A Metamodel of Access Control for Distributed Environments: Applications and Properties

Clara Bertolissi[a], Maribel Fernández[b]

[a] *LIF, UMR CNRS 7279 & AMU, Marseille, France*
[b] *Dept. of Informatics, King's College London, UK*

Abstract

We describe a metamodel for access control, designed to take into account the specific requirements of distributed environments. We see a distributed system consisting of several sites, each with its own resources to protect, as a federation, and propose a framework for the specification (and enforcement) of global access control policies that take into account the local policies specified by each member of the federation. The framework provides mechanisms to specify heterogeneous local access control policies, to define policy composition operators, and to use them to define conflict-free access authorisation decisions. We use a declarative formalism in order to give an operational semantics to the distributed metamodel. We then show how properties of policies can be directly obtained from standard results for the operational semantics of access request evaluation.

Keywords: Security Policies, Distributed Access Control, Operational Semantics, Rewriting.

1. Introduction

Access control is a fundamental aspect of computer security; it aims at protecting resources from non-authorised users. The generalised use of access control in distributed computing environments has increased the need for high-level declarative languages that enable security administrators to specify a wide range of complex policies. More specifically, distributed environments require dynamic policies that can take into account changes in the system (for example, time and location-dependent policies, or more generally, policies that change according to the events that take place in the system) or collaborative policies (e.g. policies that take into account the access control requirements of several sites in the system).

Policy Specifications. Using a formal specification language for defining access control models and policies is particularly important in distributed contexts, to understand the impact of changes in the policies and prove properties of policies. Formal theories to

Email addresses: Clara.Bertolissi@lif.univ-mrs.fr (Clara Bertolissi), Maribel.Fernandez@kcl.ac.uk (Maribel Fernández)

define and validate security policies (see, for instance, [24]) have used first-order theorem provers, purpose-built logics, or flow-analysis, but these approaches have limitations (as discussed for instance in [43]). More recently, rewriting techniques have been fruitfully exploited in the context of security protocols (see for example [12]), policies controlling information leakage (see for example [36]), and access control policies (see for example [59, 21]). Along these lines, rewriting systems appear to be well adapted for providing a semantics for distributed access control mechanisms. On one hand, security policies and protocols can conveniently be specified as sets of rules, which can be formalised as a rewriting system [2, 36]. On the other hand, rewrite systems provide a multiparadigm computation model: they have been used as a semantics for logic programming languages (via unification and narrowing [38]), for functional languages (via matching and reduction [40, 26, 27]) and they can express imperative and concurrent features (in-place update [33], or a process calculus [62]).

Rewriting systems present many advantages as a specification tool: they have a well-studied theory, with a wealth of results that can be applied to the analysis of policies, and several rewrite-based programming languages are available for fast prototyping (see, e.g. Maude [28]). Once the 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).

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 a clean and unambiguous semantics; moreover, the declarative nature of this kind of policy specifications enhances modularity, which is a crucial aspect when considering distributed security policies developed independently by different departments or organisations. The possibility to write policies as modular sets of authorisation rules offers administrators more flexibility and simplicity for specifying and combining access control policies [16].

Access control models. Over the last few years, a variety of access control models and languages for access control policy specification have been developed, often motivated by particular applications. We can mention the mandatory access control (MAC) model [14], the ANSI (hierarchical) role-based access control (H-RBAC) model [57], further extended with time and location constraints [25], the event-based access control (DEBAC) model [21], etc. A unifying metamodel for access control, which can be specialised for domain-specific applications, has been proposed in [8]. This unifying approach has advantages: for example, by identifying a core set of principles of access control, one can abstract away many of the complexities that are found in specific access control models; this, in turn, helps to simplify the task of policy writing. A rewrite-based operational semantics for this metamodel is given in [19], where the expressive power of the metamodel is also demonstrated by showing that all of the above mentioned access control models can be derived as specific instances of the metamodel.

In [20], the same approach was used to define a metamodel of access control for *distributed environments* where each component of the system preserves its autonomy. A key aspect of this approach, following [8], is to focus attention on primitive notions common to different access control models, such as grouping of entities, methods for describing their properties and means for specifying privileges and authorisations. Classic

types of groupings used in access control, like a role, a security clearance, a discrete measure of trust, etc., are seen in the metamodel as particular instances of a more general notion of *category*. For example, as far as access control models based on roles are concerned, we simply see the user-role(s) relationship as the assignment of users to their corresponding category(ies). Regarding models based on trust, we consider the association of a trust measure with a principal as the assignment of the principal to a category of users that have the same degree of trust according to some authority. This idea of categorisation applies also to distributed, federative settings. In a system with dispersed resources, classifications of entities may depend on the site to which the entity belongs. Moreover, permissions associated to categories of entities may also depend on the site where the category is defined. Therefore, we may want to use a distributed access control evaluation method, in addition to the central one proposed in [19], or we may want to combine the two.

Contributions. In this paper, we give a formal specification of the distributed metamodel in a rewrite-based language, and focus on the modular properties of the system.

The notion of distributed environment that we consider here is related to the notion of *federation* developed in the context of database systems (see for example [46, 32], where a federated system integrates several databases while preserving their autonomy). We see a distributed system consisting of several sites, each with its own resources to protect, as a federation, and focus on access control. We propose a formal framework for modelling (and enforcing) global access control policies that take into account the local policies specified and maintained by each member of the federation. In particular we ensure the coherence of a global access control decision w.r.t. local access control requirements by specifying in a tunable way how to integrate access authorisations resulting from the local policies. In this framework, distributed access control policies can be easily specified and manipulated, by means of local policy specification mechanisms and definitions of policy composition operators.

Following [20], we first axiomatise a *distributed access control metamodel*, then give a rewrite-based operational semantics for this metamodel using the techniques introduced in [18], which allow us to deal in a uniform way with distributed systems where different access control policies are maintained locally. We demonstrate the expressive power of the distributed metamodel by showing how a distributed, dynamic, event-based access control model (the Distributed DEBAC model [18]) can be defined as an instance of the metamodel. We also show examples of distributed access control policy specifications, where a policy for a distributed federation is defined as a combination of individual policies.

This declarative approach permits properties of access control policies to be proved in a modular way. In particular, we are interested in consistency and totality properties of policies. These properties guarantee that access requests will be treated as expected. A consistent access control policy specifies at most one answer for each access request (i.e., an access request cannot be both granted and denied). Totality guarantees that every access request will be given an answer. We show how consistency and totality properties of access control policies can be derived from standard properties of the rewrite framework we use.

Summarising, the main contributions of this paper are:

- a declarative, rewrite-based *specification* of a distributed, category-based access control metamodel, where distributed systems are seen as federations in which each component preserves its autonomy;
- a technique to define *combinations* of policies, by defining general policy-combining operators;
- a formal operational semantics for *access request evaluation* in centralised as well as in distributed contexts where information is shared, including mechanisms for the resolution of conflicts between local and global policies;
- a technique to prove *totality and consistency* of access control policies, based on termination and confluence properties of the underlying term rewriting system.

This paper is based on preliminary work on rewrite-based specifications for the category-based metamodel presented in [19, 20], and extends this previous work by providing new applications and new techniques for the analysis of policies obtained as instances of the metamodel.

Overview of the paper. In Section 2, we recall basic notions in rewriting and describe the main features of the access control metamodel. In Section 3, we define the extension of the metamodel for distributed environments. In Section 4 we specify the operational semantics of the distributed metamodel as a rewriting system and discuss access request evaluation methods. In Section 5, we show how a variety of operators can be included in the metamodel for combining access request answers issued locally by members of the federation, in order to obtain a final authorisation or denial of access. We describe techniques for proving properties of access control policies in Section 6. Finally, we extend the specification language with higher-order features in Section 7, to define in a concise way more involved policy combination mechanisms, and we give in Section 8 techniques to prove properties of policies defined using higher-order rules. In Section 9, we discuss related work and in Section 10 we conclude and suggest further work.

2. Preliminaries

We recall the main notions of rewriting that we will need in the rest of the paper, as well as the main features of the category-based access control metamodel. We refer the reader to [7, 3, 8] for additional information on λ-calculus, rewrite systems and the category-based metamodel, respectively.

2.1. Rewriting

Below we define a combination of λ-calculus and term rewriting; these systems are extensions of Curryfied Term Rewriting Systems (CTRS) [4], also called *algebraic λ-calculi*.

Definition 2.1 (Signatures and terms). *A signature $\mathcal{F}$ is a finite set of function symbols together with their (fixed) arities. The set $\mathcal{F}_\lambda$ is obtained from $\mathcal{F}$ by adding a special binary operator* Ap*, called* application*. $\mathcal{X}$ denotes an infinite, countable set of variables $x_1, x_2, \ldots$. The set $T_\lambda(\mathcal{F}, \mathcal{X})$ of* terms *is defined inductively as follows:*

1. $\mathcal{X} \subseteq T_\lambda(\mathcal{F}, \mathcal{X})$.
2. *If* $\mathsf{f} \in \mathcal{F}_\lambda$ *is an* n*-ary symbol* ($n \geq 0$)*, and* $t_1, \ldots, t_n \in T_\lambda(\mathcal{F}, \mathcal{X})$*, then* $\mathsf{f}(t_1, \ldots, t_n) \in T_\lambda(\mathcal{F}, \mathcal{X})$*. In particular,* $\mathsf{Ap}(t_1, t_2) \in T_\lambda(\mathcal{F}, \mathcal{X})$ *if* $t_1, t_2 \in T_\lambda(\mathcal{F}, \mathcal{X})$*; it is called an* application *term.*
3. *If* $t \in T_\lambda(\mathcal{F}, \mathcal{X})$*, and* $x \in \mathcal{X}$*, then* $\lambda x.t \in T_\lambda(\mathcal{F}, \mathcal{X})$ *and it is called a* λ-abstraction *(or simply abstraction).*

In $\lambda x.t$*,* λ *is a binder: all occurrences of* x *in* t *are* bound*. Terms are defined modulo* α*-equivalence; i.e., modulo renamings of bound variables.*

If an occurrence of a variable in a term t *is not bound, then it is* free*. The set of* free variables *of* t *(i.e., variables that have free occurrences in* t*) is denoted by* $\mathcal{FV}(t)$*. We say that a term* t *is* linear *if each variable has at most one free occurrence in* t*.*

We distinguish the following classes of terms:

- A λ*-term* is a term not containing function symbols in $\mathcal{F}$.
- An *algebraic term* (or first-order term) is a term containing neither λ nor Ap.
- An *applicative term* is a term that does not contain λ-abstractions.

Although terms are defined modulo α-equivalence, a representative of a term can be seen as a finite labelled tree, with variables and 0-ary function symbols at the leaves and internal nodes labelled by symbols in $\mathcal{F}_\lambda$ or by λx, for some x. We write $\hat{t}$ to denote a tree representing the term t. *Positions* are strings of positive integers. We use ϵ to denote the empty string, corresponding to the root position; string concatenation is denoted simply by juxtaposition: if p is a string denoting a position in the tree representation of a term, then pq denotes the position q in the subtree at position p. The subterm of $\hat{t}$ at position p is denoted by $\hat{t}|_p$ and the result of replacing $\hat{t}|_p$ with $\hat{u}$ at position p is denoted by $\hat{t}[\hat{u}]_p$. Note that for different representatives of t we may have different terms at position p, in particular when the node at position p is a bound variable. We write simply $t|_p$, $t[u]_p$ when the choice of the representative is not important.

Substitutions are written $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where each t_i is a term, and it is assumed to be different from the variable x_i. We use Greek letters for substitutions and postfix notation for their application. To apply a substitution to a term, we use the standard capture-avoiding mechanism.

Definition 2.2 (TRS_λ)**.** *A* rewrite rule *is a pair* $l \to r$ *of terms in* $T_\lambda(\mathcal{F}, \mathcal{X})$*, such that* $l \notin \mathcal{X}$*,* l *is an algebraic term, and* $\mathcal{FV}(r) \subseteq \mathcal{FV}(l)$*.*

The β-rule *is defined by the scheme:* $\mathsf{Ap}(\lambda x.t, u) \to_\beta t\{x \mapsto u\}$

An extended term rewrite system (or algebraic λ*-calculus), abbreviated* TRS_λ*, is a set of rewrite rules together with the* β*-rule.*

Definition 2.3 (Reduction). *A term* t *rewrites to* u *at position* p *with a rewrite rule* $l \to r$ *and substitution* σ*, written* $t \to_p^{l \to r} u$ *or simply* $t \to u$*, if* $t|_p = l\sigma$ *and* $u = t[r\sigma]_p$*.*

Likewise, for any t, s, u*,* $t[\mathsf{Ap}(\lambda x.s, u)]_p \to t[s\{x \mapsto u\}]_p$ *using the* β*-rule.*

The one-step reduction relation associated to a TRS_λ*, denoted by* $\to$*, is the union of the relations generated by its rewrite rules and the* β*-rule. The (multi-step) reduction relation* $\to^*$ *is the reflexive-transitive closure of* $\to$*.*

If there is u *such that* $t \to u$ *we say that* t *is* reducible*. Irreducible terms are said to be in* normal form*.*

Since variables in a TRS_λ can be substituted by λ-expressions, we obtain the usual functional programming paradigm, extended with rewrite rules. Below, to simplify the notation, we use the standard λ-calculus conventions (the symbol Ap will be omitted).

Definition 2.4 (Constructor). *We say that a rule* $\mathsf{f}(t_1, \ldots, t_n) \to r$ defines f. *If a function symbol is not defined, it is a* constructor.

Constructor terms *are terms built out of variables and constructors.*

A constructor system *is a rewrite system in which the left-hand sides of all the rewrite rules have the form* $\mathsf{f}(t_1, \ldots, t_n)$ *where* $t_1, \ldots, t_n$ *are constructor terms.*

For instance, in the following example the rules form a constructor system: $\mathsf{nil}, \mathsf{cons}, \mathsf{s}, \mathsf{z}$, true, false are constructors, append, and, $\mathsf{ifthenelse}$ are defined functions.

Example 2.5. *(Algebraic terms) Consider a signature for lists of natural numbers, with function symbols* z *of arity* 0 *and* s *of arity* 1 *to build numbers and* nil *of arity* 0, cons *of arity 2 to build lists. The list containing the numbers* 0 *and* 1 *is written* $\mathsf{cons}(\mathsf{z}, \mathsf{cons}(\mathsf{s}(\mathsf{z}), \mathsf{nil}))$, *or simply* $[\mathsf{z}, \mathsf{s}(\mathsf{z})]$ *for short. We can define list concatenation with the following rewrite rules:*

$$\begin{array}{rcl} \mathsf{append}(\mathsf{nil}, l) & \to & l \\ \mathsf{append}(\mathsf{cons}(x, l), l') & \to & \mathsf{cons}(x, \mathsf{append}(l, l')) \end{array}$$

Then we have a reduction sequence:

$$\mathsf{append}(\mathsf{cons}(\mathsf{z}, \mathsf{nil}), \mathsf{cons}(\mathsf{s}(\mathsf{z}), \mathsf{nil})) \to^* \mathsf{cons}(\mathsf{z}, \mathsf{cons}(\mathsf{s}(\mathsf{z}), \mathsf{nil}))$$

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*

$$\mathsf{and}(\mathsf{true}, x) \to x \qquad \mathsf{and}(\mathsf{false}, x) \to \mathsf{false}.$$

The notation t_1 *and* $\ldots$ *and* t_n *is syntactic sugar for* $\mathsf{and}(\ldots \mathsf{and}(\mathsf{and}(t_1, t_2), t_3) \ldots)$, *and if* b *then* s *else* t *is syntactic sugar for* $\mathsf{ifthenelse}(b, s, t)$, *such that:*

$$\mathsf{ifthenelse}(\mathsf{true}, x, y) \to x \qquad \mathsf{ifthenelse}(\mathsf{false}, x, y) \to y$$

For example, we can define the membership operator "$\in$*" as follows:*

$$\begin{array}{rcl} \in(x, \mathsf{nil}) & \to & \mathsf{false} \\ \in(x, \mathsf{cons}(h, l)) & \to & \textit{if } \mathsf{equal}(x, h) \textit{ then } \mathsf{true} \textit{ else } \in(x, l) \end{array}$$

where we assume equal *is a syntactic equality test defined by standard rewrite rules on algebraic constructor terms (e.g.,* $\mathsf{equal}(\mathsf{z}, \mathsf{z}) \to \mathsf{true}$, $\mathsf{equal}(\mathsf{z}, \mathsf{s}(x)) \to \mathsf{false}$, *etc.). We will often write* $\in$ *as an infix operator.*

Example 2.6. *(Extended TRS) We can define the function* map *(which applies a function "f" given as argument to each element of a list) as follows:*

$$\begin{array}{rcl} \mathsf{map}(f, \mathsf{nil}) & \to & \mathsf{nil} \\ \mathsf{map}(f, \mathsf{cons}(y, l)) & \to & \mathsf{cons}(f\ y, \mathsf{map}(f, l)) \end{array}$$

We say that two terms are syntactically unifiable if there is some substitution that makes them equal. Such a substitution is called a *unifier*. A *most general unifier* (mgu) is a unifier that will yield instances in the most general form; see, for instance, [52] for a description of an efficient unification algorithm.

Let $l \to r$ and $s \to t$ be two rules (renamed if necessary so that there is no common variable), p the position of a non-variable subterm of s, and μ a most general unifier of $s|_p$ and l. Then $(t\mu, s\mu[r\mu]_p)$ is a *critical pair* [3] formed from those rules. Intuitively, the existence of a critical pair implies that there is a superposition between the left-hand sides l and s of the two considered rewrite rules, i.e., the rules overlap. We use syntactic unification to compute critical pairs, since the left-hand sides of rules do not contain binders. Note that $s \to t$ may be a renamed version of $l \to r$. In this case a superposition at the root position is not considered a critical pair.

The notion of critical pair is useful for defining some properties of rewriting systems that will be developed in Sections 6 and 8.

Definition 2.7 (Properties of rewriting systems). *A TRS_λ R is*

- *non-overlapping if there are no critical pairs;*
- *left-linear if all left-hand sides of rules in R are linear;*
- *orthogonal if R is left-linear and non-overlapping;*
- *confluent if for all terms t, u, v: $t \to^* u$ and $t \to^* v$ implies $u \to^* s$ and $v \to^* s$, for some s;*
- *terminating (or strongly normalising) if all reduction sequences are finite;*
- *non-duplicating if for all rule $l \to r \in R$ and variable $x \in \mathcal{V}(l)$, the number of free occurrences of x in r is less than or equal to the number of occurrences of x in l.*

Orthogonal systems are confluent, as shown by Klop [47]. For example, the rewrite system in Example 2.5 is left-linear and non-overlapping (therefore orthogonal), confluent and terminating.

To specify distributed access control policies, we will follow the approach of [21], where distributed term rewriting systems (DTRSs) are introduced. In DTRSs, rules are partitioned into modules, each associated with an identifier, and defined function symbols are annotated with such identifiers. We associate modules to sites. For example, we may write f_ν to refer to the definition of the function symbol f in the site ν, where ν is a site identifier. The rules defining f may be distributed across several sites. We assume that each module has a unique identifier; all the functions defined in a module are annotated with the same identifier. If a symbol is used in a rule at site ν without a site annotation, we assume the function is defined locally in the site ν.

To illustrate the use of annotations on function symbols, we consider a bank scenario, where clients make deposits and withdrawals that change the average balance of their accounts. We define classes of clients using rules such as:

$$\begin{aligned} \mathsf{class}(u) \to \quad & \mathit{if}\ \mathsf{averagebalance}_\nu(\mathsf{account}(u)) > 10000 \\ & \mathit{then}\ \mathsf{gold\text{-}client}\ \mathit{else}\ \mathsf{normal\text{-}client} \end{aligned}$$

assuming there are rules to compute $\mathsf{account}(u)$ (i.e., u's bank account number) and the average balance of a user's account. The average balance of u's account is computed at site ν, but u's account number is computed locally.

For more details on Distributed Term Rewriting Systems, we refer the reader to [21].

2.2. Category-Based Metamodel

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.

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, ...
- A countable set $\mathcal{P}$ of *principals*, denoted p_0, p_1, ...
- A countable set $\mathcal{A}$ of named *actions*, denoted a_0, a_1, ...
- A countable set $\mathcal{R}$ of *resource identifiers*, denoted r_0, r_1, ...
- A finite set $\mathcal{A}uth$ of possible *answers* to access requests.
- A countable set $\mathcal{S}$ of *situational identifiers*.

We do not deal with authentication in this paper; we assume that principals that request access to resources are pre-authenticated. This does not mean that we assume any specific behaviour of the authentication program (any user could be allowed to log in). Indeed, the category-based model does not rely on principals being registered (for example, a default category could be defined for "unknown" principals).

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 $\mathcal{A}uth$ might include {grant, deny, grant-if-obligation-is-satisfied, undetermined}.

In addition to the different types of 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 $c \in \mathcal{C}$.
- *Authorisations:* $\mathcal{PAR} \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{R}$, 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 authorisations that hold according to an access control policy that specifies $\mathcal{PCA}$ and $\mathcal{ARCA}$.

Definition 2.8 (Axioms). *The relation $\mathcal{PAR}$ satisfies the following core axiom, where we assume that there exists a reflexive-transitive relationship $\subseteq$ between categories; this can simply be equality, set inclusion (the set of principals assigned to $c \in \mathcal{C}$ is a subset of the set of principals assigned to $c' \in \mathcal{C}$), or an application specific relation defining a hierarchy of categories may be used. If $c \subseteq c'$ we say that c is above c', and c' is below c; for example* manager $\subseteq$ employee.

$$(a1) \quad \forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R},\\ (\exists c \in \mathcal{C}, \exists c' \in \mathcal{C}, (p, c) \in \mathcal{PCA} \wedge c \subseteq c' \wedge (a, r, c') \in \mathcal{ARCA}) \Leftrightarrow (p, a, r) \in \mathcal{PAR}$$

The category-based metamodel of access control is based on the core axiom $(a1)$ for $\mathcal{PAR}$ given in Def. 2.8. Operationally, this axiom can be realised through a set of function definitions [19], as described below.

Definition 2.9. *The information contained in the relations $\mathcal{PCA}$ and $\mathcal{ARCA}$ is modelled by the functions* pca *and* arca*, respectively, where* pca *returns the list of categories assigned to a principal, e.g.* $\mathsf{pca}(p) \rightarrow [\mathsf{c}]$*, and* arca *returns a list of permissions assigned to a category, e.g.* $\mathsf{arca}(c) \rightarrow [(\mathsf{a}_1, \mathsf{r}_1), \ldots, (\mathsf{a}_n, \mathsf{r}_n)]$.

The rewrite-based specification of the axiom $(a1)$ in Def. 2.8 is given by the rewrite rule:

$$(a2) \quad \mathsf{par}(p, a, r) \quad \rightarrow \quad if\ (a, r) \in \mathsf{arca}^*(\mathsf{below}(\mathsf{pca}(p)))\ then\ \mathsf{grant}\ else\ \mathsf{deny}$$

As the function name suggests, below *computes the set of categories that are below (w.r.t. the hierarchy defined by the $\subseteq$ relation) any of the categories given in the list* $\mathsf{pca}(P)$*. For example, for a given category* c*, this could be achieved by using a rewrite rule* $\mathsf{below}([\mathsf{c}]) \rightarrow [\mathsf{c}, \mathsf{c}_1, \ldots, \mathsf{c}_n]$*. The function $\in$ is a membership operator on lists (see Section 2),* grant *and* deny *are answers, and* arca^* *generalises the function* arca *to take into account lists of categories:*

$$\mathsf{arca}^*(\mathsf{nil}) \rightarrow \mathsf{nil} \qquad \mathsf{arca}^*(\mathsf{cons}(c, l)) \rightarrow \mathsf{append}(\mathsf{arca}(c), \mathsf{arca}^*(l))$$

For optimisation purposes, one can compose the standard list concatenation operator append with a function removing the duplicate elements in the list.

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 $\mathsf{par}(\mathsf{p}, \mathsf{a}, \mathsf{r})$ to normal form.

Note that $(a1)$, and its algebraic version $(a2)$, state that a request by a principal p to perform the action a on a resource r is authorised only if p belongs to a category c such that for some category below c (e.g., c itself) the action a is authorised on r, otherwise the request is denied. Other alternatives (e.g., the possibility of considering *undeterminate* as answer if there is not enough information to grant the request, which is quite natural when composing policies both in centralised and in distributed systems), will be discussed in the next section.

A range of access control models can be represented as specialised instances of this metamodel: see [19] 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 event-based model DEBAC and a Chinese Wall policy. RBAC, DAC and MAC are useful in centralised and mainly static environments. Event-based models such as DEBAC are

more appropriate for dynamic environments where the rights of users depend on the current state of the system and the events that have taken place.

Distributed-DEBAC, introduced in [18], is an extension of DEBAC for *distributed* systems, where resources may be located at various sites and access control policies may depend on events that take place at the site where the resource is located, or at the site where the user is located, or both. In many distributed systems a global access control policy is defined as a combination of local policies, and includes a mechanism to resolve the conflicts that may arise between different local policies. In contrast, the category-based metamodel defined in this section does not take these issues into account: all the relations defined are monolithic, and the axioms defining authorisations are valid in the whole system. In the next section we extend and refine the metamodel in order to deal in a uniform way with access control policies for distributed systems.

3. A Distributed Category-Based Metamodel

We consider the same sets of entities as in $\mathcal{M}$. The set of situational identifiers will include identifiers for sites (or locations) which will be associated to resources or policies. For simplicity we will assume that $\mathcal{S}$ is just the set of locations that compose the distributed system. In other words, each $s \in \mathcal{S}$ identifies one of the components of the distributed system, seen as a federation. The sets $\mathcal{P}, \mathcal{C}, \mathcal{A}, \mathcal{R}$ include, respectively, the principals, categories, actions and resources in any of the sites of the system.

In addition to principal-category assignments, permissions, and authorisations (modelled by the relations $\mathcal{PCA}$, $\mathcal{ARCA}$ and $\mathcal{PAR}$; see Section 2.2), we define a notion of forbidden operation (or banned action) on resources, modelled by the relation $\mathcal{BARCA}$, and a notion of non-authorised access, modelled 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:* $\mathcal{BAR} \subseteq P \times A \times \mathcal{R}$, 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 authorised nor denied (thus producing an undeterminate answer). 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, whether in a centralised or distributed access control system. Moreover, 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 authorisations (resp. prohibitions) that are valid in the site s. The relation $\mathcal{PAR}$ defining the global authorisation policy will be obtained by composing the local policies defined by the relations $\mathcal{PAR}_s$ and $\mathcal{BAR}_s$ as indicated below. For instance, $\mathcal{PAR}$ could be defined as a union, but more sophisticated combinations are possible, in particular if policies in different sites may contain conflicting information.

The axioms for the distributed metamodel are given below; they can be seen as an extension of the axioms that define $\mathcal{M}$ (see Definition 2.8). 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; for instance, one could define them to be the sets of principals (resp., categories, actions, resources) that participate in the relations $\mathcal{PCA}_s$, $\mathcal{ARCA}_s$, $\mathcal{BARCA}_s$.

Definition 3.1 (Distributed Axioms). *In a distributed environment, the category-based metamodel is defined by the following core axiom where we assume that there exists a reflexive-transitive relationship $\subseteq$ between categories; this can simply be equality, set inclusion (i.e., the set of principals assigned to $c \in \mathcal{C}$ is a subset of the set of principals assigned to $c' \in \mathcal{C}$), or an application specific relation may be used.*

$$(b1) \quad \forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R},\ \forall s \in \mathcal{S} \\ ((\exists c \in \mathcal{C}, \exists c' \in \mathcal{C}, (p,c) \in \mathcal{PCA}_s \wedge c \subseteq c' \ \wedge (a,r,c') \in \mathcal{ARCA}_s) \Leftrightarrow \\ (p,a,r) \in \mathcal{PAR}_s)$$

If the relation $\mathcal{BARCA}$ is admitted in a site s, then the following axioms should be included:

$$(c1) \quad \forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R},\ \forall s \in \mathcal{S} \\ ((\exists c \in \mathcal{C}, \exists c' \in \mathcal{C}, (p,c) \in \mathcal{PCA}_s \wedge c' \subseteq c \ \wedge (a,r,c') \in \mathcal{BARCA}_s) \Leftrightarrow \\ (p,a,r) \in \mathcal{BAR}_s)$$

$$(d1) \quad \forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R},\ \forall s \in \mathcal{S} \\ ((p,a,r) \notin \mathcal{PAR}_s \ \wedge \ (p,a,r) \notin \mathcal{BAR}_s) \Leftrightarrow (p,a,r) \in \mathcal{UNDET}_s$$

$$(e1) \quad \forall s \in \mathcal{S}, \quad \mathcal{PAR}_s \cap \mathcal{BAR}_s = \emptyset$$

Notice that access rights are inherited upwards through the hierarchy defined by the $\subseteq$ relation, while prohibitions are propagated downwards to the basis of the hierarchy.

Finally, the axioms below describe the global authorisation relation, which is obtained from the local authorisations and prohibitions defined at each site, by using operators $\mathcal{OP}_{par}$ and $\mathcal{OP}_{bar}$.

$$(f1) \quad \forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R}, \\ (p,a,r) \in \mathcal{OP}_{par}(\{\mathcal{PAR}_s, \mathcal{BAR}_s \mid s \in S\}) \Leftrightarrow (p,a,r) \in \mathcal{PAR}$$

$$(g1) \quad \forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R}, \\ (p,a,r) \in \mathcal{OP}_{bar}(\{\mathcal{PAR}_s, \mathcal{BAR}_s \mid s \in S\}) \Leftrightarrow (p,a,r) \in \mathcal{BAR}$$

$$(h1) \quad \mathcal{PAR} \cap \mathcal{BAR} = \emptyset$$

According to these axioms, the result of an access request may be different depending on the site where the request is evaluated, since each site has its own authorisation policy defined by the local relations $\mathcal{PAR}_s$ and $\mathcal{BAR}_s$ (see axioms $(b1)$ and $(c1)$). The relation $\mathcal{UNDET}_s \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{R}$ is such that $(p,a,r) \in \mathcal{UNDET}_s$ iff the action $a \in \mathcal{A}$ on resource $r \in \mathcal{R}$ is neither allowed nor forbidden for the principal $p \in \mathcal{P}$ at site $s \in \mathcal{S}$ (see axiom $(d1)$); this implies that every tuple in $\mathcal{P} \times \mathcal{A} \times \mathcal{R}$ is in $\mathcal{PAR}_s \cup \mathcal{BAR}_s \cup \mathcal{UNDET}_s$. The axioms $(e1)$ and $(h1)$ preclude inconsistent specifications (i.e., a request cannot be both authorised and forbidden).

The final authorisation is computed by specialising the definition of the operators $\mathcal{OP}_{par}$ and $\mathcal{OP}_{bar}$, according to the application requirements (axioms $(f1)$ and $(g1)$).

These operators take as parameter a set of local answers and combine them in order to produce a final access decision. They can give priority to positive authorisations, or to undeterminate (or negative) ones in case of conflict. While most of existing policy languages (e.g. XACML) have a fixed set of combination algorithms, our metamodel can be used to specify a large range of composition operators; we give examples below.

Example 3.2. *Consider a system with two sites $s, t \in \mathcal{S}$, where we define $\mathcal{OP}_{bar} = (\mathcal{BAR}_s \cup \mathcal{BAR}_t)$ and $\mathcal{OP}_{par} = ((\mathcal{PAR}_s / \mathcal{BAR}_t) \cup (\mathcal{PAR}_t / \mathcal{BAR}_s))$. This corresponds to a union operator giving priority to* deny*, according to the "deny takes precedence principle" [45] (i.e. access is denied if it is denied by any of the component policies).*

An alternative would be to use the precedence operator defined e.g. in [23]. 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 may need to be processed in another site of the system. We can model this kind of policy definition with the axioms above using the operator $\mathcal{OP}_{par} = (\mathcal{PAR}_s \cup (\mathcal{PAR}_t / \mathcal{BAR}_s))$ and $\mathcal{OP}_{bar} = (\mathcal{BAR}_s \cup (\mathcal{BAR}_t / \mathcal{PAR}_s))$.

The distributed metamodel refines and extends $\mathcal{M}$ as defined in Section 2.2 because different access control policies (possibly using different access control models*) may be defined at different sites, and access requests can be evaluated using any combination of local access control policy as well as a global, centrally defined policy. For instance, if the system has a central policy at site ς, then we can define $\mathcal{OP}_{par} = \mathcal{PAR}_\varsigma$ and $\mathcal{OP}_{bar} = \mathcal{BAR}_\varsigma$.*

4. Operational Semantics of the Distributed Metamodel

An important aspect of the distributed metamodel is the capability of representing systems where resources may be dispersed across different sites, and the information needed to decide whether a user request is granted or denied may also be distributed. The operational semantics of the distributed model will be defined by extending the functions presented in Definition 2.9 and in addition a new function barca, corresponding to the relation $\mathcal{BARCA}$ in Section 3, will be introduced to represent explicitly forbidden actions on resources. Specific functions defined in a particular site are denoted using the site identifier, as described in Section 2.1. We recall that functions with no site annotations are assumed to be defined locally.

4.1. Rewrite-Based Specification

The function barca_s returns the list of prohibitions assigned to a category in a site s: $\mathsf{barca}_s(\mathsf{c}) \rightarrow^* [(\mathsf{a}_1, \mathsf{r}_1), \ldots, (\mathsf{a}_\mathsf{n}, \mathsf{r}_\mathsf{n})]$.

If for a given category c, the pair (a, r) is neither in $\mathsf{arca}_s(\mathsf{c})$ nor in $\mathsf{barca}_s(\mathsf{c})$, 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. The rewrite-based specification of the extended axioms of Section 3 follows.

Definition 4.1. *In a distributed environment, the rewrite-based specification of the axiom (b1) in Def. 3.1 is given by the rewrite rule:*

$$(b2) \quad \mathsf{par}_s(p, a, r) \quad \rightarrow \quad if \ (a, r) \in \mathsf{arca}_s^*(\mathsf{below}(\mathsf{pca}_s(p))) \ then \ \mathsf{grant} \ else \ \mathsf{deny}$$

where the function $\in$ *is a membership operator on lists (see Section 2),* grant *and* deny *are answers, and* arca_s^* *is the function defining the assignment of privileges to categories, as in the previous section.*

If we consider the extended axioms $(c1)$, $(d1)$ *in Def. 3.1, which involve a* $\mathcal{BARCA}$ *relation, the function* par_s *is defined as:*

$$(c2, d2) \quad \mathsf{par}_s(p, a, r) \quad \rightarrow \quad \begin{array}{l} \textit{if } (a, r) \in \mathsf{arca}_s^*(\mathsf{below}(\mathsf{pca}_s(p))) \textit{ then } \mathsf{grant} \\ \textit{else if } (a, r) \in \mathsf{barca}_s^*(\mathsf{above}(\mathsf{pca}_s(p))) \textit{ then } \mathsf{deny} \\ \qquad \textit{else } \mathsf{undeterminate} \end{array}$$

where barca_s^* *generalises the previously mentioned function* barca_s *to take into account lists of categories instead of a single category:*

$$\mathsf{barca}^*(\mathsf{nil}) \rightarrow \mathsf{nil} \qquad \mathsf{barca}^*(\mathsf{cons}(c, l)) \rightarrow \mathsf{append}(\mathsf{barca}(c), \mathsf{barca}^*(l))$$

Intuitively, this means that if c' *is below* c*, than* c *inherits all the privileges of* c'*, while if* c' *is above* c *than* c *inherits all the prohibitions of* c'*. As already mentioned, the authorisation and prohibition sets should satisfy axiom* $(e1)$*, that is,* $\mathsf{arca}_s^*(\mathsf{below}(\mathsf{pca}_s(p))) \cap \mathsf{barca}_s^*(\mathsf{above}(\mathsf{pca}_s(p))) = \emptyset$*, but note that even if axiom* $(e1)$ *is not satisfied, the operational semantics defined for* par *above is consistent: it gives priority to authorisations.*

The axioms $(f1)$ *and* $(g1)$ *are realised by the following rewrite rule (implementing* $\mathcal{OP}_{par}$ *and* $\mathcal{OP}_{bar}$ *through the use of* par_s*; see rule* $(c2, d2)$*:*

$$(f2, g2) \quad \mathsf{authorised}(p, a, r, s_1, \ldots, s_n) \quad \rightarrow \quad \mathsf{fauth}(\mathsf{op}, \mathsf{par}_{s_1}(p, a, r), \ldots, \mathsf{par}_{s_n}(p, a, r))$$

where the function fauth *combines the results into a final answer according to the operator* op.

The axioms $(f1)$ and $(g1)$ can be implemented in several ways. The version chosen in the definition above corresponds to a very general rewrite rule that can be used for evaluating an access request in a single central site, if $n = 1$ and the operator op is the identity, as well as for evaluating combinations of answers (with a suitable operator op) from n different local policies. An alternative can be to specify an authorised rewrite rule for any specific combination operator. For example, if we consider two sites s_1 and s_2, for the precedence operator previously mentioned, we may have

$$\mathsf{authorised}(p, a, r, s_1, s_2) \quad \rightarrow \quad \begin{array}{l} \textit{if } \mathsf{par}_{s_1}(p, a, r) = \mathsf{grant} \textit{ or } \mathsf{par}_{s_1}(p, a, r) = \mathsf{deny} \\ \textit{then } \mathsf{par}_{s_1}(p, a, r) \textit{ else } \mathsf{par}_{s_2}(p, a, r) \end{array}$$

In this case priority is given to local evaluation in site s_1, evaluation in site s_2 being executed only when the local policy in s_1 is not able to give as answer grant or deny. However, when dealing with policy combinations, it is unlikely to find a unique evaluation strategy that works for every possible scenario. A suitable policy integration mechanism depends on the requirements of the application and the involved parties. Distributed evaluation of access requests and combination of authorisation answers is further discussed in the next section. An overview of the rules modelling the distributed version of $\mathcal{M}$ is given in Table 1.

Par_s	$\mathsf{par}(p,a,r)$	$\rightarrow$ *if* $(a,r) \in \mathsf{arca}^*(\mathsf{below}(\mathsf{pca}(p)))$ *then* grant *else if* $(a,r) \in \mathsf{barca}^*(\mathsf{above}(\mathsf{pca}(p)))$ *then* deny *else* undeterminate
$Arca^*$	$\mathsf{arca}^*(\mathsf{cons}(c,l))$	$\rightarrow \mathsf{append}(\mathsf{arca}(c), \mathsf{arca}^*(l))$
$Arca^*$	$\mathsf{arca}^*(\mathsf{nil})$	$\rightarrow \mathsf{nil}$
$Barca^*$	$\mathsf{barca}^*(\mathsf{cons}(c,l))$	$\rightarrow \mathsf{append}(\mathsf{barca}(c), \mathsf{barca}^*(l))$
$Barca^*$	$\mathsf{barca}^*(\mathsf{nil})$	$\rightarrow \mathsf{nil}$
Pca	$\mathsf{pca}(\mathsf{p})$	$\rightarrow [\mathsf{c}_1, \ldots, \mathsf{c}_k]$ (for each principal p)
$Arca$	$\mathsf{arca}(\mathsf{c})$	$\rightarrow [(\mathsf{a}_1, \mathsf{r}_1), \ldots, (\mathsf{a}_k, \mathsf{r}_k)]$ (for each category c)
$Barca$	$\mathsf{barca}(\mathsf{c})$	$\rightarrow [(\mathsf{a}_l, \mathsf{r}_l), \ldots, (\mathsf{a}_t, \mathsf{r}_t)]$ (for each category c)
$Above$	$\mathsf{above}([\mathsf{c}])$	$\rightarrow [\mathsf{c}, \mathsf{c}_1, \ldots, \mathsf{c}_n]$ (for each category c)
$Below$	$\mathsf{below}([\mathsf{c}])$	$\rightarrow [\mathsf{c}, \mathsf{c}'_1, \ldots, \mathsf{c}'_m]$ (for each category c)
Aut	$\mathsf{authorised}(p,a,r,s_1,\ldots,s_n)$	$\rightarrow \mathsf{fauth}(\mathsf{op}, \mathsf{par}_{s_1}(p,a,r), \ldots, \mathsf{par}_{s_n}(p,a,r))$
$Fauth$	$\mathsf{fauth}(\mathsf{op}, \mathsf{answ}_1, \ldots, \mathsf{answ}_n)$	$\rightarrow \mathsf{answ}_0$ (where $\mathsf{answ}_i \in \mathcal{A}\mathsf{uth}$)

Table 1: Rewrite Specification of the Distributed Metamodel: Generic Functions, Specific Functions, and Combination Rules

4.2. Evaluating access requests

The novelty of the distributed metamodel lies in the fact that different local policies, possibly based on different access control models, can be defined and combined in this framework in a smooth and uniform manner.

Evaluation initially takes place in the site where the request is issued, using the local function authorised (see rule *Aut* in Table 1). The local rule *Aut* specifies a number of sites s_i, $i = 1 \ldots n$, where the request may be passed and evaluated by the corresponding function par_{s_i}. For defining the sites of the evaluation, we may use functions like e.g. $\mathsf{psite}(p)$, which returns the site where the principal p is registered, or $\mathsf{rsite}(r)$ which returns the site where the resource r is located. In this way, access requests can be evaluated in a predefined central site, or priority can be given to local evaluation, or more elaborated combinations of access answers can be implemented.

In order to specify a particular policy at a site s, e.g. RBAC, MAC or DEBAC, it is sufficient to specialise locally the functions arca_s, barca_s, pca_s, above_s, below_s. Their definition in Table 1 can be adapted to the application we want to consider. Thus, for example, to express a hierarchical RBAC policy at site s, the function arca_s will return the permissions associated to each defined role and below_s will return the roles inferior to a given role, according to the hierarchy specified in the model (see [19] for more application examples). In the distributed metamodel, the request can be passed to other sites (with possibly different local policies) and evaluated in a distributed way. The generic rule (Aut), implementing the axioms of Section 3, is then used to integrate the different local access request answers to provide a final authorisation decision. More precisely, the distributed answers are collected and combined by using specific rules ($Fauth$), for a suitable operator op. We can specialise the combination rules ($Aut, Fauth$) using a variety of algebraic operators, as explained in the next section.

5. Integrating combination operators in the distributed metamodel

In the distributed metamodel we can combine evaluations from different sources in a flexible way by refining the definition of the fauth function (rule *Fauth*). Standard operators to compose policies, such as *Intersection*, *Subtraction* and *Union* can be easily defined in the metamodel, as we will show below. Further operators will be introduced in Section 7. We briefly give next some intuition on the semantics of the above operators, applied to sets of authorisation answers.

- Using the *Intersection* operator (I), access is authorised (denied) if it is allowed (denied) by each of the component policies.
- Using the *Subtraction* operator ($-$), access is authorised (denied) if it is allowed (denied) by the first policy, but not by the second one.
- Finally, using the *Union* operator (U) an access can be authorised if it is allowed by any of the component policies. Since we have explicit definition of prohibitions as well as authorisations, conflicts may arise. We consider three union operators, depending on the conflict-resolution method used: U^G (i.e. access can be authorised if it is allowed by any of the component policies), U^D (i.e. access is denied if it is denied by any of the component policies) and U^U (giving as answer undeterminate in case of conflicting information).

For example, consider a principal in an international organisation belonging to the U.K. division and asking for access to the French division. The access request will be evaluated in the U.K. site, where the principal is registered, and also in the French site (the two sites may have different policies). Access will be permitted if the policies in both sites return a grant answer (denied if at least one policy denies the access). This corresponds to the definition of the U^D operator, which can be modelled by the following fauth function:

$$
\begin{array}{lcl}
\mathsf{fauth}(\mathsf{U^D}, \mathsf{deny}, x) & \rightarrow & \mathsf{deny} \\
\mathsf{fauth}(\mathsf{U^D}, x, \mathsf{deny}) & \rightarrow & \mathsf{deny} \\
\mathsf{fauth}(\mathsf{U^D}, \mathsf{undeterminate}, \mathsf{undeterminate}) & \rightarrow & \mathsf{undeterminate} \\
\mathsf{fauth}(\mathsf{U^D}, \mathsf{grant}, \mathsf{grant}) & \rightarrow & \mathsf{grant} \\
\mathsf{fauth}(\mathsf{U^D}, \mathsf{undeterminate}, \mathsf{grant}) & \rightarrow & \mathsf{undeterminate} \\
\mathsf{fauth}(\mathsf{U^D}, \mathsf{grant}, \mathsf{undeterminate}) & \rightarrow & \mathsf{undeterminate}
\end{array}
$$

Following the same idea, the rewrite system can be adapted to the other mentioned operators. For instance, the U^G operator can be obtained by exchanging deny and grant in the previous (*Fauth*) rule specification. A precedence operator (giving priority to the result in the first site of the evaluation) may be defined as

$$
\begin{array}{lcl}
\mathsf{fauth}(\mathsf{Lp}, \mathsf{grant}, x) & \rightarrow & \mathsf{grant} \\
\mathsf{fauth}(\mathsf{Lp}, \mathsf{deny}, x) & \rightarrow & \mathsf{deny} \\
\mathsf{fauth}(\mathsf{Lp}, \mathsf{undeterminate}, x) & \rightarrow & x
\end{array}
$$

For the sake of readability, the above rules have been presented using binary operators, but the generalisation to the combination of any number $n > 0$ of policies can be easily obtained. Moreover, in the same style, the combination algorithms of XACML policy

language [55, 51], such as *Permit-overrides, Deny-overrides, First-applicable, Only-one-applicable* can be specified in our framework using recursive rules (for a complete specification see [16]). One can note that in the setting of a three-valued policy (as we consider here) the XACML *Undeterminate* and *NotApplicable* results are treated in an equivalent way. We can easily obtain a four-valued policy by enlarging the set of answers $\mathcal{A}uth$ and by modifying the definition of the specific (*Fauth*) rule accordingly.

To facilitate the declarative definition of more sophisticated operators, it is useful to add higher-order features to the specification language. In Section 7 we describe a higher-order extension of the language along the lines of [16], in order to include policy combination operators such as override, closure and scope restriction (as defined e.g. in [23, 61]) and further policy combination expressions including such operators.

To illustrate the expressive power of the framework, we give below the specification of a Distributed-DEBAC policy [18] using the presented metamodel.

Example 5.1 (Distributed-DEBAC: bank scenario).
In Distributed-DEBAC, the users' rights of access vary depending on the history of events that have taken place in the system. Moreover, in this access control model operators may refer to sites explicitly through site identifiers, or use an implicit reference that will be solved dynamically and by autonomous means (for this feature see Example 7.2). This feature is useful in distributed systems where the configuration (i.e., the set of sites) is not static. Access to resources is defined as follows:

> *A user* u *is permitted to perform an action* a *on a resource* r *located at site* s *iff* u *is assigned (locally or centrally), using the (local or central) events history* h*, to a category* c *to which the permission to do the action* a *on* r *has been assigned.*

It is sufficient using our distributed metamodel to specify locally in every site s of the system a DEBAC policy, using the appropriate rewrite rules for computing the category of a principal p. Then, the par *rule will evaluate the access request according to the category permissions specified by the function* arca*. The local and the central evaluation are finally combined by the* authorised *function, which will give the final access authorisation result.*

We give a concrete example, adapted from the bank scenario described in [18].

Consider a bank where clients can perform actions such as deposits and withdrawals on their accounts, requests of loans, and so on. The bank has several sites, say a central bank site ς and several local branches. The bank's access control policy gives priority to local DEBAC branch policies and transfers evaluation to the central site only if the request cannot be treated locally. Assume that a principal p *asks for a loan at his branch and that, according to the policy specification, only a* Loyal-Client *can get a loan. In our metamodel, this scenario corresponds to the use of the operator* Lp *(defined above) for the combination of the DEBAC policies.*

$$\mathsf{authorised}(\mathsf{p}, \mathsf{GetLoan}, \mathsf{Bank}, \mathsf{psite}(\mathsf{p}), \varsigma) \rightarrow$$
$$\mathsf{fauth}(\mathsf{Lp}, \mathsf{par}_{\mathsf{psite}(\mathsf{p})}(\mathsf{p}, \mathsf{GetLoan}, \mathsf{Bank}), \mathsf{par}_{\varsigma}(\mathsf{p}, \mathsf{GetLoan}, \mathsf{Bank}))$$

Assuming $\mathsf{psite}(\mathsf{p}) \rightarrow \pi$*, the function* par_{π} *calls the local function* pca_{π} *to compute the category associated to principal* p*. Suppose that local evaluation returns as result undeterminate (e.g., in site π,* p *is associated to a category* Client *to which a loan request is*

neither permitted nor prohibited):

$$\mathsf{par}_{\mathsf{psite(p)}}(\mathsf{p}, \mathsf{GetLoan}, \mathsf{Bank}) \rightarrow^* \mathsf{undeterminate}$$

We analyse now the request in the central site ς*. The DEBAC policy defined in* ς *computes the category of a principal according to a history of events* h_ς*, as shown below. In such history, among others, we may have events affecting the privileges of the principal, such as the buying of an insurance, which may turn him into a loyal client:*

$$\begin{array}{l}\mathsf{pca}(p) \rightarrow \mathsf{categ}(p, \mathsf{h})\\ \mathsf{categ}(p, \mathsf{nil}) \rightarrow [\mathsf{Client}]\\ \mathsf{categ}(p, \mathsf{cons}(\mathsf{event}(e, p, \mathsf{buyInsurance}, t), h)) \rightarrow\\ if\ \mathsf{averagebalance}(\mathsf{account}(p)) > 10000\ \mathsf{and}\ \mathsf{notBlacklisted}_\mu(p, \mathsf{today})\\ then\ [\mathsf{Loyal\text{-}Client}]\ else\ [\mathsf{Client}]\end{array}$$

where we use a Boolean function notBlacklisted *defined in an external site* μ*. All the other functions are local.*

So, supposing we have $\mathsf{pca}_\varsigma(p) \rightarrow [\mathsf{Loyal\text{-}Client}]$ *according to the central DEBAC policy, since we assume* $\mathsf{arca}_\varsigma(\mathsf{Loyal\text{-}Client}) \rightarrow [(\mathit{GetLoan}, \mathit{Bank})]$ *we have*

$$\mathsf{par}_\varsigma(\mathsf{p}, \mathsf{GetLoan}, \mathsf{Bank})) \rightarrow^* \mathsf{grant}$$

and thus there is a reduction

$$\mathsf{authorised}(\mathsf{p}, \mathsf{GetLoan}, \mathsf{Bank}, \mathsf{psite}(\mathsf{p}), \varsigma) \rightarrow^* \mathsf{fauth}(\mathsf{Lp}, \mathsf{undeterminate}, \mathsf{grant}) \rightarrow \mathsf{grant}$$

The next example illustrates the definition of a combination of RBAC policies in a distributed organisation, as an instance of the metamodel.

Example 5.2. *(Shared agenda)*
Consider a principal working in an organisation consisting of the "ordering" and the "delivery" departments where employees share an electronic agenda a*. Suppose each department in this organisation adopts an RBAC policy for the employees. Consider the request of editing a section* $\mathsf{a_s}$ *of the agenda by the principal* p*. In order to access the agenda,* p *has to be an employee of the organisation (registered in any of the two departments). In the metamodel, we will use a union operator giving priority to grant* U^G *(see Section 4.2) to meet the requirements of this distributed scenario.*

At the local level, we have an RBAC policy implemented in the site corresponding to the ordering department, say π_1*, and another RBAC policy implemented in the site corresponding to the delivering department, say* π_2*. If principal* p *works in the delivering department, the policy in* π_2 *provides rules such as*

$$\begin{array}{l}\mathsf{pca}_{\pi_2}(\mathsf{p}) \rightarrow [\mathsf{employee}]\\ \mathsf{arca}_{\pi_2}(\mathsf{employee}) \rightarrow [(\mathsf{read}, \mathsf{order}), (\mathsf{execute}, \mathsf{delivery}), (\mathsf{write}, \mathsf{a_s}), (\mathsf{read}, \mathsf{a_s})]\\ \mathsf{barca}_{\pi_2}(\mathsf{employee}) \rightarrow [(\mathsf{modify}, \mathsf{order}), (\mathsf{cancel}, \mathsf{delivery})]\end{array}$$

On the other hand, our principal p *is not registered in site* π_1*. We suppose this is translated at the policy level as the assignment of* p *to the default category* unknown *in* π_1*, which has no associated privileges nor prohibitions.*

The request evaluation starts by calling the authorised *function local to the site where the request is issued. Evaluation is then passed to the departmental sites* π_1 *and* π_2*.*

$$\mathsf{authorised}(\mathsf{p}, \mathsf{write}, \mathsf{a_s}, \pi_1, \pi_2)) \rightarrow^* \mathsf{fauth}(\mathsf{U^G}, \mathsf{par}_{\pi_1}(\mathsf{p}, \mathsf{write}, \mathsf{a_s}), \mathsf{par}_{\pi_2}(\mathsf{p}, \mathsf{write}, \mathsf{a_s}))$$

We consider first the evaluation of the request in the site π_1*. In this case, we have* $\mathsf{par}_{\pi_1}(\mathsf{p}, \mathsf{write}, \mathsf{a_s}) \rightarrow^* \mathsf{undeterminate}$ *since the pair* $(\mathsf{write}, \mathsf{a_s})$ *is not in the list of permissions or prohibitions of the category* unknown *to which the principal belongs.*

We consider now the evaluation of the request locally to site π_2*.*

We have $\mathsf{par}_{\pi_2}(\mathsf{p}, \mathsf{write}, \mathsf{a_s}) \rightarrow^* \mathsf{grant}$ *since* p *is indeed an employee of the organisation and as such has access to the shared agenda. Thus finally the request of principal* p *to edit the section* s *of the agenda will be granted*

$$\mathsf{authorised}(\mathsf{p}, \mathsf{write}, \mathsf{a_s}, \pi_1, \pi_2) \rightarrow^* \mathsf{fauth}(\mathsf{U^G}, \mathsf{undeterminate}, \mathsf{grant}) \rightarrow^* \mathsf{grant}$$

Note that the same principal may belong to different categories as the system evolves (as in the McLean model [53], where the principal's category depends on the current accesses at a given moment). The specific function pca should be designed to reflect this; for instance, as in Example 5.1, pca could inspect the history of events in the system (accesses in this case) to compute a category for the principal.

6. Policy Analysis: Proving Properties of Policies

An access control model defined as an instance of the metamodel provides a notion of category and the corresponding definitions of the relations between categories, between principal and categories and between categories and permissions. An access control policy can then be defined in terms of the chosen access control model.

Specifying access control policies via term rewriting systems, which have a formal semantics, has the advantage that this representation admits the possibility of proving properties of policies, and this is essential for policy acceptability [56]. Rewriting properties, such as 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. More specifically, we are interested in the following properties of access control policies.

Totality: Each access request from a principal p to perform an action a on a resource r receives an answer (e.g., grant, deny, undeterminate).

Consistency: For each access request from a principal p to perform an action a on a resource r at most one result can be obtained.

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 if and only if p belongs to a category c such that $c \subseteq c'$ and c' has the permission (a, r).

Totality and consistency can be proved, for policies defined as rewrite systems, by checking that the rewrite relation generated by the rules used in a specific instance of the metamodel is confluent and terminating. Termination ensures that all access requests produce a result (e.g. a result that is not grant or deny is interpreted as undeterminate) and confluence ensures that this result is unique. The soundness and completeness of a policy can be checked by analysing the normal forms of access requests. Sufficient

completeness of the rewrite rules (a property that ensures that each operation is defined on all valid inputs) guarantees that the normal forms are of the right form [42].

Confluence and termination of rewriting are undecidable properties in general, but there are several sufficient conditions for these properties to hold. We define next a condition that will be used to prove that policy specifications defined as first-order instances of the metamodel satisfy the properties we are interested in.

Definition 6.1 (Safe system). *A rewrite system R is safe if*

1. *R is non-overlapping;*
2. *R is a constructor system;*
3. *if a rewrite rule in R defines a recursive function* f*, then the right-hand side is built out of variables, constructors, previously defined functions, or recursive calls to* f *on arguments which are smaller (with respect to a well founded ordering, e.g. subterm) than the ones in its left-hand side.*

Conditions 1. and 2. in the definition of safe systems are natural: they require functions to be defined by cases on data constructors that do not overlap. They are standard conditions in functional programs. Condition 3. will be used to ensure termination [39]. We will show that to ensure that a policy specification obtained as an instance of the metamodel is consistent and total it is *sufficient* to check that the rules defining specific functions satisfy the safeness conditions. All the examples given in this paper satisfy these conditions.

Theorem 6.2. *Assume R defines an access control policy as an instance of the metamodel, using the rules in Table 1 with additional, specific rules satisfying the safeness conditions. Then, R is terminating and confluent.*

Proof 6.3. 1. *Termination: First, observe that using the metamodel the full definition of the policy can be seen as a* hierarchical *rewrite system, where the basis includes the set of constants identifying the main entities in the model (e.g., principals, categories, etc.) as well as the set of auxiliary basic data (such as Booleans and numbers) and functions on them (*if-then-else, and*). The next level in the hierarchy includes all the auxiliary functions on lists (such as* append*) and the parameter functions of the model, that is, the specific functions* pca, arca, barca, above, below, $\in$*. The functions* arca^*, barca^*, *and* fauth *form the next level, followed by the definition of the function* par*. Finally the last level of the hierarchy consists of the definition of the function* authorised.

Several sufficient conditions for termination of rewrite systems defined as a hierarchical union of rules are available. For instance, we can use the following modularity result: a hierarchical *term rewriting system is terminating if the basis of the hierarchy is terminating and non-duplicating (i.e., rules do not duplicate variables in the right-hand side) and in the next levels of the hierarchy the recursive functions are defined, using previously defined functions, by rules that satisfy a general scheme of recursion, where recursive calls on the right-hand sides of rules are made on subterms of the left-hand side and there are no mutually recursive functions [39]. The rules given in Table 1 satisfy the required conditions (notice that the functions* par *and* authorised *are not recursive), and the assumption of safeness ensures the latter condition is satisfied for the additional rules used in a specific instance of the metamodel. We conclude that the system is terminating.*

2. *To prove confluence, first note that the rules defining generic functions in Table 1 are non-overlapping, and the same holds for specific functions by the safeness assumption. Moreover, since the sets of generic and specific functions are disjoint, and the patterns are constructor terms, there are no critical pairs in the system, and therefore the system is locally confluent. Termination and local confluence imply confluence, by Newman's Lemma [54].*

Confluence can also be obtained in other ways. For instance, orthogonal systems are confluent, as shown by Klop [47].

Corollary 6.4. *Assume R defines an access control policy as an instance of the metamodel, using the generic rules in Table 1 and additional, specific rules satisfying the safeness conditions. Then, every term has a unique normal form.*

As a consequence of the unicity of normal forms, the policy specification is *consistent*.

Property 6.5 (Consistency). *Assume R defines an access control policy as an instance of the metamodel, using the generic rules in Table 1 and additional, specific rules satisfying the safeness conditions. It is not possible to derive, from R, both* grant *and* deny *for a request* $\mathsf{authorised}(\mathsf{p}, \mathsf{a}, \mathsf{r}, \mathsf{s}_1, \ldots, \mathsf{s}_n)$.

Proof 6.6. *Follows from Corollary 6.4.*

We give next a precise characterisation of the normal forms of access requests.

Proposition 6.7. *Assuming the specific functions used in an instance of the metamodel (specific versions of* pca, arca, barca, above, below, fauth*) are well-defined (i.e., their evaluation produces a result provided the arguments are valid; for instance, the evaluation of a ground term* $\mathsf{pca}(\mathsf{p})$ *results in a list of categories for any principal* p, $\mathsf{above}(\mathsf{c})$ *and* $\mathsf{below}(\mathsf{c})$ *produce lists of categories for any category* c, $\mathsf{arca}(\mathsf{c})$ *and* $\mathsf{barca}(\mathsf{c})$ *produce lists of pairs* (a, r), fauth *produces a result in $\mathcal{A}uth$), then the normal form of a ground term* $\mathsf{authorised}(\mathsf{p}, \mathsf{a}, \mathsf{r}, \mathsf{s}_1, \ldots, \mathsf{s}_n)$ *where* $\mathsf{p} \in \mathcal{P}$, $\mathsf{a} \in \mathcal{A}$, $\mathsf{r} \in \mathcal{R}$, $\mathsf{s}_1, \ldots, \mathsf{s}_n \in \mathcal{S}$, *is in $\mathcal{A}uth$.*

Proof 6.8. *It follows from the fact that the function* par *produces the expected result if* pca, arca, barca, above, below *are well defined.*

The evaluation of $\mathsf{authorised}(\mathsf{p}, \mathsf{a}, \mathsf{r}, \mathsf{s}_1, \ldots, \mathsf{s}_n)$ *relies on* par *and* fauth*, also well defined by assumption; it returns a result in $\mathcal{A}uth$.*

As a consequence, our specification of the access control policy is *total*.

Property 6.9 (Totality). *Assuming the specific functions used in an instance of the metamodel (specific versions of* pca, arca, barca, above, below, fauth*) satisfy the safeness condition and are well-defined (i.e., the evaluation of a ground term* $\mathsf{pca}(\mathsf{p})$ *results in a category for any principal* p*, and similarly for the other functions, as described in Prop. 6.7), each request* $\mathsf{authorised}(\mathsf{p}, \mathsf{a}, \mathsf{r}, \mathsf{s}_1, \ldots, \mathsf{s}_n)$ *receives an answer in $\mathcal{A}uth$.*

Proof 6.10. *Follows from Proposition 6.7 and Theorem 6.2.*

For instance, the policies defined in the examples in this paper (e.g. see examples in Section 5) are consistent and total.

Soundness and *Completeness* are also easy to check for the examples.

Property 6.11 (Soundness and Completeness). *The policies specified by first-order systems in all the previous examples are sound and complete. More precisely, for any* $p \in \mathcal{P}$, $a \in \mathcal{A}$, *and* $r \in \mathcal{R}$, $s_1, \ldots, s_n \in \mathcal{S}$:

1. $\mathsf{authorised}(p, a, r, s_1, \ldots, s_n) \rightarrow^* \mathsf{grant}$ *if and only if* p *belongs to a category* c *such that there exists a category* c' *below* c *that has the access privilege* a *on* r.
2. $\mathsf{authorised}(p, a, r, s_1, \ldots, s_n) \rightarrow^* \mathsf{deny}$ *if and only if* p *belongs to a category* c *such that there exists a category* c' *above* c *for which the access privilege* a *on* r *is forbidden.*

Proof 6.12. *We check that (1) holds by inspecting the rewrite rules in the examples. Then, since* $\mathsf{arca}^*_s(\mathsf{below}(\mathsf{pca}_s(p))) \cap \mathsf{barca}^*_s(\mathsf{above}(\mathsf{pca}_s(p))) = \emptyset$ *for any* p *by axiom* $(e1)$ *(see Definition 4.1), we deduce (2).*

7. Adding Higher-Order Features to the Specification Language

In addition to the standard set operators presented in Section 5, we can model powerful policy combination operators by means of higher-order functions. This allows us to conveniently express a wide range of combinations in a uniform setting. For example, we can model the restriction, template and override operators defined in [23, 61]. Restriction constrains the application of a policy to the requests satisfying certain conditions. Templates are used to define partially specified policies, that can be completed by supplying the missing parameters. Override replaces a part of a policy with a fragment of a second policy. The portion to be replaced is specified using a third policy.

The override operator can be expressed using a combination of the *Union* (U), *Substraction* ($-$) and *Intersection* (I) operators of Section 5, obtaining an expression of the form $(a_1 - a_3)\mathsf{U}^{\mathsf{G}}(a_2 \mathsf{I} a_3)$, where a_i is the access control answer returned by the policy i (see [16] for more details). We are able to model the expression above, and any other expression mixing the mentioned operators, by introducing in the specification language a (higher-order) function f which encodes the structure of the policy combinator we want to specify and contains variables that will be instantiated by the actual policy parameters at run-time (by β-reduction).

Formally, we introduce in the metamodel a rule that we call *Ho-Auth*:

Ho-Auth	$\mathsf{hoAuth}(f, p, a, r, s_1, \ldots, s_n) \quad \rightarrow \quad f\ s_1\ \ldots\ s_n\ p\ a\ r$

where the variable f will be instantiated by a λ-abstraction of the form $\lambda \overline{s}' p' a' r'.e$: the bound variables $\overline{s}'$ represent the sites $s_1, \ldots, s_n$ involved in the evaluation, p', a', r' are variables to be instantiated by the actual parameters of the access request, that is the principal, the action and the resource, and e is a term specifying how the local policies are combined (in order to evaluate correctly, the term that instantiates f should not have free variables, that is, only $\overline{s}', p', a', r'$ may occur free in e, and when applied to the actual parameters of the access request it must have as normal form an authorisation answer in $\mathcal{A}uth$). We give examples below.

This additional expressive power allows us to generalise the *Aut* rule. We recall that in a completely distributed system, every local policy specifies its own *Aut* rule as well

as the way combinations with external policies have to be performed. This means that, for a site s, we may have

$$\begin{array}{lcl} \mathsf{authorised}(p,a,r,s_1,\ldots,s_n) & \rightarrow & \mathsf{fauth}(\mathsf{op},\mathsf{par}_{s_1}(p,a,r),\ldots,\mathsf{par}_{s_n}(p,a,r)) \\ & \textit{or} & \\ & \rightarrow & \mathsf{hoAuth}(\mathsf{f},p,a,r,s_1,\ldots,s_n) \end{array}$$

For example, to specify the override operator for the policies local to sites s_1, s_2, s_3, we use $\mathsf{f} = \lambda s_1 s_2 s_3 p' a' r'.e$ where $e = \mathsf{fauth}(\mathsf{U}^{\mathsf{G}}, e1, e2)$ with

$e1 = \mathsf{fauth}(-, \mathsf{par}_{s_1}(p', a', r'), \mathsf{par}_{s_3}(p', a', r'))$, and

$e2 = (\mathsf{fauth}(\mathsf{I}, \mathsf{par}_{s_2}(p', a', r'), \mathsf{par}_{s_3}(p', a', r'))$.

An example of application of this higher-order extension of the metamodel is given next.

Example 7.1. *(Shared agenda continued)*
Recall Eample 5.2 and suppose now that the organisation adopts a specific Bell-Lapadula policy on the agenda a*, maintained on a server* ν*. Consider the request of editing the agenda by the principal* p*. In this case,* p *has to be an employee of the organisation to access the agenda (as in Example 5.2) and moreover* p *must respect the "no read up" and "write only at the subject level" rules of the Bell-Lapadula policy in order to edit it. Thus, we have to compose three policies, the two departmental policies plus the agenda policy. Next, we will denote the term* $\mathsf{fauth}(\mathsf{U}^{\mathsf{G}}, \mathsf{par}_{\pi_1}(\mathsf{p}, \mathsf{write}, \mathsf{a_s}), \mathsf{par}_{\pi_2}(\mathsf{p}, \mathsf{write}, \mathsf{a_s}))$ *(which evaluates to* grant*, as seen in Example 5.2) by* A_{dep}*.*

We will use the higher-order extension of the metamodel to meet the requirements of this distributed scenario. At the local level, we have a Bell-Lapadula policy implemented in site ν*. Assume we have three different levels (top-secret, secret and public) and three different sections of the agenda* ts*,* s *and* p *that can be modified according to the category associated to the principal. The privileges depend thus on the secrecy level:*

$$\begin{array}{l} \mathsf{arca}_\nu(\mathsf{top\_secret}) \rightarrow [(\mathsf{read},\mathsf{a_{ts}}),(\mathsf{write},\mathsf{a_{ts}}),(\mathsf{read},\mathsf{a_s}),(\mathsf{read},\mathsf{a_p}),] \\ \mathsf{barca}_\nu(\mathsf{top\_secret}) \rightarrow [(\mathsf{write},\mathsf{a_s}),(\mathsf{write},\mathsf{a_p})] \\ \ldots \\ \mathsf{arca}_\nu(\mathsf{public}) \rightarrow [(\mathsf{write},\mathsf{a_p}),(\mathsf{read},\mathsf{a_p}),] \\ \mathsf{barca}_\nu(\mathsf{public}) \rightarrow [(\mathsf{write},\mathsf{a_s}),(\mathsf{write},\mathsf{a_{ts}}),(\mathsf{read},\mathsf{a_s}),(\mathsf{read},\mathsf{a_{ts}})] \end{array}$$

Assume p *is assigned to the public level by* $\mathsf{pca}_\nu(\mathsf{p}) \rightarrow [\mathsf{public}]$*, and asks for editing a section* $\mathsf{a_s}$ *in the agenda.*

The request evaluation starts by calling the $\mathsf{authorised}$ *function local to the site where the request is issued. The reduction goes as follows, where* f *is the* λ*-term using the operator* U^{D} *to combine the previous* A_{dep} *result with the Bell-Lapadula policy result.*

$$\begin{array}{l} \mathsf{authorised}(\mathsf{p},\mathsf{write},\mathsf{a_s},\pi_1,\pi_2,\nu) \rightarrow^* \mathsf{hoAuth}(\mathsf{f},\mathsf{p},\mathsf{write},\mathsf{a_s},\pi_1,\pi_2,\nu) \\ \rightarrow^* \mathsf{fauth}(\mathsf{U}^{\mathsf{D}},\mathsf{par}_\nu(\mathsf{p},\mathsf{write},\mathsf{a_s}),A_{dep}) \rightarrow^* \mathsf{fauth}(\mathsf{U}^{\mathsf{D}},\mathsf{par}_\nu(\mathsf{p},\mathsf{write},\mathsf{a_s}),\mathsf{grant}) \end{array}$$

We consider now the evaluation of the request which is performed locally to site ν*. We have* $\mathsf{par}_\nu(\mathsf{p},\mathsf{write},\mathsf{a_s}) \rightarrow^* \mathsf{deny}$ *since the pair* $(\mathsf{write},\mathsf{a_s})$ *is in the list of prohibitions of the category* public *to which the principal belongs. Thus finally the access to principal* p *at the secret section of the agenda will be denied:*

$$\mathsf{authorised}(\mathsf{p},\mathsf{write},\mathsf{a_s},\pi_1,\pi_2,\nu) \rightarrow^* \mathsf{fauth}(\mathsf{U}^{\mathsf{D}},\mathsf{deny},\mathsf{grant}) \rightarrow^* \mathsf{deny}$$

Example 7.2. *(Distributed DEBAC continued)*
We show now that the higher-order features we added to our framework allow us to fully express the Distributed DEBAC model. In Example 5.1, we have seen how central and local evaluation of Distributed DEBAC can be simulated using the (Auth) rule of the metamodel. We treated the case where site identifiers are given explicitly, or through terms reducing to a site identifier in $\mathcal{S}$. The new higher-order features allow us to specify in the metamodel the full Distributed DEBAC approach that simplifies maintenance operations by using a (higher-order) function Get *and a placeholder "$\diamond$" instead of a particular function in the right-hand side of the rule defining* categ *(as detailed in [18]). Before the evaluation of the function* pca*, placeholders are automatically replaced by calling the function* Get*.* Get *takes as parameters an event and a site, and returns a (list of) λ-term(s) containing the information on the functions to call with the appropriate site identifier. Using this approach, in Example 5.1 we would have:*

$$\begin{array}{l}\mathsf{pca}(p) \rightarrow \mathsf{categ}(p, \mathsf{h})\\ \mathsf{categ}(p, \mathsf{nil}) \rightarrow [\mathsf{Client}]\\ \mathsf{categ}(p, \mathsf{cons}(\mathsf{event}(e, p, \mathsf{buyInsurance}, t), h)) \rightarrow\\ \mathit{if}\ \mathsf{averagebalance}(\mathsf{account}(p)) > 10000\ \mathit{and}\ \diamond(p, \mathsf{today})\\ \mathit{then}\ [\mathsf{Loyal\text{-}Client}]\ \mathit{else}\ [\mathsf{Client}]\end{array}$$

and a Get *function defined in the central site ς including the rule*

$$\mathsf{Get}(\mathsf{buyInsurance}, \pi) \rightarrow [\lambda xy.\mathsf{notBlackListed}_{\mu}(x, y)]$$

which evaluates to the same definition of the rule categ *as in Example 5.1. Here we are calling* Get *from site π to obtain the function needed to deal with the event associated to p buying an insurance.*

8. Policy Analysis: Higher-Order Operators

As mentioned in Section 6, an access control policy must satisfy certain criteria to be "acceptable". For example, it should provide answers for all requests from principals to perform actions on resources, and it should not produce contradictory answers, i.e., the policy should be consistent. A key observation is that in a terminating rewriting system each term has a normal form and in a confluent system this normal form is unique. Thus, in Section 6 we derived sufficient conditions for consistency and totality of the policy from sufficient conditions for confluence and termination of the rewriting system defining the policy.

In this section we give sufficient conditions to obtain the required properties of policies specified as instances of the metamodel using the higher-order framework described above.

We first remark that the rewrite rules used to specify the functions of the metamodel (see Table 1) are non-overlapping and have linear, algebraic left-hand sides. In other words, the rules defining the generic functions do not reuse variables in the left-hand side, and all functions are defined by cases on constructors, without superpositions. Left-linear and non-overlapping rules are confluent, even in the case of higher-order rewrite systems, as proved by Klop (see, for instance, [47]). Thus, we can give a sufficient condition for confluence derived from Klop's orthogonality theorem.

Theorem 8.1. *If the specific functions in a policy obtained as an instance of the meta-model are defined using left-linear rules satisfying the safeness conditions* 1. *and* 2. *(see Definition 6.1), then the TRS_λ defining the policy is confluent.*

Proof 8.2. *Since the patterns in the left-hand sides are constructor terms, there are no superpositions between generic and specific rules. Moreover, by inspection of the rules, it is easy to see that the generic rules do not overlap. The specific ones are non-overlapping by assumption, so the union of the two sets of rules is left-linear and non-overlapping, therefore confluent. We can now deduce that the whole policy, including the β-rule, is confluent, because it is defined by an orthogonal system [47].*

As a consequence, the policy defined by the TRS_λ is consistent.

We can derive sufficient conditions for termination of policies using those given in [16] for policies defined via typeable TRS_λ. The idea is to ensure termination by requiring two properties of the rewrite system: typeability and a safe form of recursion, inspired by the primitive recursive scheme. Typeability will be checked by means of a type assignment system.

The type assignment system given below is based on the system defined in [6] using intersection types. Intersection type systems are formal systems for assigning types to untyped terms, using a set of type inference rules. They were originally devised for the λ-calculus [30]. In our examples, the use of intersection types is not essential, but it allows more terms to be typed.

Definition 8.3 (Types). *Let Sorts be a set of names of domains, and $\mathcal{V}$ be a set of* type variables. *The set $\mathcal{T}_{\wedge\mathcal{S}}$ of types is inductively defined as follows:*

- *Constant types (sorts): If $s \in Sorts$ then $s \in \mathcal{T}_{\wedge\mathcal{S}}$.*
- *Type variables: If $\varphi \in \mathcal{V}$ then $\varphi \in \mathcal{T}_{\wedge\mathcal{S}}$.*
- *Arrow types: If $\sigma, \tau \in \mathcal{T}_{\wedge\mathcal{S}}$ then $\sigma \to \tau \in \mathcal{T}_{\wedge\mathcal{S}}$.*
- *Intersection types: If $\sigma, \tau \in \mathcal{T}_{\wedge\mathcal{S}}$ then $\sigma \wedge \tau \in \mathcal{T}_{\wedge\mathcal{S}}$.*

A type is algebraic *if it contains neither $\wedge$ nor type variables. We denote by $\mathcal{T}_\mathcal{S}$ the set of algebraic types.*

We will consider types modulo associativity, commutativity and idempotency of the type operator $\wedge$; as usual $\to$ associates to the right.

Definition 8.4 (Basis). *Let t be a term in $T_\lambda(\mathcal{F}, \mathcal{X})$. A* statement *is an expression of the form $t\colon \sigma$ where $\sigma \in \mathcal{T}_{\wedge\mathcal{S}}$; t is the* subject *of the statement.*

A basis *(the set of assumptions a statement depends on) is a set of statements with only variables as subjects. Moreover there are no two statements with the same subject. If x does not occur in the basis B then $B, x\colon \sigma$ denotes the basis $B \cup \{x\colon \sigma\}$. A basis is* algebraic *if all the types occurring in it are so.*

A set $\mathcal{A}x$ of axiom statements *for $\mathcal{F}_\lambda$ is a set of statements of the form*

$$\mathcal{A}x = \{\mathsf{f}\colon \sigma_1 \to \ldots \to \sigma_n \to \sigma \quad | \, \mathsf{f} \in \mathcal{F}_\lambda - \{\mathsf{Ap}\}, arity(\mathsf{f}) = n, \sigma_1, \ldots, \sigma_n, \sigma \in \mathcal{T}_\mathcal{S}\}$$

Table 2: Type inference rules

Var	$B, x{:}\sigma \vdash^{\wedge}_{\mathcal{A}x} x\colon \sigma$	$\mathcal{F}$	$\dfrac{B \vdash^{\wedge}_{\mathcal{A}x} t_i\colon \sigma_i (1 \leq i \leq n)}{B \vdash^{\wedge}_{\mathcal{A}x} \mathsf{f}(t_1, \ldots, t_n)\colon \sigma}$ if $\mathsf{f}\colon \sigma_1 \to \ldots \to \sigma_n \to \sigma \in \mathcal{A}x$
$(\to I)$	$\dfrac{B, x{:}\sigma \vdash^{\wedge}_{\mathcal{A}x} t\colon \tau}{B \vdash^{\wedge}_{\mathcal{A}x} \lambda x.t\colon \sigma \to \tau}$	$(\to E)$	$\dfrac{B \vdash^{\wedge}_{\mathcal{A}x} t\colon \sigma \to \tau \quad B \vdash^{\wedge}_{\mathcal{A}x} u\colon \sigma}{B \vdash^{\wedge}_{\mathcal{A}x} (tu)\colon \tau}$
$(\wedge I)$	$\dfrac{B \vdash^{\wedge}_{\mathcal{A}x} t\colon \sigma \quad B \vdash^{\wedge}_{\mathcal{A}x} t\colon \tau}{B \vdash^{\wedge}_{\mathcal{A}x} t\colon \sigma \wedge \tau}$	$(\wedge E)$	$\dfrac{B \vdash^{\wedge}_{\mathcal{A}x} t\colon \sigma \wedge \tau}{B \vdash^{\wedge}_{\mathcal{A}x} t\colon \sigma}$

Definition 8.5. *A term $t \in T_\lambda(\mathcal{F}, \mathcal{X})$ is* typeable *with respect to a set $\mathcal{A}x$ of axiom statements for $\mathcal{F}_\lambda$ if there exists a basis B and a type σ such that $B \vdash^{\wedge}_{\mathcal{A}x} t{:}\sigma$ is derivable by means of the inference rules given in Table 2, which include an axiom for variables (Var) and standard inference rules for applications ($\to_E$), λ-abstractions ($\to_I$) and functional terms ($\mathcal{F}$), as well as rules to introduce and eliminate intersections (rules $(\wedge I)$ and $(\wedge E)$, respectively; only one rule is given for $(\wedge E)$ since we are working modulo commutativity of intersection).*

We express the fact that it is possible to derive $B \vdash^{\wedge}_{\mathcal{A}x} t{:}\sigma$ by simply stating $B \vdash^{\wedge}_{\mathcal{A}x} t{:}\sigma$.

Definition 8.6. *A rewrite rule $l \to r$ is typeable with respect to a set $\mathcal{A}x$ of axiom statements for $\mathcal{F}_\lambda$ if there is an algebraic basis A_l and an algebraic type σ such that $A_l \vdash^{\wedge}_{\mathcal{A}x} l\colon \sigma$ and $A_l \vdash^{\wedge}_{\mathcal{A}x} r\colon \sigma$.*

We show next that the generic rewrite rules of the metamodel (see Table 1) are typeable.

Proposition 8.7. *The generic rewrite rules (both in the first-order and higher-order versions) are typeable assuming:*

- *The set Sorts has sorts $\mathcal{P}, \mathcal{A}, \mathcal{R}, \mathcal{S}$ for principals, actions, resources, and sites, respectively, together with sorts $\mathcal{C}$ for categories, $\mathcal{A}uth$ for authorisation results and $\mathcal{B}ool$ for Booleans. We denote by $[\mathcal{C}]$ the sort of lists of categories, and by $[\mathcal{P}air]$ the sort of lists of pairs of (action,resource).*
- *$\mathcal{A}x$ includes the following declarations (note that we do not consider polymorphic function symbols in this paper, so we distinguish the constructors for lists of categories and lists of pairs):*

$\mathsf{deny}, \mathsf{grant}, \mathsf{undeterminate}:$	$\mathcal{A}uth$
$\mathsf{par}:$	$\mathcal{P} \to \mathcal{A} \to \mathcal{R} \to \mathcal{A}uth$
$\mathsf{arca}^*, \mathsf{barca}^*:$	$[\mathcal{C}] \to [\mathcal{P}air]$
$\mathsf{nil}_{\mathcal{C}}:$	$[\mathcal{C}]$
$\mathsf{cons}_{\mathcal{C}}:$	$\mathcal{C} \to [\mathcal{C}] \to [\mathcal{C}]$
$\mathsf{cons}_{\mathcal{P}air}:$	$\mathcal{P}air \to [\mathcal{P}air] \to [\mathcal{P}air]$
$\mathsf{nil}_{\mathcal{P}air}:$	$[\mathcal{P}air]$
$\mathsf{append}:$	$[\mathcal{P}air] \to [\mathcal{P}air] \to [\mathcal{P}air]$
$\mathsf{above}, \mathsf{below}:$	$[\mathcal{C}] \to [\mathcal{C}]$
$\in:$	$\mathcal{P}air \to [\mathcal{P}air] \to \mathcal{B}ool$
$\mathsf{ifthenelse}:$	$\mathcal{B}ool \to \mathcal{A}uth \to \mathcal{A}uth \to \mathcal{A}uth$
$\mathsf{pca}:$	$\mathcal{P} \to [\mathcal{C}]$
$\mathsf{arca}, \mathsf{barca}:$	$\mathcal{C} \to [\mathcal{P}air]$
$\mathsf{authorised}:$	$\mathcal{P} \to \mathcal{A} \to \mathcal{R} \to \overline{\mathcal{S}} \to \mathcal{A}uth$

Proof 8.8. *We need to show that for each rewrite rule, there exists an algebraic basis such that both the left and right-hand sides of the rule are typeable with the same algebraic type. We show the case of* par*:*

First, using rule $(\mathcal{F})$*,*

$$p\colon \mathcal{P}, a\colon \mathcal{A}, r\colon \mathcal{R} \vdash^{\wedge}_{\mathcal{A}x} \mathsf{par}(p, a, r)\colon \mathcal{A}uth$$

since we have $\mathsf{par}: \mathcal{P} \to \mathcal{A} \to \mathcal{R} \to \mathcal{A}uth \in \mathcal{A}x$*. It remains to prove that the right-hand side of the rewrite rule defining* par *also has type* $\mathcal{A}uth$ *in this basis. Since* $\mathsf{ifthenelse}: \mathcal{B}ool \to \mathcal{A}uth \to \mathcal{A}uth \to \mathcal{A}uth \in \mathcal{A}x$*, it boils down to proving*

$p\colon \mathcal{P}, a\colon \mathcal{A}, r\colon \mathcal{R} \vdash^{\wedge}_{\mathcal{A}x} (a, r) \in \mathsf{arca}^*(\mathsf{below}(\mathsf{pca}(p)))\colon \mathcal{B}ool$

and

$p\colon \mathcal{P}, a\colon \mathcal{A}, r\colon \mathcal{R} \vdash^{\wedge}_{\mathcal{A}x} (a, r) \in \mathsf{barca}^*(\mathsf{above}(\mathsf{pca}(p)))\colon \mathcal{B}ool$*.*

We derive the type for the first one as follows (the other is similar). Since the basis includes $a\colon \mathcal{A}, r\colon \mathcal{R}$*, the first argument of* $\in$ *is of type* $\mathcal{P}air$ *as expected; we show that the second argument is of type* $[\mathcal{P}air]$ *and we are done (using again rule* $\mathcal{F}$*, since* $\in\colon \mathcal{P}air \to [\mathcal{P}air] \to \mathcal{B}ool \in \mathcal{A}x$*):*

$$\dfrac{\dfrac{\dfrac{p\colon \mathcal{P}, a\colon \mathcal{A}, r\colon \mathcal{R} \vdash^{\wedge}_{\mathcal{A}x} p\colon \mathcal{P}}{p\colon \mathcal{P}, a\colon \mathcal{A}, r\colon \mathcal{R} \vdash^{\wedge}_{\mathcal{A}x} \mathsf{pca}(p)\colon [\mathcal{C}]}}{p\colon \mathcal{P}, a\colon \mathcal{A}, r\colon \mathcal{R} \vdash^{\wedge}_{\mathcal{A}x} \mathsf{below}(\mathsf{pca}(p))\colon [\mathcal{C}]}}{\mathsf{arca}^*(\mathsf{below}(\mathsf{pca}(p)))\colon [\mathcal{P}air]}$$

In [6], it is shown that the rewriting relation generated by typeable rules is terminating over typeable terms if the rewrite rules satisfy a general scheme of recursion. This is a restricted form of recursion, inspired by the primitive recursion scheme, which allows us to define recursive functions using rewrite rules where the recursive calls (in the right-hand side) are made on subterms of the arguments used in the left-hand side of the rule. We recall the general recursive scheme below. The notation $\overline{l}[\overline{x}]$ generalises the subterm notation $t[u]_p$ (see Section 2.1): it represents a sequence of algebraic terms of the form $l_i[x_{i1}]_{p_{i1}} \dots [x_{in}]_{p_{in}}$, where $x_{i1}, \dots, x_{in}$ are the variables occurring in l_i.

Definition 8.9 (General scheme of recursion). *(i) A rewrite rule* $l \to r$ satisfies the general schema *if it is of the form*

$$\mathsf{f}(\bar{l}[\overline{x}], \overline{y}) \to v[\mathsf{f}(\overline{q_1}[\overline{x}], \overline{y}), \ldots, \mathsf{f}(\overline{q_m}[\overline{x}], \overline{y})]$$

where $\overline{x}$ *and* $\overline{y}$ *are sequences of variables and*

1. $x \in \overline{y}$ *if* x *is a variable of arrow type,*
2. f *does not appear in the sequences of terms* $\bar{l}$, $\overline{q_1}, \ldots, \overline{q_m}$, *and its occurrences in* v *are only the ones explicitly indicated,*
3. $\bar{l}, \overline{q_1}, \ldots, \overline{q_m}$ *are terms with variables only in* $\overline{x}$,
4. $\forall i \in \{1..m\}$, $\overline{q_i} \lhd_{mul} \bar{l}$ *where* $\lhd$ *denotes strict subterm ordering and* $\lhd_{mul}$ *its multiset extension, defined as usual.*

(ii) A TRS_λ satisfies the general schema *if each rewrite rule satisfies the general schema and there are no mutually recursive functions.*

Proposition 8.10. *The generic rewrite rules (both in the first-order and higher-order versions) satisfy the general scheme of recursion.*

Proof 8.11. *By inspection of the rewrite rules defining* par, arca^*, barca^*, $\mathsf{authorised}$ *in Table 1. The auxiliary functions (*append, $\in$, *etc.) used in the generic rules are also defined by rules that satisfy the general scheme of recursion.*

Theorem 8.12. *If the specific rules defining the specific functions* pca, arca, barca, above, below, *and the first-order or higher-order versions of the Fauth rule in a policy are typeable and, when combined with the generic rules, yield a* TRS_λ *that satisfies the general scheme of recursion, then the reduction relation associated to the policy is terminating on typeable terms.*

Proof 8.13. *The assumptions together with Proposition 8.7 and 8.10, imply that the full set of rewrite rules is typeable and satisfies the general scheme of recursion, thus termination follows from [6].*

As a consequence, we obtain a sufficient condition for totality of the policy defined by the TRS_λ.

Property 8.14 (Totality). *Assume the* TRS_λ *defining a policy as an instance of the metamodel satisfies the assumptions of Theorem 8.12. If the specific functions of this instance as well as the combination operators (defined via* fauth, *or using a* λ*-abstraction* f *in* hoAuth*) are well defined (i.e., their evaluation produces a result provided the arguments are valid; for instance, the evaluation of a ground term* $\mathsf{pca}(\mathsf{p})$ *results in a list of categories for any principal* p, $\mathsf{arca}(\mathsf{c})$ *and* $\mathsf{barca}(\mathsf{c})$ *produce lists of pairs* (a, r), fauth *produces a result in* $\mathcal{A}uth$, *the* λ*-abstraction* f *produces an authorisation answer when applied to given arguments), then the evaluation of a ground term of the form* $\mathsf{authorised}(\mathsf{p}, \mathsf{a}, \mathsf{r}, \mathsf{s}_1, \ldots, \mathsf{s}_n)$ *where* $\mathsf{p} \in \mathcal{P}, \mathsf{a} \in A, \mathsf{r} \in R, \mathsf{s}_1, \ldots, \mathsf{s}_n \in \mathcal{S}$ *results in a normal form in* $\mathcal{A}uth$.

Correctness and completeness of the policy are in general easy to check once the totality and consistency properties have been proved for the specific instance of the metamodel under consideration.

It is important to note that the proofs of the properties above do not have to be generated by a security administrator; rather, the properties demonstrate that conditions on specific functions in a TRS_λ ensure the consistency and totality of the corresponding policy. Tools such as CiME [29] or the Maude Sufficient Completeness Checker [42] could be used to help checking properties of the first-order rewrite rules. As for higher-order policies verification, type systems similar to the one discussed in this section, but without intersection types, are available in most modern functional programming languages. A prototype for automated analysis of first-order access control policies in the (distributed) meta-model $\mathcal{M}$ is presented in [22]. This work describes a Java application with a user-friendly interface for helping the policy designer in the specification of the policy. Automatic checks, e.g. for consistency and totality, are then performed by calling in a transparent way the CiME rewrite tool.

9. Related Work

Term rewriting has been used to model a variety of problems in security, from the analysis of security protocols (see, for example, [12, 37, 60]) to the definition of policies for controlling information leakage [36]. On access control specifically, Koch et al. [48] use graph transformation rules to formalise RBAC, and more recently, [10, 59, 21] use term rewrite rules to model particular access control models and to express access control policies. Our work addresses similar issues to [10, 48, 59, 21], but is based on a notion of a metamodel of access control from which various models can be derived, instead of formalising a specific model such as RBAC or DEBAC. A related approach to policy composition via rewriting is described in [35], where reduction strategies are used to combine the rewrite rules specifying individual policies. In our metamodel, we specify policy integration giving a rewriting semantics to the composition operators. Moreover, we can deal with incomplete policy specifications, where some components are not known a priori, using the higher-order functional operators. Some of our operators to combine policies are inspired by those of Bonatti et al. [23]; but, unlike [23], our policy model supports both positive and negative authorisations stating permissions as well as denials, and also partially defined policies where the outcome of an access request can be *grant*, *deny*, or *undeterminate*. This allows us to define more general compositions, in particular when the individual policies contain contradictory information, e.g. like in XACML policy combining algorithms.

Many works exist about policy specification using logic programming (see e.g. [15, 11, 34, 44]). The term rewriting approach has similar attractions to the (C)LP approaches. Moreover, in contrast to these approaches, our proposal does not require that the syntactic restriction to access policies that are *locally stratified* [5] be adopted (to ensure the existence of a categorical semantics and thus unambiguous access control policies). Also, general policy-composition operators can be concisely specified using higher-order rewrite rules (see for instance Example 7.1).

Several proposals for general models and languages for access control have already been described in the literature. Abadi et al [1] ABLP logic also 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 category-based approach is based on common aspects of

access control, from which core functions are identified, and emphasises the use of term rewriting techniques to derive properties of the policies. Li et al.'s *RT* family of role-trust models [50] provides a general framework for defining access control policies, which can be specialised for defining specific policy requirements (in terms of credentials). The category-based metamodel, however, can be instantiated to include concepts like times, events, actions and histories that may be used to specify principal-category assignments, but which are not included as elements of *RT*.

More recently, in [9] a general access control model and a logic-based specification language are presented. Even if the model incorporates the notion of site like our distributed metamodel, the emphasis is on the expressivity of the model and not on (distributed) access request evaluation. Moreover, the issue of combinations of different policies is not addressed. In [41] the authors propose a formal generic definition of access control models, but their focus is on comparison of well-known models rather than on combining those models into a uniform framework where composition of policies may be performed, based on a sound semantics.

On the distributed aspect, decentralised systems are studied in [13], where the authors propose the constraint logic programming language SecPal. Inspired by this work, the DKAL authorisation language was proposed, based on existential fixed-point logic. In our approach, we focus on the definition of a general metamodel suitable for distributed systems rather than on the design of a specification language, but we give an operational semantics for the metamodel which can be instantiated further to derive specific access control models and policies.

In [49, 58], the authors deal with group-centric secure information sharing, where users and information come together in groups to facilitate sharing. This work is related to ours, since it specifies how to secure information access in a distributed scenario. However in [49] the emphasis is on specifying temporal and coupling interactions between users and their group(s). The authors consider for instance membership relations, since users may join, leave and re-join one or more groups. These aspects are not developed specifically in this paper, but following the idea of distributed-DEBAC in Example 5.1, the event history could be used to specify a similar requirements, based on the history of accesses (seen as events).

The notion of Federated Systems can also be related to our work. In [46, 32] local systems evolving independently cooperate in a distributed architecture called federation. In our model, we can simulate this behaviour specifying local sets of registered users and defining rules for treating requests from external users at the federation level (associated to a remote site).

10. Conclusions and Further Work

We have given a rewrite-based specification of a metamodel of access control that is based on common, core concepts of access control models. The rewriting approach can be used to give a formal semantics to policies in the case of both centralised and distributed computer systems. Rewrite rules provide a declarative specification of access control policies that facilitates the task of proving properties of policies. Also, term rewriting rules provide an executable specification of the access control policy. A first-order rewriting system can be transformed into a MAUDE program simply by adding type declarations for the function symbols and variables used and by making minor

syntactical changes [28, 59]. Policies defined by higher-order rewriting systems can be directly implemented in a functional language [17].

In future work, we will investigate the design of languages for policy specification and the practical implementation of category-based policies. It would be interesting to see how aspect-oriented techniques apply to this case (work in this direction has been reported in [31], where the authors discuss weaving rewrite-based policies into Java programs). We also aim to develop a tool, by improving the prototype proposed in [22], for helping users and policy administrators to state and prove automatically properties of policy specifications. On a more theoretical level, it would be interesting to study an abstract, modular set of sufficient conditions for totality and consistency of policies defined by composition of individual policies.

Acknowledgements:. We are grateful to the anonymous referees for many useful suggestions and to Anatoli Degtyarev for his helpful comments. This work was partially funded by EOARD.

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, L. Compagna, and Y. Lieler. Automatic compilation of protocol insecurity problems into logic programming. In *Proc. of JELIA'04*, volume 3229 of *Lecture Notes in Computer Science*, 2004.

[3] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[4] S. van Bakel and M. Fernández. Normalization results for typeable rewrite systems. *Information and Computation*, 133(2):73–116, 1997.

[5] C. Baral and M. Gelfond. Logic programming and knowledge representation. *JLP*, 19/20:73–148, 1994.

[6] F. Barbanera and M. Fernández. Intersection type assignment systems with higher-order algebraic rewriting. *Theoretical Computer Science*, 170:173–207, 1996.

[7] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, revised edition, 1984.

[8] S. Barker. The next 700 access control models or a unifying meta-model? In *SACMAT 2009, 14th ACM Symposium on Access Control Models and Technologies, Stresa, Italy, June 3-5, 2009, Proceedings*, pages 187–196. ACM Press, 2009.

[9] S. Barker, G. Boella, D. M. Gabbay, and V. Genovese. A meta-model of access control in a fibred security language. *Studia Logica*, 92(3):437–477, 2009.

[10] 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.

[11] 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.

[12] G. Barthe, G. Dufay, M. Huisman, and S. Melo de Sousa. Jakarta: a toolset to reason about the JavaCard platform. In *Proceedings of e-SMART'01*, number 2140 in Lecture Notes in Computer Science. Springer-Verlag, 2002.

[13] M. Y. Becker, C. Fournet, and A. D. Gordon. Design and semantics of a decentralized authorization language. In *Proc. of 20th IEEE Computer Security Foundations Symposium, CSF 2007*, pages 3–15. IEEE Computer Society, 2007.

[14] D. E. Bell and L. J. LaPadula. Secure computer system: Unified exposition and multics interpretation. *MITRE-2997*, 1976.

[15] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. In *SACMAT 2001, 6th ACM Symposium on Access Control Models and Technologies, Litton-TASC, Chantilly, Virginia, USA, May 3-4, 2001, Proceedings*, pages 41–52. ACM, 2001.

[16] C. Bertolissi and M. Fernández. A rewriting framework for the composition of access control policies. In *Proceedings of the 10th ACM-SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'08), Valencia, 2008.* ACM Press, 2008.

[17] C. Bertolissi and M. Fernández. Time and location based services with access control. In *NTMS 2008, 2nd International Conference on New Technologies, Mobility and Security, 2008, Tangier, Morocco*, pages 1–6. IEEE, 2008.

[18] 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.

[19] C. Bertolissi and M. Fernández. Category-based authorisation models: operational semantics and expressive power. In *Proc. of Int. Symposium on Engineering Secure Software and Systems, ESSOS 2010, Pisa*, number 5965 in Lecture Notes in Computer Science, pages 140–156. Springer, 2010.

[20] C. Bertolissi and M. Fernández. Rewrite specifications of access control policies in distributed environments. In *Proc. of STM 2010: 6th Workshop on Security and Trust Management, Athens, Greece, 2010*, number 6710 in Lecture Notes in Computer Science. Springer, 2011.

[21] C. Bertolissi, M. Fernández, and S. Barker. Dynamic event-based access control as term rewriting. In *Data and Applications Security XXI. Proceedings of DBSEC 2007*, number 4602 in Lecture Notes in Computer Science. Springer-Verlag, 2007.

[22] C. Bertolissi and W. Uttha. Automated analysis of rule-based access control policies. In *Proc. of the 6th workshop on Programming Languages meet Program Verification (PLPV'13) affiliated with POPL'13, Rome, Italy, January 22, 2013.* ACM, 2013.

[23] P. Bonatti, S. de Capitani di Vimercati, and P. Samarati. A modular approach to composing access control policies. In *CCS'00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 164–173, New York, NY, USA, 2000. ACM Press.

[24] P. A. Bonatti and P. Samarati. Logics for authorization and security. In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases*, pages 277–323. Springer, 2003.

[25] S. M. Chandran and J. B. D. Joshi. LoT-RBAC: A location and time-based rbac model. In *Proc. of WISE 2005, 6th International Conference on Web Information Systems Engineering, NY, USA, 2005*, number 3806 in Lecture Notes in Computer Science, pages 361–375. Springer, 2005.

[26] H. Cirstea and C. Kirchner. The Rewriting Calculus - Part I. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9:363–399, May 2001. Also available as Technical Report A01-R-203, LORIA, Nancy (France).

[27] H. Cirstea and C. Kirchner. The Rewriting Calculus - Part II. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9:401–434, May 2001. Also available as Technical Report A01-R-204, LORIA, Nancy (France).

[28] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The Maude 2.0 system. In *Rewriting Techniques and Applications (RTA 2003)*, number 2706 in Lecture Notes in Computer Science, pages 76–87. Springer-Verlag, 2003.

[29] É. Contejean, A. Paskevich, X. Urbain, P. Courtieu, O. Pons, and J. Forest. A3pat, an approach for certified automated termination proofs. In *Proc. of the 2010 ACM SIGPLAN workshop on Partial evaluation and program manipulation*, PEPM '10, pages 63–72, New York, NY, USA, 2010. ACM.

[30] M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.

[31] A. Santana de Oliveira, E. Ke Wang, C. Kirchner, and H. Kirchner. Weaving rewrite-based access control policies. In *Proceedings of the 2007 ACM workshop on Formal methods in security engineering, FMSE 2007, Fairfax, VA, USA, November 2, 2007*, pages 71–80. ACM, 2007.

[32] S. De Capitani di Vimercati and P. Samarati. Authorization specification and enforcement in federated database systems. *J. Comput. Secur.*, 5:155–188, March 1997.

[33] D. Dougherty, P. Lescanne, L. Liquori, and F. Lang. Addressed term rewriting systems: Syntax, semantics and pragmatics. In *Proceedings of TERMGRAPH 2004*, Electronic Notes in Theoretical Computer Science. Elsevier, 2005.

[34] D. J. Dougherty, K. Fisler, and S. Krishnamurthi. Specifying and reasoning about dynamic access-control policies. In Ulrich Furbach and Natarajan Shankar, editors, *Proc. of IJCAR'06*, volume 4130 of *Lecture Notes in Computer Science*, pages 632–646. Springer, 2006.

[35] D. J. Dougherty, C. Kirchner, H. Kirchner, and A. Santana de Oliveira. Modular access control via strategic rewriting. In *Proceedings of 12th European Symposium On Research In Computer Security, ESORICS*, pages 578–593, 2007.

[36] R. Echahed and F. Prost. Security policy in a declarative style. In *Proc. 7th ACM-SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'05)*. ACM Press, 2005.

[37] S. Escobar, C. Meadows, and J. Meseguer. A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties. *Theor. Comput. Sci.*, 367:162–202, November 2006.

[38] M. Fernández. Narrowing based procedures for equational disunification. *Applicable Algebra in Engineering, Communication and Computing*, 3:1–26, 1992.

[39] M. Fernández and J.-P. Jouannaud. Modular termination of term rewriting systems revisited. In *Recent Trends in Data Type Specification. Proc. 10th. Workshop on Specification of Abstract Data Types (ADT'94)*, number 906 in Lecture Notes in Computer Science, Santa Margherita, Italy, 1995.

[40] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J-P. Jouannaud. Introducing OBJ. In G. Malcolm, editor, *Software Engineering with OBJ: algebraic specification in action*. Kluwer, 2000.

[41] 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.

[42] J. Hendrix, M. Clavel, and J. Meseguer. A sufficient completeness reasoning tool for partial specifications. In *Proc. of 16th Int. Conference on Term Rewriting and Applications, RTA 2005, Nara, Japan, 2005*, number 3467 in Lecture Notes in Computer Science, pages 165–174. Springer, 2005.

[43] R. Jagadeesan and V. Saraswat. Timed Constraint Programming: A Declarative Approach to Usage Control. In *Proc. 7th ACM-SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'05)*. ACM Press, 2005.

[44] S. Jajodia, P. Samarati, M. Sapino, and V.S. Subrahmaninan. Flexible support for multiple access control policies. *ACM TODS*, 26(2):214–260, 2001.

[45] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. *SIGMOD Rec.*, 26(2):474–485, 1997.

[46] D. Jonscher and K. R. Dittrich. An approach for building secure database federations. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 24–35, San Francisco, CA, USA, 1994.

[47] J.-W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems, introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.

[48] M. Koch, L. Mancini, and F. Parisi-Presicce. A graph based formalism for RBAC. In *Proc. of SACMAT 2004, 9th ACM Symposium on Access Control Models and Technologies, New York, USA, 2004*, pages 129–187, 2004.

[49] R. Krishnan, R. S. Sandhu, J. Niu, and W. H. Winsborough. Foundations for group-centric secure information sharing models. In *SACMAT 2009, 14th ACM Symposium on Access Control Models and Technologies, Stresa, Italy, June 3-5, 2009, Proceedings*, pages 115–124. ACM, 2009.

[50] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *IEEE Symposium on Security and Privacy*, pages 114–130, 2002.

[51] N. Li, Q. Wang, W. H. Qardaji, E. Bertino, P. Rao, J. Lobo, and D. Lin. Access control policy combining: theory meets practice. In *SACMAT 2009, 14th ACM Symposium on Access Control Models and Technologies, Stresa, Italy, June 3-5, 2009, Proceedings*, pages 135–144. ACM, 2009.

[52] A. Martelli and U. Montanari. An efficient unification algorithm. *Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

[53] John McLean. The algebra of security. In *Proceedings of the 1988 IEEE conference on Security and privacy*, SP'88, pages 2–7, Washington, DC, USA, 1988. IEEE Computer Society.

[54] M.H.A. Newman. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2):223–243, 1942.

[55] OASIS. eXtensible Access Control Markup language (XACML), 2003. http://www.oasis-open.org/xacml/docs/.

[56] Department of Defense. Trusted computer system evaluation criteria, 1983. DoD 5200.28-STD.

[57] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[58] R. S. Sandhu, R. Krishnan, J. Niu, and W. H. Winsborough. Group-centric models for secure and agile information sharing. In *Proc. of 5th Int. Conference MMM-ACNS 2010, St. Petersburg, Russia, 2010*, volume 6258 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2010.

[59] A. Santana de Oliveira. *Réécriture et Modularité pour les Politiques de Sécurité*. PhD thesis, Université Henri Poincaré, Nancy, France, 2008.

[60] L. Viganò. Automated security protocol analysis with the AVISPA tool. In *Proc. of MFPS'05*, volume 155 of *ENTCS*, pages 61–86. Elsevier, 2005.

[61] D. Wijesekera and S. Jajodia. A propositional policy algebra for access control. *ACM Trans. Inf. Syst. Secur.*, 6(2):286–325, 2003.

[62] N. Yoshida. Channel dependent types for higher-order mobile processes. In *Proc. of the 31st ACM*

SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, pages 147–160. ACM, 2004.