

# Automated Analysis of Rule-Based Access Control Policies

Clara Bertolissi    Worachet Uttha

LIF, CNRS UMR 7279 & AMU

clara.bertolissi@lif.univ-mrs.fr, worachet.uttha@lif.univ-mrs.fr

## Abstract

In this paper we show how access control policies can be specified using a general metamodel whose operational semantics is based on term rewriting systems. The choice of the specification language aims at easing the verification task, since essential properties of access control (e.g. every request by an individual of accessing a resource always receives an answer, and this answer is unique) can be formalized and proved using rewriting techniques. We show that automated analysis of rewrite-based security policies can be done using the CiME rewriting tool which is able to produce mechanically checkable traces of security policy properties, for instance through the Coq proof assistant.

**Categories and Subject Descriptors** D.2.4 [Software Engineering]: Software/Program Verification; D.4.6 [Operating Systems]: Security and Protection - access controls; F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs - mechanical verification; F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages - program analysis

**General Terms** Verification, Theory, Security.

**Keywords** Security policies, access control, formal analysis, term rewriting.

## 1. Introduction

A security policy is a statement of what is, and what is not, allowed. When dealing with access control, a (formal) policy specification language will help to unambiguously define the rules that will govern the actions principals are allowed to execute over a set of resources. Nowadays, the generalized use of access control in distributed computing environments has increased the need for high-level declarative languages for the specification of complex policies. Security administrators need to specify a wide range of policies, to prove properties of these policies and to evaluate access requests efficiently. Among the different existing approaches, e.g. first-order theorem provers, purpose-built logics, or flow-analysis (see for instance [12, 15, 22, 26, 30]), rule-based policy specifications have the advantage to be concise and easy to maintain for security administrators. Those advantages are in great part due to the high level of abstraction of the languages used. Rewrite-based languages ensure clean and unambiguous semantics; more-

over, the declarative nature of this kind of policy specification enhances modularity, which is a crucial aspect when considering distributed security policies developed independently by different departments, organisations or institutions. Using the rewriting approach, policies are represented as sets of rewrite rules, whose evaluation produces authorization decisions, while requests are represented as algebraic terms. Thus, access request evaluation is effected by reducing terms to a *normal form*.

Another important reason to specify access control policies using rewrite-based frameworks is that rewriting tools and languages (such as MAUDE [27] and TOM [6] and CiME [28]), can be used to test, compare and experiment with evaluation strategies, to automate equational reasoning, and for the rapid prototyping of access control policies.

The main goals of the work presented in this paper are to design and characterize trusted policies, and to facilitate policy enforcement. The high level of abstraction of the specification language ensures that policy specification and enforcement can be separated from other functionalities of the system, thus avoiding bugs and increasing maintainability. We see access control as one aspect of the application, that can be specified, implemented and maintained independently. Once the rule-based policies are defined, they can be integrated into an implementation, using for instance the weaving techniques described in [31], or they can be used to guide the implementation of an access control system (in the same way as software specifications are used).

Moreover, specifying access control policies via term rewriting systems, which have a formal semantics, gives the possibility of proving properties of policies, and this is essential for policy acceptability [48]. For example, the absence of conflicts is an important property when both positive and negative authorizations are possible. It assures that for a certain access request no grant and denial are assigned at the same time. Rewriting properties like confluence (which implies the unicity of normal forms) and termination (which implies the existence of normal forms for all terms) may be used to demonstrate satisfaction of essential properties of policies, such as consistency.

We propose to security administrators a way to design expressive access control policies and verify desirable security properties by using automated security analysis techniques. It is important to remark that the security administrator does not need to learn a new specification language: the rewrite-based syntax is hidden behind a user-friendly interface and the rules for policy requirement specification are automatically generated by the application. Moreover, our approach facilitates the analysis of policies, since we integrate in the application the automated verification of access control properties via the rewrite tool CiME3. CiME3 performs checking of rewrite properties by discovering and moreover certifying, with full automation, termination and confluence proofs for term rewriting systems.

We develop a use case, an RBAC policy with dynamic roles [46] for a banking system, both in a centralized and in a distributed set-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PLPV'13, January 22, 2013, Rome, Italy.

Copyright © 2013 ACM 978-1-4503-1860-0/13/01. . \$15.00

ting (i.e. the considered bank organization is first composed by a unique central branch and later extended with several branches located in different sites). These policies are defined as particular instances of a common category-based metamodel [8], which relies on the notion of *category*, seen as a primitive concept. Different access control models are definable in the metamodel by interpreting classic types of grouping, like a role, a security clearance, a discrete measure of trust, etc., as particular instances of the more general notion of category. A rewrite-based operational semantics for this metamodel has been given in [17], where the expressive power of the metamodel is also demonstrated. We show here how automated analysis techniques can be applied to this kind of rule-based policies in order to increase the trust of the policy author on the correct behavior of the policy.

To the best of our knowledge, no automated verification results have been developed for rewrite-based access control policies as we do here. We believe this kind of analysis is essential in assisting designers to detect inconsistencies in their policy definition.

The rest of this paper is organized as follows. In Section 2, we give some details on term rewriting, the category-based access control metamodel, and a brief introduction to the tool CiME3. In Section 3 we develop a use case, an RBAC policy for a bank scenario. In Section 4 we show how we can use the rewrite tool CiME3 for automated verification of the policy and rapid testing. In Section 5 we extend the use case to a distributed scenario and we show how policy analysis can be performed in this setting. We conclude the paper in Section 7.

## 2. Preliminaries

In order to make this paper reasonably self-contained, we recall some basic notions and notations for first-order term rewriting, the category-based authorization meta-model and the rewrite tool CiME3 that will be used in the rest of the paper. We refer the reader to [4] and [8] for additional information.

### 2.1 Term Rewriting

A *signature*  $\mathcal{F}$  is a finite set of *function symbols*, each with a fixed arity.  $\mathcal{X}$  denotes a denumerable set of *variables*  $X_1, X_2, \dots$ . The set  $T(\mathcal{F}, \mathcal{X})$  of *terms* built up from  $\mathcal{F}$  and  $\mathcal{X}$  can be identified with the set of finite trees where each node is labelled by a symbol in  $\mathcal{F} \cup \mathcal{X}$  such that a node labelled by  $f \in \mathcal{F}$  must have a number of subtrees equal to the arity of  $f$ , and variables are only at the leaves. *Positions* are strings of positive integers denoting a path from the root to a node in the tree. The *subterm* of  $t$  at position  $p$  is denoted by  $t|_p$  and the result of replacing  $t|_p$  with  $u$  at position  $p$  in  $t$  is denoted by  $t[u]_p$ . This notation is also used to indicate that  $u$  is a subterm of  $t$ .  $\mathcal{V}(t)$  denotes the set of variables occurring in  $t$ . A term is *linear* if variables in  $\mathcal{V}(t)$  occur at most once in  $t$ . A term is *ground* if  $\mathcal{V}(t) = \emptyset$ . Substitutions are written as in  $\{X_1 \mapsto t_1, \dots, X_n \mapsto t_n\}$  where  $t_i$  is assumed to be different from the variable  $X_i$ . We use Greek letters for substitutions and postfix notation for their application.

**DEFINITION 1** (Rewrite step). *Given a signature  $\mathcal{F}$ , a term rewriting system on  $\mathcal{F}$  is a set of rewrite rules  $R = \{l_i \rightarrow r_i\}_{i \in I}$ , where  $l_i, r_i \in T(\mathcal{F}, \mathcal{X})$ ,  $l_i \notin \mathcal{X}$ , and  $\mathcal{V}(r_i) \subseteq \mathcal{V}(l_i)$ . A term  $t$  rewrites to a term  $u$  at position  $p$  with the rule  $l \rightarrow r$  and the substitution  $\sigma$ , written  $t \xrightarrow{l \rightarrow r}_p u$ , or simply  $t \rightarrow_R u$ , if  $t|_p = l\sigma$  and  $u = t[r\sigma]_p$ . Such a term  $t$  is called *reducible*. Irreducible terms are said to be in normal form.*

We denote by  $\rightarrow_R^+$  (resp.  $\rightarrow_R^*$ ) the transitive (resp. transitive and reflexive) closure of the rewrite relation  $\rightarrow_R$ . The subindex  $R$  will be omitted when it is clear from the context.

**EXAMPLE 1.** *Consider a signature for lists of natural numbers, with function symbols  $z$  (with arity 0) and  $s$  (with arity 1) to build numbers;  $nil$  (with arity 0) to denote an empty list,  $cons$  (with arity 2) and  $append$  (with arity 2) to construct and concatenate lists, respectively,  $\in$  (with arity 2) to test the membership of a natural number in a list. The list containing the numbers 0 and 1 is written then as  $cons(z, cons(s(z), nil))$ , or simply  $[z, s(z)]$  for short. We can specify list concatenation with the following rewrite rules:*

$$append(nil, X) \rightarrow X$$

$$append(cons(Y, X), Z) \rightarrow cons(Y, append(X, Z))$$

Then we have a reduction sequence:

$$append(cons(z, nil), cons(s(z), nil)) \rightarrow^*$$

$$cons(z, cons(s(z), nil))$$

*Boolean operators, such as disjunction, conjunction, and a conditional, can be specified using a signature that includes the constants True and False. For example, conjunction is defined by the rules  $and(True, X) \rightarrow X$ , and  $and(False, X) \rightarrow False$ . The notation  $t_1$  and  $\dots$  and  $t_n$  is syntactic sugar for  $and(\dots and(and(t_1, t_2), t_3) \dots)$ , and if  $b$  then  $s$  else  $t$  is syntactic sugar for the term if-then-else( $b, s, t$ ), with the rewrite rules: if-then-else(True,  $X, Y$ )  $\rightarrow X$  and if-then-else(False,  $X, Y$ )  $\rightarrow Y$ .*

*For example, we can define the membership operator " $\in$ " as follows:  $\in(X, nil) \rightarrow False$ ,  $\in(X, cons(H, L)) \rightarrow$  if  $X = H$  then True else  $\in(X, L)$ , where we assume " $=$ " is a syntactic equality test defined by standard rewrite rules. We will often write  $\in$  as an infix operator.*

Among the most important properties in term rewriting we have confluence and termination. A term rewriting system  $R$  is *confluent* if for all terms  $t, u, v$ :  $t \rightarrow^* u$  and  $t \rightarrow^* v$  implies  $u \rightarrow^* s$  and  $v \rightarrow^* s$ , for some  $s$ ; it is *terminating* if all reduction sequences are finite. Confluence and termination of rewriting are undecidable properties in general, but there are several results available in the literature that provide sufficient conditions for these properties to hold. For example, if all left-hand sides of rules in  $R$  are linear and there are no critical pairs (i.e., there are no superpositions of left-hand sides in the rules) then  $R$  is orthogonal. Orthogonality is a sufficient condition for confluence [43].

For the approach to distributed access control that we propose later, we use distributed term rewriting systems (DTRSs); DTRSs are term rewriting systems where rules are partitioned into modules, each associated with an identifier, and function symbols are annotated with such identifiers. We assume that each module has a unique identifier that is associated to the source of the definition of a function  $f$  (this can be a person, a site,  $\dots$ ). We say that a rule  $f(t_1, \dots, t_n) \rightarrow r$  defines  $f$ . There may be several rules defining  $f$ : for example, we may write  $f_\nu$  to indicate that the definition of the function symbol  $f$  is stored in the site  $\nu$ , where  $\nu$  is a site identifier. If a symbol is used in a rule without a site annotation, we assume the function is defined locally. For more details on Distributed Term Rewriting Systems, we refer the reader to [16].

### 2.2 The Category-Based Access Control Meta-Model

We briefly describe below the key concepts underlying the category-based metamodel of access control, henceforth denoted by  $\mathcal{M}$ . We refer the reader to [8] for a detailed description. We do not deal with authentication in this paper; we assume that principals that request access to resources are pre-authenticated.

Informally, a category is any of several distinct classes or groups to which entities may be assigned. Entities are denoted uniquely by constants in a many sorted domain of discourse, including:

- A countable set  $\mathcal{C}$  of *categories*, denoted  $c_0, c_1, \dots$
- A countable set  $\mathcal{P}$  of *principals*, denoted  $p_0, p_1, \dots$
- A countable set  $\mathcal{A}$  of named *actions*, denoted  $a_0, a_1, \dots$
- A countable set  $\mathcal{R}$  of *resource identifiers*, denoted  $r_0, \dots$
- A finite set  $\mathcal{Auth}$  of possible *answers* to access requests.
- A countable set  $\mathcal{S}$  of *situational identifiers*, denoted  $s_0, s_1, \dots$

Situational identifiers are used to denote contextual or environmental information e.g., locations, times, system states, etc. The precise set  $\mathcal{S}$  of situational identifiers that is admitted is application specific.

An important element in access control models is the request-response component. In the metamodel, the answer to a request may be one of a series of constants. For instance, the set  $\mathcal{Auth}$  might include {grant, deny, undetermined}.

In addition to the entities mentioned above, the metamodel includes the following relations that are of primary importance for the specification of access control policies:

- *Principal-category assignment*:  $\mathcal{PCA} \subseteq \mathcal{P} \times \mathcal{C}$ , such that  $(p, c) \in \mathcal{PCA}$  iff a principal  $p \in \mathcal{P}$  is assigned to the category  $c \in \mathcal{C}$ .
- *Permissions*:  $\mathcal{ARCA} \subseteq \mathcal{A} \times \mathcal{R} \times \mathcal{C}$ , such that  $(a, r, c) \in \mathcal{ARCA}$  iff the action  $a \in \mathcal{A}$  on resource  $r \in \mathcal{R}$  can be performed by principals assigned to the category  $c \in \mathcal{C}$ .
- *Authorizations*:  $\mathcal{PAR} \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{R}$ , such that  $(p, a, r) \in \mathcal{PAR}$  iff a principal  $p \in \mathcal{P}$  can perform the action  $a \in \mathcal{A}$  on the resource  $r \in \mathcal{R}$ .

Thus,  $\mathcal{PAR}$  defines the set of authorizations that hold according to an access control policy that specifies  $\mathcal{PCA}$  and  $\mathcal{ARCA}$ . In addition, for enhancing expressivity (e.g. for the modeling of 3-valued policies) we define a notion of forbidden operation on resources, modeled by the relation  $\mathcal{BARCA}$ , and a notion of non-authorized access, modeled by the relation  $\mathcal{BAR}$ :

- *Banned actions on resources*:  $\mathcal{BARCA} \subseteq \mathcal{A} \times \mathcal{R} \times \mathcal{C}$ , such that  $(a, r, c) \in \mathcal{BARCA}$  iff the action  $a \in \mathcal{A}$  on resource  $r \in \mathcal{R}$  is forbidden for principals assigned to the category  $c \in \mathcal{C}$ .
- *Banned access*:  $\mathcal{BAR} \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{R}$ , such that  $(p, a, r) \in \mathcal{BAR}$  iff performing the action  $a \in \mathcal{A}$  on the resource  $r \in \mathcal{R}$  is forbidden for the principal  $p \in \mathcal{P}$ .

Additionally, a relation  $\mathcal{UNDET}$  could be defined if  $\mathcal{PAR}$  and  $\mathcal{BAR}$  are not complete, i.e., if there are access requests that are neither authorized nor denied (thus producing an undetermined answer).

The relations satisfy the following core axioms,  $\forall p \in \mathcal{P}, \forall a \in \mathcal{A}, \forall r \in \mathcal{R}, \forall c \in \mathcal{C}$ :

- $(p, c) \in \mathcal{PCA} \wedge (a, r, c) \in \mathcal{ARCA} \Leftrightarrow (p, a, r) \in \mathcal{PAR}$
- $(p, c) \in \mathcal{PCA} \wedge (a, r, c) \in \mathcal{BARCA} \Leftrightarrow (p, a, r) \in \mathcal{BAR}$
- $(p, c) \in \mathcal{PCA} \wedge (a, r, c) \notin \mathcal{ARCA} \wedge (a, r, c) \notin \mathcal{BARCA} \Leftrightarrow (p, a, r) \in \mathcal{UNDET}$
- $\mathcal{PAR} \cap \mathcal{BAR} = \emptyset$

An inclusion relationship between categories (in the style of the RBAC role hierarchy), can be included in a generalized version of the axioms, as detailed in [17].

A range of access control models can be represented as specialized instances of this metamodel: see [17] for the specifications of traditional access control models, such as RBAC, DAC and MAC (including the well-known Bell-LaPadula model), as well as the Chinese Wall policy and the event-based model DEBAC [16].

### 2.3 A short introduction to CiME

CiME [50], developed within the A3PAT project, is an open source toolbox dedicated to the handling and analysis of rewriting programs. It allows one to define term algebras and rewriting systems, and to perform a range of treatments over them: computation, normalisation, matching and unification, completion and proofs of equality. An important part of CiME is dedicated to proofs of termination, local confluence and convergence. The last version of CiME, called CiME3, provides a certification mechanism which issues proof traces in XML format and certificates, e.g in the form of a Coq [3] script. The techniques for the generation of Coq scripts for certification relies on the Coccinelle library, which allows for deep and shallow embeddings of the theory of rewriting. For efficiency reasons, the termination engine may use an external SAT-solver to find termination orderings. For experimental results the reader can refer to [28].

## 3. CASE STUDY: a banking scenario using an RBAC policy with attributes

We have developed a case study for an access control policy in a banking scenario. We consider a three-valued role-based access control policy extended with attributes [46], that is a policy with the usual RBAC structure (permissions are granted to roles which are assigned to users) but with additional flexibility: user-roles assignments are based on a set of attributes related to the user (such as age, nationality, work experience, etc.). We are not interested here in establishing attribute certificates, we simply assume attribute values come from a trustworthy source. We specify this access control model by using an appropriate instantiation of the metamodel  $\mathcal{M}$ .

### 3.1 Use case description

The bank policy includes some roles, such as manager, banker, clerk and client and a set of permissions and prohibitions associated to each category:

- **Manager**: he/she can consult the account of any client; he/she can consult the loan list and the loan demand list.
- **Banker**: same as the manager, plus the rights of accepting or refusing a loan demand.
- **Clerk**: he/she can consult the account of any client and modify user data.
- **Client**: he/she can consult his/her account;
- **Gold-client**: same as client, plus the right of asking for a loan.

We have a similar description for prohibitions, for example a client cannot consult or modify the loan list and the loan demand list. The usefulness of explicit prohibitions will be clearer in Section 5, where we consider the combination of several access control policies in a distributed domain.

We have a set of registered users to which categories are assigned according to some attributes, such as age, diploma, work experience, qualifications, etc. For instance we may have

- **Manager**: a user is assigned to the category *manager* if he/she has a master's degree and a work experience of more than five years.
- **Gold-Client**: a user is assigned to the category *gold-client* if he/she is an adult (more than 20 years old) and he/she is not in the blacklist (due to a negative account balance for instance).

Given an access request from a registered user to perform an action  $a$  on a resource  $r$ , the bank policy will grant (resp. deny) the access if and only if the right of doing  $a$  on  $r$  belongs to the list of

permissions (resp. prohibitions) of a role which is assigned to the user.

### 3.2 Use case specification

We show how the bank policy access control requirements can be simply represented in terms of  $\mathcal{M}$ .

- We consider the categories manager, banker, clerk, gold-client and client.
- We have a list of principals such as Hertz Dupont, Thomas Durant, Gringo Joe, etc.
- Among the possible actions we have consult, modify, demand, accept or refuse.
- The set of resources contains accounts, loans, loan demand list and user data.
- The possible authorization decisions are grant, deny or undeterminate.

The category-based metamodel of access control is based on the core axioms given in Section 2. Operationally, these axioms can be realized through a set of function definitions, as shown in [17]. Recall that authorizations and prohibitions, defined by the relation  $\mathcal{PAR}$  and  $\mathcal{BAR}$ , are derived from the relations  $\mathcal{PCA}$ ,  $\mathcal{ARCA}$  and  $\mathcal{BARCA}$ . The information contained in such relations will be modeled by the functions  $pca$ ,  $arca$  and  $barca$ , respectively. The function  $arca$  returns a list of permissions assigned to a category, e.g.  $arca(\text{manager}) \rightarrow [(consult, account), (consult, loanList), (consult, loanDemands)]$ . Similarly, the function  $barca$  returns a list of prohibitions assigned to a category, e.g.  $barca(\text{manager}) \rightarrow [(accept, loan), (refuse, loan)]$ . The function  $pca$  returns the category assigned to a principal by exploiting the information associated to the principal's attributes, e.g.

$$pca(p) \rightarrow \text{if } age(p) > 20 \text{ and notBlacklisted}(p) \\ \text{then } [GoldClient] \text{ else } [Client]$$

The rewrite-based specification of the axioms in Section 2 is given by the rewrite rule:

$$\text{par}(p, a, r) \rightarrow \text{if } (a, r) \in arca(pca(p)) \text{ then grant} \\ \text{else if } (a, r) \in barca(pca(p)) \text{ then deny} \\ \text{else undeterminate}$$

where the function  $\in$  is a membership operator on lists (see Section 2). For optimization purposes, one can compose the standard list concatenation operator  $\text{append}$  with a function removing the duplicate elements in the list.

If for a given category  $c$ , a pair  $(a, r)$  is neither in  $arca(c)$  nor in  $barca(c)$ , then such access privilege is undefined. This permits a finer-grained evaluation of access requests: the constant undeterminate is a possible answer, at the same level as grant and deny. This is particularly important in open distributed policies, as we will see in Section 5.

It can be shown (cf. [17]) that the rewrite-based definition of  $\text{par}$  is a correct realization of the axioms defining  $\mathcal{M}$ .

An access request by a principal  $p$  to perform the action  $a$  on the resource  $r$  can then be evaluated simply by rewriting the term  $\text{par}(p, a, r)$  to normal form. For instance, assuming the principal Gringo Joe is associated to the manager category, we have:

$$\text{par}(\text{GringoJoe}, \text{consult}, \text{loanList}) \\ \rightarrow \text{if } (\text{consult}, \text{loanList}) \in arca(pca(\text{GringoJoe})) \\ \text{then grant} \\ \text{else if } (\text{consult}, \text{loanList}) \in barca(pca(\text{GringoJoe})) \\ \text{then deny else undeterminate}$$

$$\rightarrow^* \text{if } (\text{consult}, \text{loanList}) \in arca(\text{manager}) \\ \text{then grant else } \dots \\ \rightarrow^* \text{if } (\text{consult}, \text{loanList}) \in [\dots, (\text{consult}, \text{loanList}), \dots] \\ \text{then grant else } \dots \\ \rightarrow^* \text{grant}$$

### 3.3 Use case implementation

The rewrite rules provide an executable specification of the policy (the rewrite rules can be seen both as a specification *and* an implementation of the access control functions in a declarative language). However, in order to help security administrators to define such a kind of access control policies, we have implemented a user-friendly Java interface where categories, permissions and prohibitions can be specified, together with the principal category assignments and the category permissions and prohibitions assignments. The set of principals and their attributes are stored in a relational database accessible via the JDBC API from the application. Using our prototype, a security administrator can create a policy and make queries on it by filling blanks e.g. in permission/prohibition tables or in if-then-else English sentences. Suggestions for possible fillings appear automatically in drop-down lists exploiting the database information: for instance the policy writer may choose a (set of) user(s) and the list of available RBAC attributes for such user(s) will be automatically displayed. We do not give here a complete description of the interface, which is under continual refinement. The reader can find some screen shots at [20]. Once the policy has been specified via the interface, a translation from the fields entered by the user to the corresponding rewrite rules set is automatically generated and stored in a file *policy.trs*. The file is then passed to the rewrite-based tool CiME in order to perform policy analysis (see next section for details on the properties that can be checked). A prototype of this implementation for the banking use case is available at [20]. The *.trs* file containing the bank access control policy includes rules for the principal category assignments and for the category permission and prohibition assignments (specified via the interface), the principals relevant information (retrieved from the user database) and additional rules for algebraic and arithmetic operators as well as functions dealing with manipulations of data structures (included by default in the file).

The application graphic interface has been designed to make reading and definition of authorizations as clear as possible (by interacting with a database and using parametric drop-down lists) making it easier for administrators to specify and update access control requirements. Moreover, since the policy implementation follows the metamodel general schema, the prototype provides an intuitive means of specifying a wide range of policies. Indeed, different access control models can be understood in category-based terms (e.g. variants of RBAC, including hierarchical, time and location aspects, as well as lattice-based models such as the Bell-Lapadula and the McLean models can be specified).

After having specified the policy, the security administrator can choose via the interface which security property he wants to test. The translation of the specification into the rule-based syntax is executed in a transparent way and the tool CiME3 is launched in order to perform automated analysis. The results obtained on the rewrite systems are then translated into natural language using security policy terms to ease the readability for the policy designer. For some properties (such as termination), if the tests are successful, CiME3 is able to provide a formal certification by producing a trace of the proof in the form of a Coq certificate, which guarantees high-assurance checking.

## 4. Automated policy analysis

The rewrite-based specification of access control policies allows one to use automated analysis techniques in order to prove essential properties of the policies.

In this section we first define the security properties we consider and then we exemplify how verification of these properties can be performed by the CiME3 rewrite tool.

### 4.1 Properties of policies

An access control policy must satisfy certain criteria to be trustable. For example, it should not specify that any user is granted and denied the same access privilege on the same resource (i.e., the policy should be consistent). More precisely, we are interested in the following properties:

- *Totality*: Each access request from a valid user  $p \in \mathcal{P}$  to perform a valid action  $a \in \mathcal{A}$  on the resource  $r \in \mathcal{R}$  receives an answer (e.g. grant, deny or undeterminate).
- *Consistency*: For any  $p \in \mathcal{P}$ ,  $a \in \mathcal{A}$ ,  $r \in \mathcal{R}$ , it is not possible to derive more than one result for a request from  $p$  to access  $r$  for  $a$ .
- *Soundness and Completeness*: For any  $p \in \mathcal{P}$ ,  $a \in \mathcal{A}$ ,  $r \in \mathcal{R}$ , an access request by  $p$  to perform the action  $a$  on  $r$  is granted (resp. denied) if and only if  $p$  belongs to a category that has the permission (resp. prohibition)  $(a, r)$ .

A key observation is that, if the rewriting system defining a policy is terminating and confluent, we can deduce that each access request has a result which is unique. This is a consequence of the fact that in a terminating system each term has a normal form (e.g. grant or deny) and in a confluent system this normal form is unique. Thus, totality and consistency can be proved, for access control policies defined as term rewriting systems, by checking that the generated rewrite relation is confluent and terminating. The soundness and completeness of a policy can be checked by analyzing the normal forms of access requests. In addition, sufficient completeness of the rewrite rules (a property that ensures that each operation is defined on all valid inputs) may be considered in order to guarantee that the normal forms are of the right form [23, 38]. This kind of property can be checked using narrowing-based techniques (also mentioned in Section 7).

Confluence and termination of rewriting are undecidable properties in general, but there are several techniques available that ensures these properties to hold. By exploiting such techniques, CiME3 is able to check and moreover certify termination and local confluence on a large set of problems (see results obtained from the termination problem database on CiME website [50]). For defining a term rewrite system, one needs to define a term algebra signature and term rewrite rules on this algebra. Termination proofs for the rewrite system may then be obtained using various criteria, such as dependency pairs and graphs refinements, as well as various orderings, such as polynomial interpretations and recursive path ordering. Local confluence is then proved by showing that in particular each critical pair is joinable: the rewrite engine tries to normalize each member of the pair and to show that both reduce to the same term. Certification of proofs may be obtained, by an application of Newman's lemma [47].

### 4.2 Automated analysis

It is important to note that the proofs of the properties above do not have to be generated by a security administrator. Automated policy verification can be launched via the application interface. The obtained results (e.g. on confluence and termination of the rewrite system) will then be interpreted in access control terms and displayed back to the policy designer (resulting in a message like

"the analysis of the policy specification is successful: the policy is consistent"). It is worth noticing that results on confluence and termination are based on the structure of the rewrite rules defining the policy. This means that once the analysis has been done, it can be considered valid for a certain range of data. For example, if we consider the *pca* rule in Section 3.2, it is clear that attribute *age* of principal  $p$  can vary in the interval  $[0 - 20]$  without affecting the result of the derivation (i.e. the assignment of the category Client to  $p$ ). Following this reasoning, by examining the set of rewrite rules (in particular the right-hand sides of conditional rules) we may deduce a set of data for which the result of the analysis is guaranteed. However, this process of parametrization is difficult to automatize and is not pursued here.

Termination verification is done by executing the following program:

```
let R_trs_variable = variables "p,a,r,c,..." ;
let R_trs_signature =
  signature "pca : 1; arca : 1; par : 3;
  isblacklist : 1; ..." ;
let R_trs_algebra = algebra R_trs_signature ;
let R_trs =
  trs R_trs_algebra "
    barca(banker) -> cons(modify-data, nil);
    arca(banker) -> ...
    ...rules definition...
  " ;

termination R_trs;
```

`R_trs` contains the definition of the set of variables  $\mathcal{X}$ , the signature  $\mathcal{F}$ , the generated term algebra  $T(\mathcal{F}, \mathcal{X})$  and the rewrite rules defining the access control policy. This specification is directly generated by CiME3 from the *policy.trs* file received from the Java program and is ready for execution. The execution gives as a result:

```
termination R_trs;
...
- : bool = true
```

informing the user that a termination proof is found. We proceed similarly for confluence:

```
local_confluence R_trs;
...
- : bool = true
```

If both properties are satisfied, as it is the case for our case study, then the access control policy is proved to be total and consistent. Otherwise, the cause of the inconsistency can be detected by CiME:

```
local_confluence R_trs;
...
The rule [39] arca(manager) overlaps the rule
[48] arca(manager) at position (epsilon)
yielding the non-joinable critical pair P=...
- : bool = false
```

In this example, we can see that two different lists of permissions are associated to the same category manager, eventually yielding to inconsistent access authorizations. The policy designer can use this information for performing the necessary modifications to make the policy definition correct.

The evaluation of access authorization (available via the interface, by specifying the principal requesting the access, the action and the resource) can be performed by CiME3 simply by computing the normal form of an initial term representing the access request.

```
let t = term R_trs_algebra
  "par(GringoJoe, consult, loanList)";
```

```

normalize R_trs_t;
...
- : R_trs_signature term = grant

```

In this case, the request of principal Gringo Joe to consult the loan list is granted, according to the bank access control policy specification. Other useful tests can be automatically performed using the rewrite tool. For example, in the policy specification, lists of permissions and prohibitions can be specified for all the categories. In this context, conflicts may arise among rules that assign to a category a permission and at the same time the negated permission. In order to avoid this kind of problem, automatic tests can be generated by the application. This is done by checking for every category that the list of permissions *arca* and the list of prohibitions *barca* are disjoint. If a conflict arises, the pair(s) (action, resource) generating the conflict is (are) returned. For example:

```

let t = term R_trs_algebra
  "inter(arca(manager),barca(manager))";
normalize R_trs_t;
...
- : R_trs_signature term = nil

```

Here the test computes an empty final term, meaning that no conflicts arise for the manager category. Instead we may have:

```

let t = term R_trs_algebra
  "inter(arca(banker),barca(banker))";
normalize R_trs_t;
...
- : R_trs_signature term = cons(consult-loanList,nil)

```

Here we can see that the banker category has a conflicting permission concerning the action of consulting the loan list. Therefore, the output of the analysis tool is rich enough to provide useful diagnostic information to a policy writer or system engineer. Tests for all categories can be automatically generated by the application and results obtained via CiME3 are not difficult to understand for a security administrator (results are a list of conflicts of the form action-resource). Also soundness and completeness can be tested, by analyzing the normal forms computed by CiME3 on access request terms of the form  $\text{par}(p, a, r)$ , for any  $p \in \mathcal{P}$ ,  $a \in \mathcal{A}$ ,  $r \in \mathcal{R}$ . A possible alternative would be to use narrowing techniques, as hinted in Section 7.

The designer of the access control policy can thus benefit from a range of tests performed by the rewrite tool and relying on formal analysis techniques. A library of tests for the banking use case is available at [20].

## 5. Distributed access control

An important aspect when dealing with distributed applications is the capability of representing systems where resources may be dispersed across different sites. The information needed to decide whether a user request is granted or denied may also be distributed. Moreover, a conflict resolution mechanisms has to be specified in order to compose locally specified access control policies and compute a global access authorization decision. As we will see in this section, these distributed features can be easily accommodated in the metamodel  $\mathcal{M}$ .

### 5.1 Composition of access control policies in $\mathcal{M}$

In order to represent a distributed environment, the set of situational identifiers of  $\mathcal{M}$  will now include identifiers for sites (or locations) which will be associated to resources or policies. Each  $s \in \mathcal{S}$  identifies one of the components of the distributed system, seen as a federation [33]. For simplicity, we assume that the sets  $\mathcal{P}$ ,  $\mathcal{C}$ ,  $\mathcal{A}$ ,  $\mathcal{R}$  are globally known in the federation (an alternative would be to define

sets  $\mathcal{P}_s, \mathcal{C}_s, \mathcal{A}_s, \mathcal{R}_s$  for each site). To take into account the fact that the system may be composed of several sites, with different policies in place at each site, we consider families of relations  $\mathcal{PCA}_s, \mathcal{ARCA}_s, \mathcal{BARCA}_s, \mathcal{BAR}_s, \mathcal{UNDET}_s$  and  $\mathcal{PAR}_s$  indexed by site identifiers. Intuitively,  $\mathcal{PAR}_s$  (resp.  $\mathcal{BAR}_s$ ) denotes the authorizations (resp. prohibitions) that are valid in the site  $s$ . These notions are essential for integrating partially specified policies, i.e. policies that may be "not applicable" to requests on resources that are out of their jurisdiction. The relation  $\mathcal{PAR}$  defining the global authorization policy will be obtained by composing the local policies defined by the relations  $\mathcal{PAR}_s$  and  $\mathcal{BAR}_s$ . The axioms of the distributed metamodel can be seen as an extension to multiple sites of the axioms that define  $\mathcal{M}$  (see Section 2) where relations  $\mathcal{BAR}$ ,  $\mathcal{PAR}$  etc. are replaced by their local versions  $\mathcal{BAR}_s$ ,  $\mathcal{PAR}_s$ , etc. (the complete list of axioms is given in [18]). We give below the axiom describing the global authorization relation, which is obtained by combining the local authorizations and prohibitions defined at each site, by using an operator  $\mathcal{OP}$ . This operator is applied to the set of answers returned from the local sites and can be specified according to a particular application: it can give priority to positive authorisations, or to negative ones in case of conflict<sup>1</sup>. We assume that, at the global level, if the collected information is not sufficient for granting a request, then the access is denied (close policy approach).

$$\begin{aligned}
& \forall p \in \mathcal{P}, \forall a \in \mathcal{A}, \forall r \in \mathcal{R}, \\
& (p, a, r) \in \mathcal{OP}(\{\mathcal{PAR}_s, \mathcal{BAR}_s \mid s \in \mathcal{S}\}) \\
& \Leftrightarrow (p, a, r) \in \mathcal{PAR}
\end{aligned}$$

According to the axiom, the result of an access request may be different depending on the site where the request is evaluated, since each site has its own authorization policy defined by the local relations  $\mathcal{PAR}_s$  and  $\mathcal{BAR}_s$ . The combination operator  $\mathcal{OP}$  specifies the way final authorization decisions are computed (e.g. following the *deny-takes-precedence*, or the *most-specific-takes-precedence* principles). We refer the reader to [18] for the definition of a variety of operators that can be accommodated in the distributed version of  $\mathcal{M}$ . We exemplify next the operational semantics of the distributed metamodel by applying it to a specific scenario, extending the use case defined in Section 3.

### 5.2 Case study extended to a distributed scenario

We consider now a distributed bank organization composed of several branches, among which a main branch which has the role of head office. Clients can perform actions on their accounts, requests of loans, and so on. We suppose that bank loans are not dealt with in local branches, but are under the domain of the head office. The bank's access control general policy gives priority to local branch policies and transfers evaluation to the main site only if the request cannot be treated locally.

We have extended the use case scenario of Section 3 by considering a bank organization with two branches: a local branch  $l$  and a main branch  $c$ . The local policy in  $l$  is a three-valued RBAC policy with attributes as defined in Section 3, with the exception of manager and banker privileges on loans that will be now under the jurisdiction of the head office policy. The policy in place at site  $c$  is a two-valued RBAC policy (we assume that the main site denies accesses that are not explicitly permitted, i.e. it specifies a closed policy). We may notice that any category-based access control model can be accommodated in any site in a similar way.

<sup>1</sup>Intuitively, we may define an "intersection" operator corresponding to a policy which will grant the access only if all local policies grant it, or a "union" operator which gives as result grant if at least one local policy grants the access.

The functions  $\text{par}$  in the site  $l$  is defined as follows:

$$\text{par}_l(p, a, r) \rightarrow \begin{cases} \text{if } (a, r) \in \text{arca}_l(\text{pca}_l(p)) \text{ then grant} \\ \text{else if } (a, r) \in \text{barca}_l(\text{pca}_l(p)) \text{ then deny} \\ \text{else undeterminate} \end{cases}$$

where  $\text{arca}_l$ ,  $\text{pca}_l$  and  $\text{barca}_l$  are defined for the categories clerk and client as the corresponding functions in Section 3.

The functions  $\text{par}$  in the site  $c$  is defined as follows:

$$\text{par}_c(p, a, r) \rightarrow \begin{cases} \text{if } (a, r) \in \text{arca}_c(\text{pca}_c(p)) \text{ then grant} \\ \text{else deny} \end{cases}$$

where  $\text{arca}_c$  and  $\text{pca}_c$  are defined for the categories banker and manager as the corresponding functions in Section 3.<sup>2</sup>

Following this specification, the bank security administrator defines locally the rights for clients and clerks, and specifies the permissions concerning loans (associated to bankers and managers) in the main branch. In this scenario, combinations of access authorizations can be dealt with using the precedence operator  $\text{po}$ , as defined e.g. in [21]: if the information available in a site  $s$  is not sufficient to grant a principal the right to access a certain resource, but does not ban the access either (i.e., the outcome of the request evaluation cannot be determined at  $s$ ), then the access request is processed in another site of the system.

A new function  $\text{authorize}$  is needed in order to combine results obtained in the different sites into a final access authorization decision. For our specific bank scenario, we choose the following definition:

$$\text{authorize}(p, a, r) \rightarrow \text{combine}_{\text{po}}(\text{par}_l(p, a, r), \text{par}_c(p, a, r))$$

together with the definition of the  $\text{combine}_{\text{po}}$  function:

$$\begin{aligned} \text{combine}_{\text{po}}(\text{deny}, x) &\rightarrow \text{deny} \\ \text{combine}_{\text{po}}(\text{grant}, x) &\rightarrow \text{grant} \\ \text{combine}_{\text{po}}(\text{undeterminate}, x) &\rightarrow x \end{aligned}$$

In this case priority is given to local evaluation in site  $l$ , external evaluation in site  $c$  being executed only when the local policy in  $l$  is not able to give as an answer grant or deny. This corresponds to an implementation of the axiom in Section 5.1 for a distributed system composed of two sites  $l, c \in \mathcal{S}$  and the operator  $\mathcal{OP}$  instantiated by the precedence operator  $\text{po}$ . Notice that conflicts that may arise between the local and the central policy are automatically solved at run-time according to the operational semantics specified for the chosen combination operator. A more general definition using  $n$ -ary operators for evaluating combinations of answers from  $n$  different local policies can be specified. We refer the reader to [18] for additional information.

### 5.3 Distributed policy analysis

We implemented this distributed scenario and used the rewrite tool CiME3 for testing the security properties of the system. The generated input file contains the rules defining the policy in site  $l$  (annotated with  $l$ ), together with the rules defining the policy in site  $c$  (annotated with  $c$ ) and the rules defining the global policy conflict resolution mechanism (i.e. rules  $\text{authorize}$  and  $\text{combine}$ ). Moreover, the usual rewrite rules for dealing with arithmetics and data structures are also included. The policy is proven to be total and consistent. Access decisions are computed with the CiME command:

```
let t = term R_trs_algebra
    "authorize(alertoAlice, consult, loanList)";
```

<sup>2</sup>For the sake of clarity, the set of categories defined in  $l$  and those defined in  $c$  are disjoint. However, we may have one or more categories defined in both sites with different privileges. This does not prevent the global policy from being consistent.

```
normalize R_trs_t;
...
- : R_trs_signature term = grant
```

where  $\text{authorize}(\text{alertoAlice}, \text{consult}, \text{loanList})$  is the access request we want to evaluate (seen as a term) and  $\text{R\_trs}$  is the rewrite system specifying the distributed access control policy. This request will lead to a positive answer if the principal Alice belongs to the category manager or banker in the central site (the local policy not being able to decide about the access permission requested).

Incompleteness of the policy can be detected, for example if the  $\text{authorize}$  function is not completely defined over the set of inputs, e.g. a principal has no category assignment in any site. In this case, the computation will result in a final term which is not an answer in  $\mathcal{Auth}$  and an error identifying such a principal will be given (concretely, the function  $\text{pca}_s(p)$  cannot be evaluated in any of the sites  $s \in \mathcal{S}$ ). Notice that the existence of one or more sites where  $\text{pca}(p)$  is not defined is not surprising in a distributed scenario where information is only partially specified in each site, i.e. local policies do not hold the complete information, which is dispersed over the whole system.

## 6. Related work

A number of works in the literature address the problem of access control policy representation and analysis. This section does not pretend to be exhaustive.

In [44, 45] a uniform framework based on graph transformations is described for the specification of access control policies and their analysis. Graph transformation rules combine a visual representation with formal categorical semantics, and thus provide a solid basis for policy analysis. Our work addresses similar issues to [44, 45] but provides a different formulation of access control policies that facilitates the performing of automated analysis by existing rewrite tools.

Several works exist dealing with rewrite-based policies [9, 34–36], including works by the first author [16–19], where the category-based metamodel is introduced. The security properties considered in these works are defined for general abstract policies and formally proved “by hand”, without the assistance of any automated tool. Moreover, their focus is essentially on the issue of defining an expressive and flexible framework for policy (or policy combination) modeling and specification, without considering its practical usability by real policy administrators. More recently, rewrite-based policy verification is addressed e.g. in [24, 42, 52], where examples about information flow policies or firewalls are treated. However, no rewrite tools such as CiME3 have been used for automated validation and certification of classical security properties on these policies.

There have been several works which give formal semantics to policies by using a logic programming approach. For instance, Abadi et al [1] ABLP logic provides a formal framework for reasoning about a wide range of features of access control. The focus in ABLP logic is on language constructs for formulating access control policies and axioms and inference rules for defining a system for proof, e.g., for proving authorised forms of access. In contrast, our approach is based on the rewrite-based specification of the metamodel, which is proved to be well-adapted for access requests evaluation, and emphasises the use of term rewriting techniques to derive properties of the policies.

The work on access control proposed in [10, 11, 39] is also related to ours. In these approaches policies are modelled using (constrained) Horn clauses and evaluation is based on resolution [53]. The term rewriting approach is similarly declarative, and has similar attractions to the logic programming approaches. However, in

contrast to these approaches, our proposal does not require the syntactic restriction to *locally stratified* access policies [7] to be adopted.

The distributed model that we have used is more expressive than any of the *Datalog*-based languages that have been proposed for distributed access control (e.g. OASIS [5], SD3 [40], Cassandra [32], Binder [13]), which are based on a monotonic semantics and thus not especially well suited for representing dynamic distributed access request policies.

The work reported in [29] is based on the use of logic programs for specifying and analyzing authorization and obligation policies. However obligation management is quite different from access control, where a principal usually has no individual control over the permissions it has. Therefore, we do not consider obligations in this paper. Moreover, we focus on a user-friendly approach for policy designers which may use our prototype to easily specify rules and queries and to read in natural language the analysis result.

In [25] the authors define an expressive logical framework for policy representation and reasoning, but their emphasis is mainly on policy composition. In [14] an access control policy language is presented with an associated analysis framework based on trans-action logics, however explicit prohibition is not representable with this approach. In [37] the authors introduce a logical formalism for policy representation and analysis. As they work with pure first-order logics, default decision policies are not expressible, thus complete definitions have to be specified. Moreover, they do not provide explicit support for groups and roles. Finally, in [2], efficient automated analysis of access control policies is provided, however only centralised RBAC policies are concerned by this technique; in [41] a unified framework covering several models of access control is introduced, but its focus is on policy specification while policy analysis is not discussed.

## 7. Conclusions

We have described an application whose aim is to help policy designers to specify expressive access control policies and to (automatically) verify desirable security properties on those policies. To illustrate the feasibility of our approach, we have successfully tested the application on a case study for a simplified banking scenario (both in a centralized and in a distributed setting, following the general category-based access control metamodel  $\mathcal{M}$ ). Using this approach, we gain clarity in the representation of policies and also the ability to smoothly combine different access control models in order to study the behaviour of collaborating policies on potentially different (sub-)systems.

In future work we want to tackle problems related to the enforcement of rule-based policies. The rewrite-based approach we have chosen provides executable specifications for security policies, which can be independently designed, verified, and then anchored on programs using a modular discipline. We aim at studying a general methodology, in the style of [31], for weaving rule-based access control policies descriptions into existing code, such that the resulting (distributed) system transforms the untrusted code in a safe program.

Another interesting issue is to explore other verification techniques based on rewriting, such as the *narrowing* process (also supported by CiME3). This may be useful for checking properties such as the sufficient completeness of a rewrite system. An experimental tool for sufficient completeness checking is available for confluent and terminating Maude specifications [51]. Also, narrowing provides an abstract mechanism for solving queries from the security administrator [49], such as "what is the result of assigning to a principal  $p$  an additional category  $c$ ". We believe that, under some restrictions on the form of the rewrite rules, this kind of technique can also help in analysing safety issues (usually undecidable), that

is the problem of deciding whether a principal can ever obtain a specific permission or not. Some preliminary work in this sense has been reported in [42] for strategic rewriting. It would be interesting to adapt these techniques in our framework and derive proofs of security properties for category-based policies.

## References

- [1] M. Abadi, M. Burrows, B. W. Lampson, and G. D. Plotkin. A calculus for access control in distributed systems. *ACM Trans. Program. Lang. Syst.*, 15(4):706–734, 1993.
- [2] A. Armando and S. Ranise. Automated and efficient analysis of role-based access control with attributes. In *Proc. of the 26th Annual IFIP WG 11.3 conference on Data and Applications Security and Privacy, DBSec'12*, pages 25–40. Springer-Verlag, 2012.
- [3] The Coq Proof Assistant. Reference manual – version v8.0, <http://coq.inria.fr>.
- [4] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, Great Britain, 1998.
- [5] J. Bacon, K. Moody, and W. Yao. A model of OASIS RBAC and its support for active security. *TISSEC*, 5(4):492–540, 2002.
- [6] E. Balland, P. Brauner, R. Kopetz, P.-E. Moreau, and A. Reilles. Tom: Piggybacking rewriting on java. In *Proc. of RTA 2007*, volume 4533 of *LNCS*, pages 36–47. Springer, 2007.
- [7] C. Baral and M. Gelfond. Logic programming and knowledge representation. *JLP*, 19/20:73–148, 1994.
- [8] S. Barker. The next 700 access control models or a unifying meta-model? In *Proc. of SACMAT 2009*, pages 187–196. ACM Press, 2009.
- [9] S. Barker and M. Fernández. Term rewriting for access control. In *Data and Applications Security. Proceedings of DBSec'2006*, Lecture Notes in Computer Science. Springer-Verlag, 2006.
- [10] S. Barker and P. Stuckey. Flexible access control policy specification with constraint logic programming. *ACM Trans. on Information and System Security*, 6(4):501–546, 2003.
- [11] Steve Barker. Action-status access control. In *SACMAT*, pages 195–204, 2007.
- [12] Steve Barker. Logical approaches to authorization policies. In *Logic Programs, Norms and Action*, volume 7360 of *Lecture Notes in Computer Science*, pages 349–373. Springer, 2012.
- [13] M. Becker and P. Sewell. Cassandra: Distributed access control policies with tunable expressiveness. In *POLICY 2004*, pages 159–168, 2004.
- [14] M. Y. Becker and S. Nanz. A logic for state-modifying authorization policies. *ACM Trans. Inf. Syst. Secur.*, 13(3):20:1–20:28, July 2010.
- [15] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. *ACM Trans. Inf. Syst. Secur.*, 6(1):71–127, 2003.
- [16] C. Bertolissi and M. Fernández. Distributed event-based access control. *International Journal of Information and Computer Security, Special Issue: selected papers from Crisis 2008*, 3(3–4), 2009.
- [17] C. Bertolissi and M. Fernández. Category-based authorisation models: operational semantics and expressive power. In *Proc. of ESSOS 2010*, number 5965 in *LNCS*. Springer, 2010.
- [18] C. Bertolissi and M. Fernández. Rewrite specifications of access control policies in distributed environments. In *Proc. of STM 2010*, number 6710 in *Lecture Notes in Computer Science*. Springer, 2011.
- [19] C. Bertolissi, M. Fernández, and S. Barker. Dynamic event-based access control as term rewriting. In *Proceedings of DBSEC 2007*, number 4602 in *LNCS*. Springer-Verlag, 2007.
- [20] C. Bertolissi and W. Uttha. Automated analysis of rule-based access control policies. available at <http://www.lif.univ-mrs.fr/~clara/research.html>.
- [21] P. Bonatti, S. de Capitani di Vimercati, and P. Samarati. A modular approach to composing access control policies. In *CCS'00: Proceedings*, pages 164–173, New York, NY, USA, 2000. ACM Press.



- [22] P. A. Bonatti and P. Samarati. Logics for authorization and security. In *Logics for Emerging Applications of Databases*, pages 277–323. Springer, 2003.
- [23] A. Bouhoula and F. Jacquemard. Sufficient completeness verification for conditional and constrained trs. *J. Applied Logic*, 10(1):127–143, 2012.
- [24] T. Bourdier and H. Cirstea. Symbolic analysis of network security policies using rewrite systems. In *Proc. of PPDP'11*, pages 77–88. ACM, 2011.
- [25] G. Bruns and M. Huth. Access control via belnap logic: Intuitive, expressive, and analyzable policy composition. *ACM Trans. Inf. Syst. Secur.*, 14(1):9:1–9:27, June 2011.
- [26] S. Chen, D. Wijesekera, and S. Jajodia. Flexflow: A flexible flow control policy specification framework. In *Proc. of Conference on Data and Application Security, DBSec 2003*, pages 358–371. Kluwer, 2003.
- [27] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The Maude 2.0 system. In *Proc of RTA 2003*, number 2706 in LNCS, pages 76–87. Springer-Verlag, 2003.
- [28] É. Contejean, A. Paskevich, X. Urbain, P. Courtieu, O. Pons, and J. Forest. A3pat, an approach for certified automated termination proofs. In *Proc. of PEPM '10*, pages 63–72, New York, NY, USA, 2010. ACM.
- [29] R. Craven, J. Lobo, J. Ma, A. Russo, E. Lupu, and A. Bandara. Expressive policy analysis with enhanced system dynamicity. In *Proc. of the 4th International Symposium on Information, Computer, and Communications Security, ASIACCS '09*, pages 239–250, New York, NY, USA, 2009. ACM.
- [30] Á. Darvas, R. Hähnle, and D. Sands. A theorem proving approach to analysis of secure information flow. In *Workshop on Issues in the Theory of Security*, 2003.
- [31] A. Santana de Oliveira, E. Ke Wang, C. Kirchner, and H. Kirchner. Weaving rewrite-based access control policies. In *Proc. of FMSE 2007*, pages 71–80. ACM, 2007.
- [32] J. DeTreville. Binder, a logic-based security language. In *Proc. IEEE Symposium on Security and Privacy*, pages 105–113, 2002.
- [33] S. De Capitani di Vimercati and P. Samarati. Authorization specification and enforcement in federated database systems. *J. Comput. Secur.*, 5:155–188, March 1997.
- [34] D. J. Dougherty, C. Kirchner, H. Kirchner, and A. Santana de Oliveira. Modular access control via strategic rewriting. In *Proc. of ESORICS 2007*, pages 578–593, 2007.
- [35] R. Echahed and F. Prost. Security policy in a declarative style. In *Proc. of PPDP'05*. ACM Press, 2005.
- [36] L. Habib, M. Jaume, and Charles Morisset. Formal definition and comparison of access control models. *Journal of Information Assurance and Security*, 4:372–378, 2009.
- [37] Joseph Halpern and Vicky Weissman. Using first-order logic to reason about policies. *ACM Trans. Inf. Syst. Secur.*, 11(4):21:1–21:41, July 2008.
- [38] J. Hendrix, M. Clavel, and J. Meseguer. A sufficient completeness reasoning tool for partial specifications. In *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, number 3467 in Lecture Notes in Computer Science, pages 165–174. Springer, 2005.
- [39] S. Jajodia, P. Samarati, M. Sapino, and V.S. Subrahmanian. Flexible support for multiple access control policies. *ACM TODS*, 26(2):214–260, 2001.
- [40] T. Jim. SD3: A trust management system with certified evaluation. In *IEEE Symp. Security and Privacy*, pages 106–115, 2001.
- [41] X. Jin, R. Krishnan, and R. S. Sandhu. A unified attribute-based access control model covering dac, mac and rbac. In *Proc. of the 26th Annual IFIP WG 11.3 conference on Data and Applications Security and Privacy, DBSec'12*, pages 41–55. Springer-Verlag, 2012.
- [42] C. Kirchner, H. Kirchner, and A. S. de Oliveira. Analysis of rewrite-based access control policies. *ENTCS*, 234:55–75, 2009.
- [43] J.-W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems, introduction and survey. *TCS*, 121:279–308, 1993.
- [44] M. Koch, L. V. Mancini, and F. Parisi-Presicce. Foundations for a graph-based approach to the specification of access control policies. In *Proc. of the 4th International Conference on Foundations of Software Science and Computation Structures, FoSSaCS '01*, pages 287–302. Springer-Verlag, 2001.
- [45] M. Koch, L. V. Mancini, and F. Parisi-Presicce. Decidability of safety in graph-based models for access control. In *In Proc. of 7th ESORICS, volume 2502 of LNCS*, pages 229–243. Springer, 2002.
- [46] D. Richard Kuhn, Edward J. Coyne, and Timothy R. Weil. Adding attributes to role-based access control. *Computer*, 43:79–81, 2010.
- [47] M.H.A. Newman. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2):223–243, 1942.
- [48] Department of Defense. Trusted computer system evaluation criteria, 1983. DoD 5200.28-STD.
- [49] S. Oh, R. S. Sandhu, and X. Zhang. An effective role administration model using organization structure. *ACM Trans. Inf. Syst. Secur.*, 9(2):113–137, 2006.
- [50] The A3PAT project. <http://a3pat.ensiie.fr>.
- [51] The Maude System. Maude sufficient completeness checker <http://maude.cs.uiuc.edu/tools/scc/>.
- [52] M. Jaume T. Bourdier, H. Cirstea and H. Kirchner. Formal specification and validation of security policies. In *FPS-MITACS Workshop, Selected Papers*, volume 6888 of LNCS, pages 148–163. Springer, 2011.
- [53] *The XSB System Version 2.7.1, Programmer's Manual*, 2005.