalent. In this case, we say that the access control models are not *weakly structurally equivalent*. The second aspect concerns the attributes used to characterize subjects, objects, and privileges. For example, a mandatory access control model assigning to each subject and object an access class is structurally different from an access control model which does not consider this information. In this case, we say that the access control models are not *strongly structurally equivalent*.

Model1

subject(*self* : *subject*, *name* : *string*) **user**(*self* : *user*)
role(*self* : *role*) **object**(*self* : *object*, *name* : *string*)
privilege(*self* : *privilege*, *name* : *string*) **session**(*self* : *session*, *name* : *string*)
$Scheme_1^*$(**user**) = (*self* : *user*, *name* : *string*) $Scheme_1^*$(**role**) = (*self* : *role*, *name* : *string*)

IACMI

$r1 : InLessR(R_1 : X, R_2 : Y) \leftarrow LessR(R_1 : X, R_2 : Y)$
$r2 : InLessR(R_1 : X, R_2 : Y) \leftarrow LessR(R_1 : X, R_2 : Z), InLessR(R_1 : Z, R_2 : Y)$
$r3 : UserPlay(U : X, R : Y) \leftarrow Play(U : X, R : Y)$
$r4 : UserPlay(U : X, R : Y) \leftarrow Play(U : X, R : Z), InLessR(R_1 : Y, R_2 : Z)$
$r5 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : +, O' : X_5, S' : X_6, P' : X_7) \leftarrow Auth_p(O : X_1, S : X_8, P : X_3, G : X_4, \epsilon : +, O' : X_5, S' : X_6, P' : X_7), InLessR(R_1 : X_8, R_2 : X_2)$
$r6 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8) \leftarrow Auth_p(O : X_1, S : X_9, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8), UserPlay(U : X_2, R : X_9), ActiveRole(U : X_2, S : X_{10}, R : X_9)$
$r7 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_1, S' : X_2, P' : X_3) \leftarrow Auth_d(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5)$
$r8 : Auth(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5) \leftarrow Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8)$

Fig. 9. A role-based model (Model1 in the examples).

Weak and strong structural subsumption/equivalence can be formally defined as follows.

Definition 12 (*Weak and Strong Structural Subsumption/Equivalence*). Let ac_1 and ac_2 be two access control models representable in our framework by two ACMSs $S_1 =< B_1, Scheme_1, A_1, ISA_1, Z_1 >$ and $S_2 =< B_2, Scheme_2, A_2, ISA_2, Z_2 >$.

—ac_1 is weakly structurally (w-structurally) subsumed by ac_2 (denoted by $ac_1 \subseteq_w^s ac_2$), iff $B_1 - R_{user\_def} \subseteq B_2 - R_{user\_def}$. ac_1 is weakly structurally equivalent to ac_2 (denoted by $ac_1 \equiv_w^s ac_2$), if and only if both the following conditions hold: (i) $ac_1 \subseteq_w^s ac_2$, and (ii) $ac_2 \subseteq_w^s ac_1$.

—ac_1 is strongly structurally (s-structurally) subsumed by ac_2, denoted by $ac_1 \subseteq_s^s ac_2$, if the following conditions hold: (i) ac_1 is weakly structurally subsumed by ac_2; (ii) for each $b \in \{B_1 - R_{user\_def}\}$, $Scheme_1(b) \subseteq Scheme_2(b)$. ac_1 is strongly structurally equivalent to ac_2, denoted by $ac_1 \equiv_s^s ac_2$, if the following conditions hold: (i) $ac_1 \subseteq_s^s ac_2$, and (ii) $ac_2 \subseteq_s^s ac_1$.

Note that both w-structurally and s-structurally equivalence do not consider user-defined predicates, since they do not represent the structure of an access control model but auxiliary information used for the specification of authorization, propagation, and constraint rules.

Note that, based on the translation presented in Section 6, the Bell and La Padula model is not w-structurally equivalent with any RBAC model, due to the presence of class role in RBAC models.

Example 4. Consider two different role-based access control models, say Model1 and Model2, whose class entity names schema and corresponding IACMI are presented in Figure 9 and Figure 10, respectively. The main

Model2

subject($self$: $subject$, $name$: $string$)
user($self$: $user$)
role($self$: $role$, $temp\_dis$: $string$)
object($self$: $object$, $name$: $string$)
privilege($self$: $privilege$, $name$: $string$)
session($self$: $session$, $name$: $string$)
PropDir(R : $role$, P : $privilege$, $direction$: $string$)
$Scheme_2^*$(**user**) = ($self$: $user$, $name$: $string$)
$Scheme_2^*$(**role**) = ($self$: $role$, $temp\_dis$: $string$, $name$: $string$)

IACMI

$r1 : InLessR(R_1 : X, R_2 : Y) \leftarrow LessR(R_1 : X, R_2 : Y)$
$r2 : InLessR(R_1 : X, R_2 : Y) \leftarrow LessR(R_1 : X, R_2 : Z), InLessR(R_1 : Z, R_2 : Y)$
$r3 : UserPlay(U : X, R : Y) \leftarrow Play(U : X, R : Y)$
$r4 : UserPlay(U : X, R : Y) \leftarrow Play(U : X, R : Z), InLessR(R_1 : Y, R_2 : Z)$
$r5 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : +, O' : X_5, S' : X_6, P' : X_7) \leftarrow Auth_p(O : X_1, S : X_8, P : X_3, G : X_4, \epsilon : +, O' : X_5, S' : X_6, P' : X_7), InLessR(R_1 : X_8, R_2 : X_2), PropDir(R : X_8, P : X_3, direction : up)$
$r6 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : +, O' : X_5, S' : X_6, P' : X_7) \leftarrow Auth_p(O : X_1, S : X_8, P : X_3, G : X_4, \epsilon : +, O' : X_5, S' : X_6, P' : X_7), InLessR(R_1 : X_2, R_2 : X_8), PropDir(R : X_8, P : X_3, direction : down)$
$r7 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8) \leftarrow Auth_p(O : X_1, S : X_9, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8), UserPlay(U : X_2, R : X_9), ActiveRole(U : X_2, S : X_{10}, R : X_9), role(self : X_9, name : X_{11}, temp\_dis : false)$
$r8 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_1, S' : X_2, P' : X_3) \leftarrow Auth_d(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5)$
$r9 : Auth(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5) \leftarrow Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8)$

Fig. 10. A role-based model (Model2 in the examples).

difference between Model1 and Model2 is that Model2 associates with each role a boolean attribute, `temp_dis`, that is true when the role is temporarily not usable by the users authorized to play it. This means that when `temp_dis` is true the privileges assigned to that role are not propagated to the users authorized to play it. Moreover, Model2 contains a user-defined predicate, `PropDir`, that, for each role and privilege, specifies whether the privilege is propagated according to either an upward or downward direction in the role hierarchy starting from that role.

From Definition 12, it follows that Model1 and Model2 are w-structurally equivalent. Indeed, since user-defined predicates are not taken into account, the corresponding ACMSs are based on the same class names. On the other hand, they are not s-structurally equivalent, since class `role` has two different sets of attributes in the two models. However, Model1 is s-structurally subsumed by Model2, since all the attributes defining classes in Model1 are also contained in the schema of the corresponding classes in Model2.

Weak (strong) structural subsumption/equivalence is of course always decidable, as stated by the following theorem, since it corresponds to determining inclusion between finite sets.

THEOREM 4. *Structural subsumption/equivalence is decidable.*

7.1.2 *Access Subsumption/Equivalence.* Two access control instances are equivalent if they enforce exactly the same set of accesses. We call this kind of equivalence *access equivalence*. On the other hand, an access control instance is *access-subsumed* by another access control instance if the set of accesses enforced by the first instance is also enforced by the second one. For both these dimensions we can consider more than one version depending on the "granularity" of the sets of accesses we compare: a strong version compares all the accesses, a weaker version compares only sets of positive accesses, whereas the weakest version compares only sets of positive accesses where the subject is a user.

Subsumption and equivalence can be first analyzed with respect to access control model instances, resulting in the following definition.

Definition 13 (*Instance Access Subsumption/Equivalence*). Let ic_1 and ic_2 be two access control model instances of two access control models ac_1 and ac_2, representable in our framework by two ACMIs I_1 and I_2, over two ACMS S_1 and S_2, respectively. We say that:[13]

—ic_1 is strongly access-subsumed by ic_2 (denoted by $ic_1 \subseteq_s^{ia} ic_2$) if for each $m_1 \in \mathcal{GS}(D(I_1))$ there exists $m_2 \in \mathcal{GS}(D(I_2))$ such that $m_{1_{Auth}} \subseteq m_{2_{Auth}}$. ic_1 is strongly access-equivalent to ic_2 (denoted by $ic_1 \equiv_s^{ia} ic_2$) if the following conditions hold: (i) $ic_1 \subseteq_s^{ia} ic_2$; and (ii) $ic_2 \subseteq_s^{ia} ic_1$.

—ic_1 is positively access-subsumed by ic_2 (denoted by $ic_1 \subseteq_p^{ia} ic_2$) if for each $m_1 \in \mathcal{GS}(D(I_1))$ there exists $m_2 \in \mathcal{GS}(D(I_2))$ such that $m^+_{1_{Auth}} \subseteq m^+_{2_{Auth}}$. ic_1 is positively access-equivalent to ic_2 (denoted by $ic_1 \equiv_p^{ia} ic_2$) if the following conditions hold: (i) $ic_1 \subseteq_p^{ia} ic_2$; and (ii) $ic_2 \subseteq_p^{ia} ic_1$.

—ic_1 is user-access subsumed by ic_2 (denoted by $ic_1 \subseteq_u^{ia} ic_2$) if for each $m_1 \in \mathcal{GS}(D(I_1))$ there exists $m_2 \in \mathcal{GS}(D(I_2))$ such that $m^+_{1_{\overline{Auth}}} \subseteq m^+_{2_{\overline{Auth}}}$. ic_1 is user-access equivalent to ic_2 (denoted by $ic_1 \equiv_u^{ia} ic_2$) if both the following conditions hold: (i) $ic_1 \subseteq_u^{ia} ic_2$; and (ii) $ic_2 \subseteq_u^{ia} ic_1$.

Example 5. Consider the Bell and La Padula model presented in Figure 11, and the RBAC model presented in Figures 12 and 13. The two models are not w-structurally equivalent, since roles are not present in the Bell and La Padula model.

Now consider the instance of the Bell and La Padula model obtained by considering the EACMI in Figure 11. This instance deals with three users (*Ann*, *Bob*, *Mary*), three objects ($o1$, $o2$, $o3$), three privileges (read (R), append (A), write (W)), and three access classes ($c1$, $c2$, $c3$) such that:

—$c2 < c1$, $c3 < c1$;
—the access class of *Ann* and $o1$ is $c1$;
—the access class of *Bob* and $o2$ is $c2$;
—the access class of *Mary* and $o3$ is $c3$.

[13]See Definition 9 for the meaning of the notation used in this definition.

Since the considered IACMI rules do not use negation, this instance admits a unique stable model, generating the authorization set presented in Figure 11.

Now consider an instance of the RBAC Model2 presented in Example 4, represented in our framework by the ACMI presented in Figure 12. This instance deals with three users (*Ann*, *Bob*, *Mary*), three objects ($o1$, $o2$, $o3$), three privileges (read (R), append (A), write (W)), three roles ($r1, r2, r3$), and three sessions (one for each user) such that:

—$r2 < r1$ and $r3 < r1$;
—*Ann* can play role $r1$ and activates it in her session;
—*Bob* can play role $r2$ and activates it in his session;
—*Mary* can play role $r3$ and activates it in her session;
— authorizations involving role $r1$ and privilege A can be propagated downward along the role hierarchy;
— authorizations involving role $r2$ (or $r3$) and privilege R can be propagated upward along the role hierarchy;
— role $r1$ is authorized to exercise privileges R, A, W on $o1$;
— role $r2$ is authorized to exercise privileges R, A, W on $o2$;
— role $r3$ is authorized to exercise privileges R, A, W on $o3$.

Since the considered IACMI rules do not use negation, this instance also admits a unique stable model, generating the authorization set presented in Figure 13.

From Figures 11, 12, and 13 it follows that the accesses entailed by the considered instances coincide, even if the structures of the considered models are quite different.

On the other hand, models with a similar structure, such as the models presented in Example 4, can greatly differ with respect to the entailed authorization sets. Consider for example two instances of such models, presented in Figures 12, 13, and in Figures 14, and 15, respectively. Altough the extensional part is the same for the two models, the accesses entailed by these instances are different and are not comparable even if we use the weakest dimension.

Instance access subsumption and equivalence are always decidable, since stable model semantics is always computable. However, comparing two access control models on a per-instance basis could be useful only in specific situations in which there is the interest, given two specific access control model instances, of deciding whether the accesses entailed by the first instance are equivalent or subsumed by the set of accesses entailed by the second one. However, in a general setting, it is more useful to reason on the above properties independently from the specific instances, determining whether these properties hold for *all* instances of two access control models. For example, we may be interested in determining whether, for each instance of a given model, there exists an instance of the second one, entailing the same set of authorizations. This property will help us in comparing access control model expressive power.

One of the problems to cope with in order to answer these questions is which properties the instances on which the analysis is performed must satisfy. To this purpose, it seems reasonable to assume the two instances contain the

Model3

subject$(self : subject, name : string, access\_class : string)$

user$(self : user)$

process$(self : process, id_{proc} : int)$

object$(self : object, name : string, access\_class : string)$

privilege$(self : privilege, name : string)$

LessC$(C_1 : string, C_2 : string)$

InLessC$(C_1 : string, C_2 : string)$

$Scheme_3^*($ **user**$) = (self : user, name : string, access\_class : string)$

$Scheme_3^*($ **process**$) = (self : process, id_{proc} : int, name : string, access\_class : string)$

EACMI

$user(self : \#1, name : Ann, access\_class : c1)$
$user(self : \#2, name : Bob, access\_class : c2)$
$user(self : \#3, name : Mary, access\_class : c3)$
$object(self : \#4, name : o1, access\_class : c1)$
$object(self : \#5, name : o2, access\_class : c2)$
$object(self : \#6, name : o3, access\_class : c3))$
$subject(self : \#1, name : Ann, access\_class : c1)$
$subject(self : \#2, name : Bob, access\_class : c2)$
$subject(self : \#3, name : Mary, access\_class : c3)$
$privilege(self : \#7, name : R)$
$privilege(self : \#8, name : A)$
$privilege(self : \#9, name : W)$
$LessC(C_1 : c2, C_2 : c1)$
$LessC(C_1 : c3, C_2 : c1)$

IACMI

$r1 : Auth_d(O : X, S : Y, P : \#7(R), G : \#SA, \epsilon : +) \leftarrow subject(self : Y, name : M, access\_class : K), object(self : X, name : N, access\_class : K)$

$r2 : Auth_d(O : X, S : Y, P : \#7(R), G : \#SA, \epsilon : +) \leftarrow subject(self : Y, name : M, access\_class : W), object(self : X, name : N, access\_class : K), InLessC(C_1 : K, C_2 : W)$

$r3 : Auth_d(O : X, S : Y, P : \#8(A), G : \#SA, \epsilon : +) \leftarrow subject(self : Y, name : M, access\_class : K), object(self : X, name : N, access\_class : K)$

$r4 : Auth_d(O : X, S : Y, P : \#8(A), G : \#SA, \epsilon : +) \leftarrow subject(self : Y, name : M, access\_class : W), object(self : X, name : N, access\_class : K), InLessC(C_1 : W, C_2 : K)$

$r5 : Auth_d(O : X, S : Y, P : \#9(W), G : \#SA, \epsilon : +) \leftarrow subject(self : Y, name : M, access\_class : K), object(self : X, name : N, access\_class : K)$

$r6 : InLessC(C_1 : X, C_2 : Y) \leftarrow LessC(C_1 : X, C_2 : Y)$

$r7 : InLessC(C_1 : X, C_2 : Y) \leftarrow LessC(C_1 : X, C_2 : K), InLessC(C_1 : K, C_2 : Y)$

$r8 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_1, S' : X_2, P' : X_3) \leftarrow Auth_d(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5)$

$r9 : Auth(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5) \leftarrow Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8)$

Authorizations set

$(o1, Ann, R)$	$(O : \#4, S : \#1, P : \#7, G : \#SA, \epsilon : +)$
$(o2, Bob, R)$	$(O : \#5, S : \#2, P : \#7, G : \#SA, \epsilon : +)$
$(o3, Mary, R)$	$(O : \#6, S : \#3, P : \#7, G : \#SA, \epsilon : +)$
$(o2, Ann, R)$	$(O : \#5, S : \#1, P : \#7, G : \#SA, \epsilon : +)$
$(o3, Ann, R)$	$(O : \#6, S : \#1, P : \#7, G : \#SA, \epsilon : +)$
$(o1, Ann, A)$	$(O : \#4, S : \#1, P : \#8, G : \#SA, \epsilon : +)$
$(o2, Bob, A)$	$(O : \#5, S : \#2, P : \#8, G : \#SA, \epsilon : +)$
$(o3, Mary, A)$	$(O : \#6, S : \#3, P : \#8, G : \#SA, \epsilon : +)$
$(o1, Bob, A)$	$(O : \#4, S : \#2, P : \#8, G : \#SA, \epsilon : +)$
$(o1, Mary, A)$	$(O : \#4, S : \#3, P : \#8, G : \#SA, \epsilon : +)$
$(o1, Ann, W)$	$(O : \#4, S : \#1, P : \#9, G : \#SA, \epsilon : +)$
$(o2, Bob, W)$	$(O : \#5, S : \#2, P : \#9, G : \#SA, \epsilon : +)$
$(o3, Mary, W)$	$(O : \#6, S : \#3, P : \#9, G : \#SA, \epsilon : +)$

Fig. 11. Complete definition of a Bell and La Padula model instance (Model3 in the examples).

Model2

subject$(self : subject, name : string)$
user$(self : user)$
object$(self : object, name : string)$
role$(self : role, temp\_dis : string)$
privilege$(self : privilege, name : string)$
session$(self : session, name : string)$
PropDir$(R : role, P : privilege, direction : string)$
$Scheme_2^*($ **user**$) = (self : user, name : string)$
$Scheme_2^*($ **role**$) = (self : role, temp\_dis : string, name : string)$

EACMI

$user(self : \#1, name : Ann)$	$role(self : \#10, name : r1, blocked\_u : false)$
$user(self : \#2, name : Bob)$	$role(self : \#11, name : r2, blocked\_u : false)$
$user(self : \#3, name : Mary)$	$role(self : \#12, name : r3, blocked\_u : false)$
$object(self : \#4, name : o1)$	$LessR(R_1 : \#11(r2), R_2 : \#10(r1))$
$object(self : \#5, name : o2)$	$LessR(R_1 : \#12(r3), R_2 : \#10(r1))$
$object(self : \#6, name : o3)$	$Prop\_dir(R : \#11(r2), P : \#7(R), direction : up)$
$privilege(self : \#7, name : R)$	$Prop\_dir(R : \#12(r3), P : \#7(R), direction : up)$
$privilege(self : \#8, name : A)$	$Prop\_dir(R : \#10(r1), P : \#8(A), direction : down)$
$privilege(self : \#9, name : W)$	$ActiveRole(U : \#1(Ann), S : \#13(Ann), R : \#10(r1))$
$session(self : \#13, name : Ann)$	$ActiveRole(U : \#2(Bob), S : \#14(Bob), R : \#11(r2))$
$session(self : \#14, name : Bob)$	$ActiveRole(U : \#3(Mary), S : \#15(Mary), R : \#12(r3))$
$session(self : \#15, name : Mary)$	$Auth_d(O : \#4(o1), S : \#10(r1), P : \#7(R), G : \#SA, \epsilon : +)$
$subject(self : \#1, name : Ann)$	$Auth_d(O : \#5(o2), S : \#11(r2), P : \#7(R), G : \#SA, \epsilon : +)$
$subject(self : \#2, name : Bob)$	$Auth_d(O : \#6(o3), S : \#12(r3), P : \#7(R), G : \#SA, \epsilon : +)$
$subject(self : \#3, name : Mary)$	$Auth_d(O : \#4(o1), S : \#10(r1), P : \#8(A), G : \#SA, \epsilon : +)$
$subject(self : \#10, name : r1)$	$Auth_d(O : \#5(o2), S : \#11(r2), P : \#8(A), G : \#SA, \epsilon : +)$
$subject(self : \#11, name : r2)$	$Auth_d(O : \#6(o3), S : \#12(r3), P : \#8(A), G : \#SA, \epsilon : +)$
$subject(self : \#12, name : r3)$	$Auth_d(O : \#4(o1), S : \#10(r1), P : \#9(W), G : \#SA, \epsilon : +)$
$Play(U : \#1(Ann), R : \#10(r1))$	$Auth_d(O : \#5(o2), S : \#11(r2), P : \#9(W), G : \#SA, \epsilon : +)$
$Play(U : \#2(Bob), R : \#11(r2))$	$Auth_d(O : \#6(o3), S : \#12(r3), P : \#9(W), G : \#SA, \epsilon : +)$
$Play(U : \#3(Mary), R : \#12(r3))$	

IACMI

$r1 : InLessR(R_1 : X, R_2 : Y) \leftarrow LessR(R_1 : X, R_2 : Y)$
$r2 : InLessR(R_1 : X, R_2 : Y) \leftarrow LessR(R_1 : X, R_2 : Z), InLessR(R_1 : Z, R_2 : Y)$
$r3 : UserPlay(U : X, R : Y) \leftarrow Play(U : X, R : Y)$
$r4 : UserPlay(U : X, R : Y) \leftarrow Play(U : X, R : Z), InLessR(R_1 : Y, R_2 : Z)$
$r5 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : +, O' : X_5, S' : X_6, P' : X_7) \leftarrow Auth_p(O : X_1, S : X_8, P : X_3, G : X_4, \epsilon : +, O' : X_5, S' : X_6, P' : X_7), InLessR(R_1 : X_8, R_2 : X_2), PropDir(R : X_8, P : X_3, direction : up)$
$r6 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : +, O' : X_5, S' : X_6, P' : X_7) \leftarrow Auth_p(O : X_1, S : X_8, P : X_3, G : X_4, \epsilon : +, O' : X_5, S' : X_6, P' : X_7), InLessR(R_1 : X_2, R_2 : X_8), PropDir(R : X_8, P : X_3, direction : down)$
$r7 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8) \leftarrow Auth_p(O : X_1, S : X_9, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8), UserPlay(U : X_2, R : X_9), ActiveRole(U : X_2, S : X_{10}, R : X_9), role(self : X_9, name_r : X_{11}, temp\_dis : false)$
$r8 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_1, S' : X_2, P' : X_3) \leftarrow Auth_d(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5)$
$r9 : Auth(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5) \leftarrow Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8)$

Fig. 12. Complete definition of a role-based model instance (Model2 in the examples) - part A.

Model2
Authorization set

$(o1, r1, R)$	$(O : \#4, S : \#10, P : \#7, G : \#SA, \epsilon : +)$
$(o2, r2, R)$	$(O : \#5, S : \#11, P : \#7, G : \#SA, \epsilon : +)$
$(o3, r3, R)$	$(O : \#6, S : \#12, P : \#7, G : \#SA, \epsilon : +)$
$(o1, r1, A)$	$(O : \#4, S : \#10, P : \#8, G : \#SA, \epsilon : +)$
$(o2, r2, A)$	$(O : \#5, S : \#11, P : \#8, G : \#SA, \epsilon : +)$
$(o3, r3, A)$	$(O : \#6, S : \#12, P : \#8, G : \#SA, \epsilon : +)$
$(o1, r1, W)$	$(O : \#4, S : \#10, P : \#9, G : \#SA, \epsilon : +)$
$(o2, r2, W)$	$(O : \#5, S : \#11, P : \#9, G : \#SA, \epsilon : +)$
$(o3, r3, W)$	$(O : \#6, S : \#12, P : \#9, G : \#SA, \epsilon : +)$
$(o2, r1, R)$	$(O : \#5, S : \#11, P : \#7, G : \#SA, \epsilon : +)$
$(o3, r1, R)$	$(O : \#6, S : \#12, P : \#7, G : \#SA, \epsilon : +)$
$(o1, r2, A)$	$(O : \#4, S : \#10, P : \#8, G : \#SA, \epsilon : +)$
$(o1, r3, A)$	$(O : \#4, S : \#10, P : \#8, G : \#SA, \epsilon : +)$
$(o1, Ann, R)$	$(O : \#4, S : \#1, P : \#7, G : \#SA, \epsilon : +)$
$(o2, Bob, R)$	$(O : \#5, S : \#2, P : \#7, G : \#SA, \epsilon : +)$
$(o3, Mary, R)$	$(O : \#6, S : \#3, P : \#7, G : \#SA, \epsilon : +)$
$(o1, Ann, A)$	$(O : \#4, S : \#1, P : \#8, G : \#SA, \epsilon : +)$
$(o2, Bob, A)$	$(O : \#5, S : \#2, P : \#8, G : \#SA, \epsilon : +)$
$(o3, Mary, A)$	$(O : \#6, S : \#3, P : \#8, G : \#SA, \epsilon : +)$
$(o1, Ann, W)$	$(O : \#4, S : \#1, P : \#9, G : \#SA, \epsilon : +)$
$(o2, Bob, W)$	$(O : \#5, S : \#2, P : \#9, G : \#SA, \epsilon : +)$
$(o3, Mary, W)$	$(O : \#6, S : \#3, P : \#9, G : \#SA, \epsilon : +)$
$(o2, Ann, R)$	$(O : \#5, S : \#1, P : \#7, G : \#SA, \epsilon : +)$
$(o3, Ann, R)$	$(O : \#6, S : \#1, P : \#7, G : \#SA, \epsilon : +)$
$(o1, Bob, A)$	$(O : \#4, S : \#2, P : \#8, G : \#SA, \epsilon : +)$
$(o1, Mary, A)$	$(O : \#4, S : \#3, P : \#8, G : \#SA, \epsilon : +)$

Fig. 13. Complete definition of a role-based model instance (Model2 in the examples) - part B.

same information for common structural components. In this case, we say that the two instances are *compatible*. In order to formally define the concept of compatibility, we need to first define the projection of an EACMI over a given ACMS.

Definition 14 (*EACMI Projection*). Let E be an EACMI and let $S =< B, Scheme, A, ISA, Z >$ be an ACMS. The projection of E onto S, denoted by $\Pi_S(E)$, is the set $\{f(l_1 : v_1, \ldots, l_n : v_n) | f(l_1 : v_1, \ldots, l_n : v_n) \in E, f \in B, \{l_1, \ldots, l_n\} \in Scheme(f)\}$.

Compatibility can be now defined as follows.

Definition 15 (*Compatible EACMIs*). Let $S_1 =< B_1, Scheme_1, A_1, ISA_1, Z_1 >$ and $S_2 =< B_2, Scheme_2, A_2, ISA_2, Z_2 >$ be two ACMSs. Let $S =< B, Scheme, A, ISA, Z >$ be the schema constructed as follows:

— $B = B_1 \cap B_2$;
— $\forall b \in B_1 \cap B_2, Scheme(b) = Scheme_1(b) \cap Scheme_2(b)$
— $A = A_1 \cap A_2$;
— $ISA = ISA_1 = ISA_2$;
— $Z = Z_1 \cap Z_2$.

Model1

subject$(self : subject, name : string)$ **user**$(self : user)$
role$(self : role)$ **object**$(self : object, name : string)$
privilege$(self : privilege, name : string)$ **session**$(self : session, name : string)$
$Scheme_1^*($ **user**$) = (self : user, name : string)$ $Scheme_1^*($ **role**$) = (self : role, name : string)$

EACMI

$user(self : \#1, name : Ann)$ $user(self : \#2, name : Bob)$
$user(self : \#3, name : Mary)$ $object(self : \#4, name : o1)$
$object(self : \#5, name : o2)$ $object(self : \#6, name : o3)$
$privilege(self : \#7, name : R)$ $privilege(self : \#8, name : A)$
$privilege(self : \#9, name : W)$ $role(self : \#10, name : r1)$
$role(self : \#11, name : r2)$ $role(self : \#12, name : r3)$
$session(self : \#13, name : Ann)$ $session(self : \#14, name : Bob)$
$ActiveRole(U : \#1(Ann), S : \#13(Ann), R : \#10(r1))$ $session(self : \#15, name : Mary)$
$ActiveRole(U : \#2(Bob), S : \#14(Bob), R : \#11(r2))$ $LessR(R_1 : \#11(r2), R_2 : \#10(r1))$
$ActiveRole(U : \#3(Mary), S : \#15(Mary), R : \#12(r3))$ $LessR(R_1 : \#12(r3), R_2 : \#10(r1))$
$Auth_d(O : \#4(o1), S : \#10(r1), P : \#7(R), G : \#SA, \epsilon : +)$ $subject(self : \#1, name : Ann)$
$Auth_d(O : \#5(o2), S : \#11(r2), P : \#7(R), G : \#SA, \epsilon : +)$ $subject(self : \#2, name : Bob)$
$Auth_d(O : \#6(o3), S : \#12(r3), P : \#7(R), G : \#SA, \epsilon : +)$ $subject(self : \#3, name : Mary)$
$Auth_d(O : \#4(o1), S : \#10(r1), P : \#8(A), G : \#SA, \epsilon : +)$ $subject(self : \#10, name : r1)$
$Auth_d(O : \#5(o2), S : \#11(r2), P : \#8(A), G : \#SA, \epsilon : +)$ $subject(self : \#11, name : r2)$
$Auth_d(O : \#6(o3), S : \#12(r3), P : \#8(A), G : \#SA, \epsilon : +)$ $subject(self : \#12, name : r3)$
$Auth_d(O : \#4(o1), S : \#10(r1), P : \#9(W), G : \#SA, \epsilon : +)$ $Play(U : \#1(Ann), R : \#10(r1))$
$Auth_d(O : \#5(o2), S : \#11(r2), P : \#9(W), G : \#SA, \epsilon : +)$ $Play(U : \#2(Bob), R : \#11(r2))$
$Auth_d(O : \#6(o3), S : \#12(r3), P : \#9(W), G : \#SA, \epsilon : +)$ $Play(U : \#3(Mary), R : \#12(r3))$

IACMI

$r1 : InLessR(R_1 : X, R_2 : Y) \leftarrow LessR(R_1 : X, R_2 : Y)$
$r2 : InLessR(R_1 : X, R_2 : Y) \leftarrow LessR(R_1 : X, R_2 : Z), InLessR(R_1 : Z, R_2 : Y)$
$r3 : UserPlay(U : X, R : Y) \leftarrow Play(U : X, R : Y)$
$r4 : UserPlay(U : X, R : Y) \leftarrow Play(U : X, R : Z), InLessR(R_1 : Y, R_2 : Z)$
$r5 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : +, O' : X_5, S' : X_6, P' : X_7) \leftarrow Auth_p(O : X_1, S : X_8, P : X_3, G : X_4, \epsilon : +, O' : X_5, S' : X_6, P' : X_7), InLessR(R_1 : X_8, R_2 : X_2)$
$r6 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8) \leftarrow Auth_p(O : X_1, S : X_9, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8), UserPlay(U : X_2, R : X_9), ActiveRole(U : X_2, S : X_{10}, R : X_9)$
$r7 : Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_1, S' : X_2, P' : X_3) \leftarrow Auth_d(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5)$
$r8 : Auth(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5) \leftarrow Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8)$

Fig. 14. Complete definition of a role-based model instance (Model1 in the examples) - part A.

Let E_1 be an EACMI over S_1 and E_2 an EACMI over S_2. We say that E_1 and E_2 are compatible if and only if $\Pi_S(E_1) = \Pi_S(E_2)$.

Similarly to Definition 13, in order to introduce access subsumption/ equivalence, we consider three different cases.

Definition 16 (*Access Subsumption/Equivalence*). Let ac_1 and ac_2 be two access control models representable in our framework by two IACMIs I_1 and I_2 over two ACMSs S_1 and S_2, respectively. We say that:

—ac_1 is strongly access-subsumed by ac_2 (denoted by $ac_1 \subseteq_s^a ac_2$), if for each EACMI E_1 constructed over S_1 there exists an EACMI E_2 over S_2 such that E_1 and E_2 are compatible and $E_1 \cup I_1 \subseteq_s^{ia} E_2 \cup I_2$. ac_1 is strongly

Model1
Authorization set

$(o1, r1, R)$	$(O : \#4, S : \#10, P : \#7, G : \#SA, \epsilon : +)$
$(o2, r2, R)$	$(O : \#5, S : \#11, P : \#7, G : \#SA, \epsilon : +)$
$(o3, r3, R)$	$(O : \#6, S : \#12, P : \#7, G : \#SA, \epsilon : +)$
$(o1, r1, A)$	$(O : \#4, S : \#10, P : \#8, G : \#SA, \epsilon : +)$
$(o2, r2, A)$	$(O : \#5, S : \#11, P : \#8, G : \#SA, \epsilon : +)$
$(o3, r3, A)$	$(O : \#6, S : \#12, P : \#8, G : \#SA, \epsilon : +)$
$(o1, r1, W)$	$(O : \#4, S : \#10, P : \#9, G : \#SA, \epsilon : +)$
$(o2, r2, W)$	$(O : \#5, S : \#11, P : \#9, G : \#SA, \epsilon : +)$
$(o3, r3, W)$	$(O : \#6, S : \#12, P : \#9, G : \#SA, \epsilon : +)$
$(o2, r1, R)$	$(O : \#5, S : \#10, P : \#7, G : \#SA, \epsilon : +)$
$(o3, r1, R)$	$(O : \#6, S : \#10, P : \#7, G : \#SA, \epsilon : +)$
$(o2, r1, A)$	$(O : \#5, S : \#10, P : \#8, G : \#SA, \epsilon : +)$
$(o3, r1, A)$	$(O : \#6, S : \#10, P : \#8, G : \#SA, \epsilon : +)$
$(o2, r1, W)$	$(O : \#5, S : \#10, P : \#9, G : \#SA, \epsilon : +)$
$(o3, r1, W)$	$(O : \#6, S : \#10, P : \#9, G : \#SA, \epsilon : +)$
$(o1, Ann, R)$	$(O : \#4, S : \#1, P : \#7, G : \#SA, \epsilon : +)$
$(o2, Bob, R)$	$(O : \#5, S : \#2, P : \#7, G : \#SA, \epsilon : +)$
$(o3, Mary, R)$	$(O : \#6, S : \#3, P : \#7, G : \#SA, \epsilon : +)$
$(o1, Ann, A)$	$(O : \#4, S : \#1, P : \#8, G : \#SA, \epsilon : +)$
$(o2, Bob, A)$	$(O : \#5, S : \#2, P : \#8, G : \#SA, \epsilon : +)$
$(o3, Mary, A)$	$(O : \#6, S : \#3, P : \#8, G : \#SA, \epsilon : +)$
$(o1, Ann, W)$	$(O : \#4, S : \#1, P : \#9, G : \#SA, \epsilon : +)$
$(o2, Bob, W)$	$(O : \#5, S : \#2, P : \#9, G : \#SA, \epsilon : +)$
$(o3, Mary, W)$	$(O : \#6, S : \#3, P : \#9, G : \#SA, \epsilon : +)$
$(o2, Ann, R)$	$(O : \#5, S : \#1, P : \#7, G : \#SA, \epsilon : +)$
$(o3, Ann, R)$	$(O : \#6, S : \#1, P : \#7, G : \#SA, \epsilon : +)$
$(o2, Ann, A)$	$(O : \#5, S : \#1, P : \#8, G : \#SA, \epsilon : +)$
$(o3, Ann, A)$	$(O : \#6, S : \#1, P : \#8, G : \#SA, \epsilon : +)$
$(o2, Ann, W)$	$(O : \#5, S : \#1, P : \#9, G : \#SA, \epsilon : +)$
$(o3, Ann, W)$	$(O : \#6, S : \#1, P : \#9, G : \#SA, \epsilon : +)$

Fig. 15. Complete definition of a role-based model instance (Model1 in the examples) - part B.

access-equivalent to ac_2 (denoted by $ac_1 \equiv_s^a ac_2$) if both the following conditions hold: (i) $ac_1 \subseteq_s^a ac_2$; and (ii) $ac_2 \subseteq_s^a ac_1$.

— ac_1 is positively access-subsumed by ac_2 (denoted by $ac_1 \subseteq_p^a ac_2$), if, for each EACMI E_1 constructed over S_1 there exists an EACMI E_2 over S_2 such that E_1 and E_2 are compatible and $E_1 \cup I_1 \subseteq_p^{ia} E_2 \cup I_2$. ac_1 is positively access-equivalent to ac_2 (denoted by $ac_1 \equiv_p^a ac_2$) if both the following conditions hold: (i) $ac_1 \subseteq_p^a ac_2$; and (ii) $ac_2 \subseteq_p^a ac_1$.

— ac_1 is user-access subsumed by ac_2 (denoted by $ac_1 \subseteq_u^a ac_2$) if, for each EACMI E_1 constructed over S_1 there exists an EACMI E_2 over S_2 such that E_1 and E_2 are compatible and $E_1 \cup I_1 \subseteq_u^{ia} E_2 \cup I_2$. ac_1 is user-access equivalent to ac_2 (denoted by $ac_1 \equiv_u^a ac_2$) if both the following conditions hold: (i) $ac_1 \subseteq_u^a ac_2$; and (ii) $ac_2 \subseteq_u^a ac_1$.

The intuitive meaning of Definition 16 is to establish whether, independently from a specific extensional part, the accesses entailed by the first access control model can always be generated by an instance of the second access control model. In this case, the second model is more expressive than the first one.

Example 6. Consider the models presented in Example 4. We want to show that for each instance of Model1 there exists an instance of Model2 entailing the same set of authorizations. Given an EACMI E_1 for Model1, the corresponding instance E_2 for Model2 can be constructed by inserting in each fact for predicate `role` contained in E_1 a new attribute `temp_dis` set to `false`. Moreover, we must insert one fact for predicate `PropDir` for each combination of role/privilege, setting the value for attribute `direction` to `down`. It is easy to show that $E_1 \cup I_1 \equiv_s^a E_2 \cup I_2$, thus Model1 $\subseteq_s^a$ Model2. Note that the constructed instances are compatible since their contents coincide for the common structural information.

Informally, the obtained result means that Model2 is more expressive than Model1. Indeed, with Model2, we can disable the propagation from some role to users playing that role and we can decide the direction of the propagation. These operations are not allowed in Model1.

As an additional example of access subsumption, it can be proved that there exists a constrained RBAC access control model that strongly access-subsumes the Bell and La Padula model (see, for example, Sandhu [1996] for a sketch of the proof).

In the particular case in which $S_1 = S_2$, access subsumption/equivalence for access control models represented by our framework can be reduced to a logic programming problem, as stated by the following proposition.

PROPOSITION 1. *Let ac_1 and ac_2 be two access control models representable in our framework by two IACMIs I_1 and I_2 over an ACMS S. We say that:*

—*$ac_1 \subseteq_s^a ac_2$ if and only if $D(I_1) \subseteq_{Auth} D(I_2)$, where $\subseteq_{Auth}$ is the usual containment relationship between logic programs, considering $Auth$ as query predicate. $ac_1 \equiv_s^a ac_2$ holds if and only if $D(I_1) \equiv_{Auth} D(I_2)$, where $\equiv_{Auth}$ is the usual equivalence relationship between logic programs, considering $Auth$ as query predicate.*[14]

—*$ac_1 \subseteq_p^a ac_2$ $D'(I_1) \subseteq_{Auth^+} D'(I_2)$, where $\subseteq_{Auth^+}$ is the usual containment relationship between logic programs, considering $Auth^+$ as query predicate, and $D'(I) = D(I) \cup \{Auth^+(O : X_1, S : X_2, P : X_3, G : X_4) \leftarrow Auth(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : +)\}$. $ac_1 \equiv_s^a ac_2$ holds if and only if $D'(I_1) \equiv_{Auth^+} D'(I_2)$, where $\equiv_{Auth^+}$ is the usual equivalence relationship between logic programs, considering $Auth^+$ as query predicate.*

—*$ac_1 \subseteq_u^a ac_2$ if and only if $D''(I_1) \subseteq_{\overline{Auth}} D''(I_2)$, where $\subseteq_{\overline{Auth}}$ is the usual containment relationship between logic programs, considering $\overline{Auth}$ as query predicate, and $D''(I) = D(I) \cup \{\overline{Auth}(O : X_1, S : X_2, P : X_3, G : X_4) \leftarrow Auth(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : +), subject(self : X_2, \ldots))\}$. $ac_1 \equiv_u^a ac_2$ holds if and only if $D''(I_1) \equiv_{\overline{Auth}} D''(I_2)$, where $\equiv_{\overline{Auth}}$ is the usual equivalence relationship between logic programs, considering $\overline{Auth}$ as query predicate.*

[14]A logic program P_1 is contained in a logic program P_2 with respect to a query predicate q, denoted by $P_1 \subseteq_q P_2$, if for any set of facts F for extensional predicates, the solutions for q generated by $P_1 \cup F$ are contained in the solutions for q generated by $P_2 \cup F$. P_1 is equivalent to P_2 with respect to the query predicate q, denoted by $P_1 \equiv_q P_2$, if $P_1 \subseteq_q P_2$ and $P_2 \subseteq_q P_1$.

Table V. Decidability Results: SAT = Satisfiability, QR = Query Reachability, EQ = Equivalence, CN = Containment

Datalog restriction/extension	Sat/QR	EQ/CN
Project-join queries	decidable	decidable
Datalog rules	decidable	undecidable
Unary int. predicates	decidable	decidable
Unary int. predicates, negation on nonrecursive predicates, 1 recursive rule, $\neq$	undecidable	undecidable
Unary recursive int. pred., binary nonrecursive int. pred., negation on nonrecursive predicates, 1 recursive rule, no $\neq$	undecidable	undecidable
Binary int. pred.	decidable	undecidable
Dense order constraints	decidable	undecidable
Unary ext. pred., stratified negation	decidable	decidable
Dense order constraints, negation on ext. pred.	decidable	undecidable

Containment/equivalence of generic Datalog programs with negation is undecidable [Gaifman et al. 1987]. However, it is decidable under specific conditions, some of which are summarized in Table V, taken from Levy et al. [1993]. Since IACMIs in general do not satisfy any of the conditions listed in Table V, it follows that equivalence of access control models represented in our framework is in general undecidable. However, IACMIs have a well defined structure. Therefore, it is possible to identify specific classes of IACMIs for which subsumption and equivalence are decidable. For such classes of IACMIs, the proposed properties can be checked by using typical logic programming techniques. We leave to future work a comprehensive investigation of this problem.

7.1.3 *Constraint Containment/Equivalence.* Besides the set of entailed accesses and the structural components, another dimension by which access control models can be compared concerns the constraints enforced by the model. We say that two access control models are *constraint equivalent* if they enforce exactly the same constraints. An access control model is *constraint contained* into another access control model if the validity of the constraints enforced by the first model implies the validity of the constraints enforced by the second one. Formally, constraint containment/equivalence can be defined as follows.

Definition 17 (*Constraint Subsumption/Equivalence*). Let ac_1 and ac_2 be two access control models representable in our framework by two IACMIs I_1 and I_2 over two ACMSs S_1 and S_2, respectively. ac_1 is constraint subsumed by ac_2 (denoted by $ac_1 \subseteq^c ac_2$) if $D(I_1) \subseteq_{ErrorC} D(I_2)$. ac_1 is constraint equivalent to ac_2 (denoted by $ac_1 \equiv^c ac_2$) if the following conditions hold: (i) $ac_1 \subseteq^c ac_2$; and (ii) $ac_2 \subseteq^c ac_1$.

Also in this case, decidability of constraint subsumption/equivalence depends on the decidability of subsumption/equivalence of logic programs. For example, it is possible to prove that if rules defining predicate *ErrorC* in both programs do not depend on predicates $Auth, Auth_p$ and on recursive user-defined predicates, and do not use negation, constraint containment/equivalence is decidable.

7.2 Intra-Model Properties

Intra-model properties concern the analysis of the characteristics of a single access control model. In analyzing access control models, we have devised the following set of relevant properties:

—*Reachability*: by reachability we mean the ability to determine whether a certain authorization can be generated by a given access control model, possibly conditionally to the generation of another authorization. This property can be useful for determining dependencies existing among authorizations.
—*Consistency*: An access control model is consistent if it admits at least one instance that satisfies all the specified constraints, that is, it generates at least one consistent set of authorizations.

The previous properties can be formally defined in our framework as follows.

Definition 18 (*Reachability*). Let ac_1 be an access control model representable in our framework by an IACMI I_1 over an ACMS S. Atom $Auth_p(t_1, \ldots, t_m)$ is reachable from atom $Auth(t'_1, \ldots, t'_n)$ in $D(I_1)$ if there exists an EACMI E such that it is possible to derive $Auth(t'_1, \ldots, t'_n)$ from a derivation tree in which an instance of $Auth_p(t_1, \ldots, t_m)$ appears.

Reachability can help the security administrator in the specification of authorizations and in the maintenance of the system, since it is a means to determine the effect of an authorization rule on the state of the system. For instance, by using this property, the administrator is able to determine, whether: (i) a negative authorization can be derived from a positive (negative) authorization; (ii) a positive authorization can be derived from a positive (negative) authorization. This information is very useful, for instance, for checking dependencies existing among authorizations or for discovering the presence of conflicts.

Reachability is a property that has been investigated in the context of deductive databases. In general, reachability is decidable for Datalog programs, also with negation on extensional predicates. It is important to note that IACMIs, except for the rules defining $Auth_d$, are Datalog programs without negation. Thus, undecidability can only arise because of the structure of $Auth_d$ rules. Table V reports some decidability results, depending on the structure of the considered program. For IACMIs satisfying these properties, reachability is decidable. For example, reachability is decidable in the Bell and La Padula and NIST models, as stated by the following proposition.

PROPOSITION 2. *Reachability is decidable in the Bell and La Padula and NIST models.*

Example 7. Consider an RBAC model supporting the inheritance interpretation of role hierarchy and positive and negative authorizations. Moreover, suppose that positive authorizations for a given role are propagated to more powerful roles, whereas negative authorizations for a given role are propagated to less powerful roles. In this context, reachability results can be used to

answer the following questions:

i Is a negative authorization for a role r on a given object o for a given privilege p reachable from a positive authorization stating that the same role r is authorized for the same privilege on the same object?

From a formal point of view, this question corresponds to determining whether atom $Auth_p(O : o, S : r, P : p, G : G1, \epsilon : -, O' : X_1, S' : X_2, P' : X_3)$ is reachable from atom $Auth(O : o, S : r, P : p, G : G1, \epsilon : +)$, where $G1$ is a variable representing an arbitrary grantor, and X_1, X_2, X_3 are variables representing an arbitrary information source.

ii Is a negative authorization for a user u on a given object o for a given privilege p reachable from a negative authorization stating that the role r is denied for the privilege p on the object o?

From a formal point of view, this question corresponds to determining whether atom $Auth_p(O : o, S : u, P : p, G : G1, \epsilon : -, O' : X_1, S' : X_2, P' : X_3)$ is reachable from atom $Auth(O : o, S : r, P : p, G : G1, \epsilon : -)$, where $G1$ is a variable representing an arbitrary grantor, and X_1, X_2, X_3 are variables representing an arbitrary information source.

iii Is a negative authorization stating that user u_1 is denied for a given privilege p on a given object o reachable from a positive authorization stating that user u_2 is authorized for the same privilege on the same object?

From a formal point of view, this question corresponds to determining whether atom $Auth_p(O : o, S : u_1, P : p, G : G1, \epsilon : -, O' : X_1, S' : X_2, P' : X_3)$ is reachable from atom $Auth(O : o, S : u_2, P : p, G : G1, \epsilon : +)$, where $G1$ is a variable representing an arbitrary grantor, and X_1, X_2, X_3 are variables representing an arbitrary information source.

iv Is a positive authorization for a role r on a given object o for a given privilege p reachable from a positive authorization stating that a user u is authorized for the same privilege on the same object?

From a formal point of view, this question corresponds to determining whether atom $Auth_p(O : o, S : r, P : p, G : G1, \epsilon : +, O' : X_1, S' : X_2, P' : X_3)$ is reachable from $Auth(O : o, S : u, P : p, G : G1, \epsilon : +)$ atom , where $G1$ is a variable representing an arbitrary grantor, and X_1, X_2, X_3 are variables representing an arbitrary information source.

The ability to give an answer to the above questions supports the administrator in the creation and maintenance of a conflict resolution function according to the needed security requirements, and, in general, in the analysis of the model behavior. For instance, if the answer to question i) is yes, a conflict is generated; this information supports the administrator in the identification and resolution of conflicts.

As an example, consider the NIST Hierarchical access control model presented in Figures 14 and 15. The atom $(a) : Auth_p(O : \#5(o2), S : \#11(r2), P : \#7(R), G : \#SA, \epsilon : +, O' : \#5(o2), S' : \#11(r2), P' : \#7(R))$ is reachable from $(b) : Auth(O : \#5(o2), S : \#1(Ann), P : \#7(R), G : \#SA, \epsilon : +)$ since there exists a derivation tree for (b) in which an instance of (a) appears. Such a tree is presented in Figure 16. Note that:

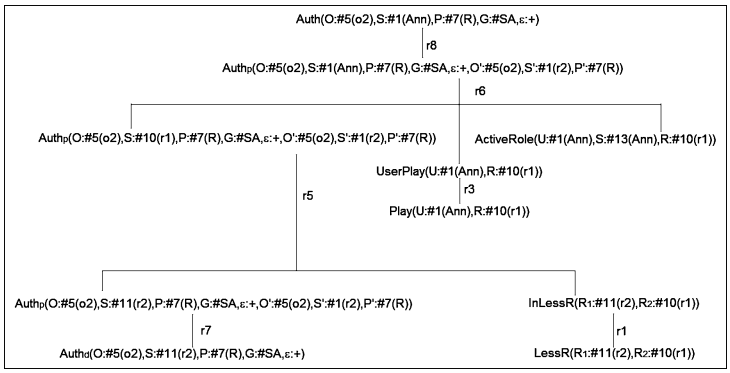


Fig. 16. A derivation tree.

— From $Auth_d(O : \#5(o2), S : \#11(r2), P : \#7(R), G : \#SA, \epsilon : +)$, by applying rule $r7$, we obtain (a).
— From (a) and rule $r5$ we obtain $(i) : Auth_p(O : \#5(o2), S : \#10(r1), P : \#7(R), G : \#SA, \epsilon : +, O' : \#5(o2), S' : \#11(r2), P' : \#7(R))$.
— From (i), by applying rule $r6$, we obtain $(ii) : Auth_p(O : \#5(o2), S : \#1(Ann), P : \#7(R), G : \#SA, \epsilon : +, O' : \#5(o2), S' : \#11(r2), P' : \#7(R))$.
— From (ii) and rule $r8$ we obtain (b).

Consistency can be formally defined in our framework as follows.

Definition 19 (*Consistency*). Let ac be an access control model representable in our framework by an IACMI I over an ACMS S. ac is consistent if there exists an EACMI E such that a consistent stable model exists for $I \cup E$.

It is simple to prove that consistency holds if $\{Error() \leftarrow\} \subseteq_{ErrorC} I$ is not true.

PROPOSITION 3. *Let ac be an access control model representable in our framework by an IACMI I over an ACMS S. ac is consistent iff* $\{Error() \leftarrow\} \subseteq_{ErrorC} I$ *does not hold.*

Example 8. Consider a NIST Constrained access control model supporting the inheritance interpretation of role hierarchy. Moreover, consider the following rules:

1. $ErrorC() \leftarrow ActiveRole(U : X, S : Y, R : Z), ActiveRole(U : X, S : Y, R : K)$;
2. $ErrorC() \leftarrow Auth(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : -)$;
3. $ErrorC() \leftarrow Auth(O : X_1(o1), S : X_2(r1), P : X_3(p1), G : \#SA, \epsilon : +), notAuth(O : X_4(o2), S : X_5(u2), P : X_6(p2), G : \#SA, \epsilon : -), object(self : X_1, name : o1), role(self : X_2, name : r1), privilege(self : X_3, name : p1), object(self : X_4, name : o2), user(self : X_5, name : u2), privilege(self : X_6, name : p2)$;

4. $Auth_d(O : X_1, S : X_2(r1), P : X_3(p1), G : \#4, \epsilon : +) \leftarrow object(self : X_1, name : X_2), role(self : X_2, name : r1), privilege(self : X_3, name : p1)$;

5. $p() \leftarrow object(self : X_1, name : o1)$;

6. $ErrorC() \leftarrow not\, p()$.

Rules 1,2,3, and 6 define constraints, rules 4 defines direct authorizations, whereas rule 5 defines a user-defined predicate, used in constraint 6.

Constraint 1 states that two distinct roles cannot be activated by the same user within the same session. An access control model containing just this constraint is certainly consistent, since there exists at least an EACMI that does not violate the constraint. For example, the EACMI of Model1 in Figure 14 does not violate the constraint since there does not exist a user that can activate more than a role within its session.

By contrast, consider a model in which rules 4, 5 and constraints 2,3, and 6 have been inserted. Rule 4 states that role $r1$ must be authorized to exercise privilege $p1$ on all objects, whereas rule 5 and constraint 6 state that object $o1$ must be always present. Constraint 3 states that an error occurs if role $r1$ is authorized to exercise privilege $p1$ on $o1$ and, at the same time, user $u2$ is not denied the exercise of privilege $p2$ on $o2$. Finally, constraint 2 states that only positive authorizations are admitted.

An access control model with rules 4, 5, and constraints 2,3, and 6 is not consistent because an *ErrorC* fact is always generated, regardless of the considered EACMI. In fact:

— if some negative authorizations are entailed by the model, an error is generated due to constraint 2;

— if $o1$ does not exist, an error is generated due to rule 5 and constraint 6;

— if $o1$ exists, role $r1$ is authorized to exercise privilege $p1$ on $o1$ by rule 4; in this case, if user $u2$ is not denied to exercise privilege $p2$ on $o2$, an error occurs due to constraint 3; otherwise he/she is denied but in this case, an error occurs due to constraint 2 since a negative authorization has been generated.

8. CONCLUSIONS

In this paper we have presented a formal framework for reasoning about access control models. The proposed framework is based on the C-Datalog language. The framework is general enough to model a large variety of access control models. In the paper, besides giving the syntax and the formal semantics of our framework, we have shown some examples of its applicability. Moreover, we have presented a set of dimensions for the analysis and the comparison of access control models and we have stated decidability results for them.

The framework we have introduced can be seen as a formal approach for analyzing and comparing existing access control models. In particular, we may assume that the user is assisted by a tool, automatizing the mapping process. The user will supply the model characteristics by using a GUI. The tool will then translate the specified information in an instance of the proposed framework, starting from which the analysis can be performed.

The work reported in this paper is a first step of a wider project we are currently working on. Future work includes the definition of a formal methodology for mapping access control models onto our framework, and a more comprehensive investigation of the proposed dimensions taking into account conflict resolution aspects. In this context we also plan to investigate the problem of safety verification in our framework. Another important extension concerns the definition of a notion of mapping complexity, that is, a measure of the effort required to map an access control model into our framework. We also plan to investigate the complexity of authorization administration and checking of the various models by using our framework as a formal basis. Such complexity measures represent the basis for criteria that can be used for the SA when having to choose among different models. Moreover, we plan to define specific algorithms supporting the detection of access equivalence relationships between access control models.

Additionally, we are developing, based on the proposed framework, a multipolicy system [Bertino et al. 2002]. This system will provide, among various features, a tool for access control model mapping and analysis. The system will also include tools for policy integration, in distributed heterogeneous systems, and policy evolution. Moreover, the system will provide: (i) a tool enabling the security administrator to specify and analyze access control policies, independently of any specific access control mechanism; and (ii) conversion tools for mapping these policies, once validated, onto access control mechanisms. We also plan to extend our framework for supporting administrative functions and we plan to compare this new framework with previous work on this topic [Ammann and Sandhu 1991; Sandhu 1992a; Sandhu 1992b; Sandhu and Ganta 1993]. Another planned extension concerns the representation of temporal access control (e.g. Bertino et al. [1998]) models inside our framework. In this respect we note that, given the characteristics of our framework, it will be quite easy to represent temporal constraints by making use of class attributes.

APPENDIX

A. FROM C-DATALOG TO DATALOG

In the following, we briefly survey how a C-Datalog program can be translated into an equivalent Datalog-like program. We refer the reader to Greco et al. [1992] for additional information.

The transformation involves three main steps: explicit representation of inheritance relationships, removal of labels, and representation of class terms. In the following, the previous steps are briefly described.

Instance inheritance. Inheritance relationships can be represented by inserting additional rules that make explicit the relations expressed by the ISA hierarchy. As an example, suppose that $a, b \in K$, and that b is a superclass of a. Let $Scheme^*(b) = \{b_1, \ldots, b_n\}$ and $Scheme^*(a) \setminus Scheme^*(b) = \{a_1, \ldots, a_m\}$. Then, the following rule captures the existing ISA relationship between a and b: $b(b_1 : X_1, \ldots, b_n : X_n) \leftarrow a(b_1 : X_1, \ldots, b_n : X_n, a_1 : X_{n+1}, \ldots, a_m : X_{n+m})$.

Labeling of terms. Let $a \in B$ be an entity name with schema closure $Scheme^*(b)$ of arity n. If we propose to fix an ordering for the n attributes of $Scheme^*(b)$ (with $self$ attribute in first position), we can then maintain the correspondence between terms and their labels even when removing the labels from atoms.

Class terms in atoms. Class terms must be extracted from atoms and represented as conjunctions of Datalog atoms. Suppose that $a(T)$ is an atom such that a class term $c(\ldots)$ occurs in T. We can replace $c(\ldots)$ in T with its first attribute (referring to self attribute), obtaining T', and replace atom $a(T)$ with the following conjunction of Datalog atoms: $a(T'), c(\ldots)$. The same procedure must be applied to each literal that belongs to the body of a rule with the following exception: if a rule contains a negated literal *not* $a(T)$ such that a class term $c(\ldots)$ occurs in T, we replace *not* $a(T)$ with *not* $a'(T')$, where $a'(T')$ is a new atom whose arguments are the variables of $a(T'), c(\ldots)$, defined by the rule $a'(T') \leftarrow a(T'), c(\ldots)$.

B. FORMAL PROOFS

THEOREM 1 . Let I_{ac} be an instance of the Bell and La Padula model and let $\mathcal{I}$ be the ACMI constructed as pointed out above. We have to show that I_{ac} and $\mathcal{I}$ agree. We first note that no constraints are enforced by the Bell and La Padula Model and no conflict can arise, since only positive authorizations are generated. Additionally, the grantor of each authorization is always the SA. Thus, we do not consider the grantor and the sign component of an authorization in the remainder of the proof. Moreover, only one set of authorizations A_1 is entailed by I_{ac} and, by construction, $D(\mathcal{I})$ admits a unique consistent stable model M. Based on the previous considerations we note that no conflict resolution policy is required for assigning a semantics to $D(\mathcal{I})$ and that $\mathcal{S}(D(\mathcal{I})) = \mathcal{GS}(D(\mathcal{I})) = M$. We have to show therefore that $M_{Auth} = A_1$, where A_1 is the set of authorizations entailed by I_{ac}. This corresponds to proving that $A_1 \subseteq M_{auth}$ and $M_{auth} \subseteq A_1$.

$A_1 \subseteq M_{auth}$ Valid authorizations in the Bell and La Padula model are all and only those that satisfy the simple security and *-Property.

Suppose that the authorization $\langle o, s, read \rangle \in A_1$ and it is entailed by the simple security property. It means that subject s has a **read** access on object o and thus the access class of s - say l_s - dominates the access class of o - say l_o -. By construction, $D(\mathcal{I})$ contains facts: $subject(self : \#s, name : s, access\_class : l_s)$, and $object(self : \#o, name : o, access\_class : l_o)$, where $\#s$ and $\#o$ are identifiers. Moreover, by construction, $l_s = l_o$ or $InLessC(C_1 : l_o, C_2 : l_s)$ holds. In the first case, from rule 3a), we deduce fact $Auth_d(O : \#o, S : \#s, P : \#read, G : \#SA, \epsilon : +)$, where $\#read$ is the identifier of the read privilege. In the second case, the same fact is deduced from rule 3b). Thus, $\langle \#o, \#s, \#read \rangle \in M_{Auth}$.

Now suppose that the authorization $\langle o, s, write \rangle \in A_1$ due to the *-Property. This means that subject s has a **write** access on object o and its access class l_s is equal to the access class l_o of the object. By construction, $D(\mathcal{I})$ contains facts: $subject(self : \#s, name : s, access\}class : l_s)$, and $object(self : \#o, name : o, access\_class : l_o)$, and $l_o = l_s$. Thus, from rule 3e), we deduce the fact

$Auth_d(O : \#o, S : \#s, P : \#write, G : \#SA, \epsilon : +)$, where $\#write$ is the identifier of the write privilege. Thus, $\langle \#o, \#s, \#write \rangle \in M_{Auth}$.

Finally, suppose that the authorization $\langle o, s, append \rangle \in A_1$ due to the *-Property. This means that subject s can exercise the **append** privilege on object o and its access class l_s is dominated by the access class of the object l_o. By construction, $D(\mathcal{I})$ contains facts: $subject(self : \#s, name : s, access\_class : l_s)$, and $object(self : \#o, name : o, access\_class : l_o)$. Moreover, by construction, $l_s = l_o$ or $InLessC(C_1 : l_o, C_2 : l_s)$ holds. In the first case, from rule 3c) we deduce the fact $Auth_d(O : \#o, S : \#s, P : \#append, G : \#SA, \epsilon : +)$, where $\#append$ is the identifier of the append privilege. In the second case, the same fact is deduced from rule 3d). Thus, $\langle \#o, \#s, \#append \rangle \in M_{Auth}$.

We can therefore conclude that $A_1 \subseteq M_{auth}$

$M_{auth} \subseteq A_1$. The thesis follows by applying a reasoning similar to that presented in the previous proof, by noting that in $\mathcal{I}$ we represent just the subjects (users or processes), objects, and privileges existing in I_{ac} and that dominance relationship among access classes is exactly represented by the definition of predicates $LessC$ and $InLessC$.

THEOREM 2. Let I_{ac} be an instance of the Flat RBAC model and let $\mathcal{I}$ be the ACMI constructed as shown in Section 6.2.1. We have to show that I_{ac} and $\mathcal{I}$ agree. We first note that no constraints are enforced by the Flat RBAC model and no conflict can arise, since only positive authorizations are generated. Additionally, the grantor of each authorization is always the SA. Thus, we do not consider the grantor and the sign component of an authorization in the remainder of the proof. Moreover, only one set of authorizations A_1 is entailed by the Flat RBAC instance and, by construction, $D(\mathcal{I})$ does not contain negation and admits a unique consistent stable model M. This means that no conflict resolution policy is required and that $\mathcal{S}(D(\mathcal{I})) = \mathcal{GS}(D(\mathcal{I})) = M$. We therefore have to show that $M_{Auth} = A_1$. This corresponds to proving that $A_1 \subseteq M_{auth}$ and $M_{auth} \subseteq A_1$.

$A_1 \subseteq M_{auth}$ An instance I_{ac} of the Flat RBAC model consists of the sets U, R, P, S and of the functions UA and PA defined over these sets. Set P consists of pairs (a, o), where $a \in A$ is an access model and $o \in O$ is an object.

Valid authorizations in the Flat RBAC model are all and only those of the form $\langle s, p \rangle$, $s \in U \cup R$, $p \in P$, that can be deduced from the user-role and the permission-role assignment functions. We suppose that $\langle s, p \rangle \in A_1$. Two cases arise depending on whether s is a user or a role.

By construction, $D(\mathcal{I})$ contains a fact for each element of the sets U, R, A, S, and O. Let us first suppose that $s = r, r \in R$. Since $\langle r, p \rangle \in A_1$, then $(p, r) \in PA$. Thus, by construction $D(\mathcal{I})$ contains the fact $Auth_d(O : \#p_o, S : \#r, P : \#p_a, G : \#SA, \epsilon : +)$, where $\#j$ represents the identifier of j. Thus, $\langle r, p \rangle \in M_{Auth}$.

Let us now suppose that $s = u, u \in U$. Since $\langle u, p \rangle \in A_1$, this means that both the following conditions hold: *i)* $(u, r) \in UA$; *ii)* $(p, r) \in PA$. By construction of AC, $D(\mathcal{I})$ contains the facts: $Play(U : \#u, R : \#r)$, $UserPlay(U : \#u, R : \#r)$, and $ActiveRole(U : \#u, S : \#s, R : \#r)$.

Additionally, since $(p, r) \in PA$, the fact $Auth_d(O : \#p_o, S : \#r, P : \#p_a, G : \#SA, \epsilon : +)$ belongs to $D(\mathcal{I})$. It is easy to show that by applying rules 8, 12 and

13, the fact $Auth(O : \#p_o, S : \#u, P : \#p_a, G : \#SA, \epsilon : +)$ is generated. Thus, $\langle u, p\rangle \in M_{Auth}$ which proves the thesis.

$M_{auth} \subseteq A_1$. The thesis follows by applying a reasoning similar to that presented in the previous proof, by noting that in $\mathcal{I}$ we represent just the subjects (users or roles), objects, and privileges existing in I_{ac} and that user-role and permission-role assignments are exactly represented by the definition of predicates $Play$, $ActiveRole$ and $Auth_d$.

THEOREM 3. Let I_{ac} be an instance of the Hierarchical RBAC model and let $D(\mathcal{I})$ be the ACMI constructed as shown in Section 6.2.2. We have to show that I_{ac} and $\mathcal{I}$ agree. We first note that no constraints are enforced by the Hierarchical RBAC model and no conflict can arise, since only positive authorizations are generated. Additionally, the grantor of each authorization is always the SA. Thus, we do not consider the grantor and the sign component of an authorization in the remainder of the proof. Moreover, only one set of authorizations A_1 is entailed by the Hierarchical RBAC instance and, by construction, $D(\mathcal{I})$ admits a unique consistent stable model M. This means that no conflict resolution policy is required and that $\mathcal{S}(D(\mathcal{I})) = \mathcal{GS}(D(\mathcal{I})) = M$. We have to show therefore that $M_{Auth} = A_1$. This corresponds to proving that $A_1 \subseteq M_{auth}$ and $M_{auth} \subseteq A_1$.

$A_1 \subseteq M_{auth}$ An instance I_{ac} of the Hierarchical RBAC model consists of sets U, R, P, S, of functions UA, and PA and of the role hierarchy RH. Set P consists of pairs (a, o), where $a \in A$ is an access mode, and $o \in O$ is an object. Valid authorizations in the Hierarchical RBAC model are all and only those of the form $\langle s, p\rangle$, $s \in U \cup R$, $p \in P$, that can be deduced from the user-role, the permission-role assignment functions, and the role hierarchy RH. We suppose that $\langle s, p\rangle \in A_1$. Two cases arise depending on whether s is a user or a role.

By construction, $D(\mathcal{I})$ contains a fact for each element of the sets U, R, A, P, S and O. Moreover, for each $\langle r_j, r_i\rangle \in RH$, $D(\mathcal{I})$ contains a fact $LessR(R_1 : \#r_j, R_2 : \#r_i)$, where $\#l$ represents the identifier of l. Let us first suppose that $s = r$, $r \in R$. Since $\langle r, p\rangle \in A_1$, then either *i)* $(p, r) \in PA$ or *ii)* $(p, r') \in PA$ such r' is a direct or indirect child of r in RH. Suppose first that *i)* holds. Thus, by construction $D(\mathcal{I})$ contains the fact $Auth_d(O : \#p_o, S : \#r, P : \#p_a, G : \#SA, \epsilon : +)$. Thus, $\langle r, p\rangle \in M_{Auth}$, which proves the thesis. Let us now suppose that *ii)* holds. Thus, by construction $D(\mathcal{I})$ contains the fact $Auth_d(O : \#p_o, S : \#r', P : \#p_a, G : \#SA, \epsilon : +)$. It is easy to show that by rules 12, 13, 5, fact $Auth(O : \#p_o, S : \#r, P : \#p_a, G : \#SA, \epsilon : +)$ is deduced. Thus, $\langle r, p\rangle \in M_{Auth}$, which proves the thesis.

Let us now suppose that $s = u$, $u \in U$. Since $\langle u, p\rangle \in A_1$, this means that both the following conditions hold: *i)* $(u, r) \in UA$ and *ii)* $(p, r') \in PA$ such that either *a)* $r = r'$ or *b)* the inheritance interpretation of RH is considered and r' is a child of r. By construction of AC, if $(u, r) \in UA$, then $D(\mathcal{I})$ contains the facts: $Play(U : \#u, R : \#r)$, $UserPlay(U : \#u, R : \#r)$, and $ActiveRole(U : \#u, S : \#, R : \#r)$. Let us first suppose that *a)* holds. Since by hypothesis $(p, r) \in PA$, the fact $Auth_d(O : \#p_o, S : \#r, P : \#p_a, G : \#SA, \epsilon : +)$ belongs to $D(\mathcal{I})$. It is easy to show that independently from which interpretation of the role hierarchy is

chosen, the fact $Auth(O : \#p_o, S : \#u, P : \#p_a, G : \#SA, \epsilon : +)$ is generated. Thus, $\langle u, p\rangle \in M_{Auth}$, which proves the thesis.

Suppose now that *b)* holds. Thus, $Auth_d(O : \#p_o, S : \#r', P : \#p_a, G : \#SA, \epsilon : +)$ belongs to $D(\mathcal{I})$, where r' is a child of r. It is easy to show that by rules 3a, 3b, 4a, 4b, 5, 8, 12 and 13, we deduce the fact $Auth(O : \#p_o, S : \#u, P : \#p_a, G : \#SA, \epsilon : +)$. Thus, $\langle u, o\rangle \in M_{Auth}$, which proves the thesis.

$M_{auth} \subseteq A_1$. The thesis follows by applying a reasoning similar to that presented in the previous proof, by noting that in $\mathcal{I}$ we represent just the subjects (users or roles), objects, and privileges existing in I_{ac} and that role hierarchy, user-role and permission-role assignments is exactly represented by the definition of predicates $LessR$, $InLessR$, $Play$, $ActiveRole$, and $Auth_d$.

PROPOSITION 2. The IACMs representing the Bell and La Padula, Flat RBAC, Hierarchical RBAC, and Constrained RBAC models are a Datalog program without negation. Based on Table V, reachability in such programs is decidable.

PROPOSITION 3. If ac_1 is consistent, there exists an EACMI E such that a consistent stable model exists for $I_1 \cup E$, that is, a model not containing any facts for predicate $ErrorC$. This also means that it is not true that, for each EACM E, no consistent stable model for $I_1 \cup E$ exists. This corresponds to saying that $\{Error() \leftarrow\} \subseteq_{ErrorC} I_1$ does not hold.

REFERENCES

ADAM, N., ATLURI, V., BERTINO, E., AND FERRARI, E. 2002. A Content-Based Authorization Model for Digital Libraries. *IEEE Trans. Knowl. Data Eng. 14*, 2 (March/April), 296–315.

AGG. See http://tfs.cs.tu-berlin.de/agg/docu.html.

AMMANN, P. AND SANDHU, R. 1991. Safety Analysis for the Extended Schematic Protection Model. In *Proceedings of the IEEE Symposium on Security and Privacy*. Oakland, California, 87–97.

ATLURI, V. AND HUANG, W. 2000. A Petri Net Based Safety Analysis of Workflow Authorization Models. *J. Comput. Secu. 8*, 2&3.

BELL, D. AND PADULA, L. L. 1975. Secure Computer Systems: Unified Exposition and Multics Interpretation. Tech. Rep. ESD-TR-75-306, Hanscom Air Force Base, Bedford, MA.

BERTINO, E., BETTINI, C., FERRARI, E., AND SAMARATI, P. 1998. An Access Control Mechanism Supporting Periodicity Constraints and Temporal Reasoning. *ACM Trans. Database Syst. 23*, 3, 231–285.

BERTINO, E., BUCCAFURRI, F., FERRARI, E., AND RULLO, P. 2000. A Logic-Based Approach for Enforcing Access Control. *J. Comput. Secu. 8*, 2&3.

BERTINO, E., CATANIA, B., FERRARI, E., AND PERLASCA, P. 2002. A System to Specify and Manage Multipolicy Access Control Models. In *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*.

BERTINO, E., FERRARI, E., AND ATLURI, V. 1999. The Specification and Enforcement of Authorization Constraints in Workflow Managenent Systems. *ACM Trans. Inform. Syst. Secu. 2*, 1, 65–104.

BERTINO, E., SAMARATI, P., AND JAJODIA, S. 1997. An Extended Authorization Model. *IEEE Trans. Knowl. Data Engi. 9*, 1 (January/February).

CASTANO, S., FUGINI, M., MARTELLA, G., AND SAMARATI, P. 1995. *Database Security*. Addison-Wesley.

CORAL. See ftp.cs.wisc.edu/coral/.

ECLiPSe. See http://www-icparc.doc.ic.ac.uk/eclipse/.

EHRIG, H., KREOWSKI, H., MONTANARI, U., AND ROZENBERG, G., Eds. 1999. *Handbook of Graph Grammars and Computing by Graph Transformation*. vol. 2 (Applications, Languages, and Tools). World Scientific.

FERNANDEZ, E., GUDES, E., AND SONG, H. 1994. A Model for Evaluation and Administration of Security in Object-Oriented Databases. *IEEE Trans. Knowl. Data Eng. 6*, 275–292.

FERRARI, E. AND THURAISINGHAM, B. 2000. Secure Database Systems. In *Advanced Databases: Technology and Design*, O. Diaz and M. Piattini, Eds. Artech House, London.

GAIFMAN, H., MAIRSON, H., SAGIV, Y., AND VARDI, M. 1987. Undecidable Optimization Problems in Database Logic Programs. In *Proceedings of the 2nd IEEE Symposium on Logic in Computer*. 106–115.

GLAUERT, J., KENNAWAY, R., AND SLEEP, R. 1991. DACTL: An Experimental Graph Rewriting Language. In *Proceedings of the 4th. International Workshop on Graph Grammars and their Application to Computer Science*, Springer-Verlag, Ed. vol. 532. 378–395.

GRECO, S., LEONE, N., AND RULLO, P. 1992. COMPLEX: An Object-Oriented Logic Programming System. *IEEE Trans. Knowl. Data Eng. 4*, 72–87.

HAAS, L., CHANG, W., AND LOHMAN, G. 1990. Starbust Mid-Flight: As the Dust Clears. *IEEE Trans. Knowl. Data Eng. 2*, 33–54.

JAEGER, T. AND TIDSWELL, J. 2001. Practical Safety in Flexible Access Control Models. *ACM Trans. Inform. Syst. Secu. 4*, 2 (May), 158–190.

JAJODIA, S., SAMARATI, P., SAPINO, M., AND SUBRAHMANIAN, V. 2001. Flexible Support for Multiple Access Control Policies. *ACM Trans. Database Syst. 26*, 2 (June), 214–260.

JAJODIA, S., SAMARATI, P., SUBRAHMANIAN, V., AND BERTINO, E. 1997. A Unified Framework for Enforcing Multiple Access Control Policies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 474–485.

KOCH, M., MANCINI, L., AND PARISI-PRESICCE, F. 2000. A Formal Model for Role-Based Access Control Using Graph Transformation. In *Proceedings of the 6th European Symposium on Research in Computer Security*. 122–139.

KOCH, M., MANCINI, L., AND PARISI-PRESICCE, F. 2001. On the Specification and Evolution of Access Control Policies. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT-01)*. Chantilly, Virginia, USA, 121–130.

LEVY, A., MUMICK, I., SAGIV, Y., AND SHMUELI, O. 1993. Equivalence, Query-Reachability, and Satisfiability in Datalog Extensions. In *Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. 109–122.

LLOYD, J. 1987. *Foundations of Logic Programming*. Springer-Verlag.

MILLEN, J. AND LUNT, T. 1992. Security for Object-Oriented Database Systems. In *Proceedings of the IEEE Symposium on Security and Privacy*. Oakland (Ca), USA, 260–272.

RABITTI, F., BERTINO, E., KIM, W., AND WOELK, D. 1991. A Model of Authorization for Next-Generation Database Systems. *ACM Trans. Database Syst. 16*, 1 (March), 88–131.

ROZENBERG, G., Ed. 1997. *Handbook of Graph Grammars and Computing by Graph Transformation*. vol. 1 (Foundations). World Scientific, Singapore.

SAMARATI, P., BERTINO, E., AND JAJODIA, S. 1996. An Authorization Model for a Distributed Hypertext System. *IEEE Trans. Knowl. Data Eng. 8*, 4 (August), 555–562.

SANDHU, R. 1992a. Expressive Power of the Schematic Protection Model. *J. Comput. Secu. 1*, 1.

SANDHU, R. 1992b. The Typed Access Matrix Model. In *Proceedings of the IEEE Symposium on Security and Privacy*. 122–136.

SANDHU, R. 1996. Role Hierarchies and Constraints for Lattice-based Access Controls. In *Computer Security - Esorics'96*, E. Bertino, H. Kurth, G. Martella, and E. Montolivo, Eds. Number 1146 in Lecture Notes in Computer Science. Rome, Italy, 65–79.

SANDHU, R., COYNE, E., FEINSTEIN, H., AND YOUMAN, C. 1996. Role-Based Access Control Models. *IEEE Comput. 29*, 2 (February), 38–47.

SANDHU, R., FERRAIOLO, D., AND KUHN, R. 2000. The NIST Model for Role-Based Access Control: Towards a Unified Standard. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control*. Berlin, Germany, 47–63.

SANDHU, R. AND GANTA, S. 1993. Expressive Power of the Single-Object Typed Access Matrix Model. In *Proceedings of the 9th Annual Computer Security Applications Conference*.

SCHURR, A. 1991. PROGRES: A VHL-language based on Graph Grammars. In *Proceedings of the 4th International Workshop on Graph Grammars and their Application to Computer Science*. Lecture Notes in Computer Science, vol. 532. Springer-Verlag, 641–659.

STRAWBERRY PROLOG. See http://www.dobrev.com/index.html.

THOMAS, R. AND SANDHU, R. 1997. Task-Based Authorization Controls (TBAC): Models for Active and Enterprise-Oriented Authorization Management. In *Proceedings of the 11th IFIP Working Conference on Database Security*. Lake Tahoe (CA), 136–151.

ULLMAN, J. 1989. *Principles of Database and Knowledge Base Systems*. vol. 1& 2. Computer Science Press.

WINSLETT, M., CHING, N., JONES, V., AND SLEPCHIN, I. 1997. Using Digital Credentials on the World Wide Web. *J. Comput. Secu. 5*, 3.

XSB. See http://xsb.sourceforge.net/.

Received October 2001; revised February 2002, March 2002,
April 2002, September 2002; accepted September 2002