Time and Location Based Services with Access Control*

Clara Bertolissi[1]
Laboratoire d'Informatique fondamentale (LIF)
and Université de Provence
Marseille, France
Email: Clara.Bertolissi@kcl.ac.uk

Maribel Fernández
King's College London
Dept. of Computer Science
London WC2R 2LS, U.K.
Email: Maribel.Fernandez@kcl.ac.uk

***Abstract*—We propose an access control model that extends RBAC (Role-Based Access Control) to take time and location into account, and use term rewriting systems to specify access control policies in this model. We discuss implementation techniques for rewrite-based policy specifications, and the integration of these policies in web applications. The declarative nature of the model facilitates the analysis of policies and the evaluation of access requests: we present two case-studies.**

I. Introduction

The recent growth of digital communication has increased the demand of security for protecting resources and preserving the integrity and confidentiality of data. Thus, considerable interest has been centred on the area of access control models in recent years. Access control is concerned with deciding which actions a subject can execute on the objects of a given system. One of the most popular choices, for use with centralised systems, is the Role-Based Access Control (RBAC) model [19]. In RBAC, users are assigned to roles by a security administrator; roles usually map to job titles in an organisation, and as such, this model is well-suited for relatively static environments.

High mobility of users and services in the emerging mobile applications entails the need for access control models that take the location of the user and the time of the request into account in order to decide whether to grant or deny an access request. Several extensions to RBAC have been proposed to incorporate spatio-temporal information in the model. One of the first time-based RBAC models was proposed in [5], and later generalised in the GTRBAC model [17]. In these systems, the roles are enabled by time constraints. An example of location-based model is GEO-RBAC, introduced in [13]. The GEO-RBAC model allows a user to activate a role from a particular location, and the role and its permissions are predefined for that location. Other models considering both location and time constraints have been proposed, see for example [12], [11], [22], [9].

In this paper we describe a location- and time-based RBAC model, TLRBAC, that we specify using term rewriting. Term rewriting systems are usually defined by specifying a set of terms, and a set of rewrite rules that are used to "reduce" terms. Term rewriting techniques have been successfully applied to many domains, and have had deep influence in the development of computation models, programming and specification languages, theorem provers and proof assistants. Recently, rewriting techniques have been fruitfully exploited in the context of security [3], [15], [20], [2], [7], in particular, to ensure that access control policies satisfy essential properties (such as consistency and totality, see for instance [2], [20], [6], [14]). Another important reason to specify access control policies using rewrite-based frameworks is that rewriting tools and languages (such as MAUDE [10], TOM [23], and ML [18]), can be used to test, compare and experiment with evaluation strategies, to automate equational reasoning, and for the rapid prototyping of access control policies. However, none of the previously mentioned rewrite-based access control models deals with time and location constraints.

After defining the TLRBAC model as a rewrite system, we discuss implementation techniques, and apply them to two case studies. The first one describes a time and location-based access control policy for a hospital, which has been implemented using the programming language Maude. The second case study is a museum, where RBAC is used to control the access to the expositions, and a location-based RBAC model is used to specify the access policy for virtual visitors. This example has been implemented using the functional programming language CAML [8].

The rest of this paper is organised as follows. In Section II, we give some details on term rewriting and access control models, to help to make the paper self-contained. In Section III we specify the TLRBAC model as a term rewriting system. Section IV discusses implementation techniques; we describe two case studies in Section V and conclude the paper in Section VI.

II. Preliminaries

We recall the main notions of rewriting as well as the RBAC model and its extensions. We refer the reader to [1], [2], [13] for additional information.

* Research partially funded by the EU project *Implementing access control mechanisms using rewriting techniques*, Marie Curie Intra European Fellowships Programme.

[1] Research visiting fellow at the Department of Computer Science, King's College London.

A. Rewriting

A *signature* $\mathcal{F}$ is a finite set of *function symbols* together with their (fixed) arity. $\mathcal{X}$ denotes a denumerable set of *variables* $X_1, X_2, \ldots$, and $T(\mathcal{F}, \mathcal{X})$ denotes the set of *terms* built up from $\mathcal{F}$ and $\mathcal{X}$. Terms are identified with finite labeled trees. *Positions* are strings of positive integers. The *subterm* of t at position p is denoted by $t|_p$ and the result of replacing $t|_p$ with u at position p in t is denoted by $t[u]_p$.

$\mathcal{V}(t)$ denotes the set of variables occurring in t. Substitutions are written $\{X_1/t_1, \ldots, X_n/t_n\}$ where t_i is assumed to be different from the variable X_i. We use Greek letters for substitutions and postfix notation for their application.

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

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

Example 1: Consider a signature for lists of natural numbers, with function symbols z (with arity 0) and s (with arity 1, denoting the successor function) to build numbers;

Nil (with arity 0, to denote an empty list), Cons (with arity 2, to construct non-empty lists), and Length (with arity 1, to compute the length of a list). The list containing the numbers 0 and 1 is written: Cons(z, Cons(s(z), Nil)), or simply [z, s(z)] for short. We can specify the function Length by the recursive rewrite rules:

$$\begin{array}{rcl} \mathsf{Length}(\mathsf{Nil}) & \to & \mathsf{z} \\ \mathsf{Length}(\mathsf{Cons}(X, L)) & \to & \mathsf{s}(\mathsf{Length}(L)) \end{array}$$

Then we have a reduction sequence:

$$\mathsf{Length}(\mathsf{Cons}(\mathsf{z}, \mathsf{Cons}(\mathsf{s}(\mathsf{z}), \mathsf{Nil}))) \to^* \mathsf{s}(\mathsf{s}(\mathsf{z}))$$

B. RBAC and extensions

RBAC policies are specified with respect to a domain of discourse that includes the sets $\mathcal{U}$ of users, $\mathcal{O}$ of objects (or resources), and $\mathcal{A}$ of access privileges, together with a (finite) set $\mathcal{R}$ of *roles*. A user u may exercise an access privilege a on a resource o if and only if u is assigned to a role r to which the access privilege a on o is also assigned.

The capability of assigning users to roles, and permissions (i.e., access privilege assignments on objects) to roles, are the basic requirements of all RBAC models. We have two types of relations involving roles:

- user-role assignments: we write $ura(u, r)$ if and only if user $u \in \mathcal{U}$ is assigned to role $r \in \mathcal{R}$;
- permission-role assignments: we write $pra(a, o, r)$ if and only if the access privilege $a \in \mathcal{A}$ on object $o \in \mathcal{O}$ is assigned to the role $r \in \mathcal{R}$.

The model can be extended to include the notion of a *role hierarchy*, defined as a (partially) ordered (and finite) set of roles. The ordering relation is denoted $senior\_to(r_i, r_j)$, and means that the role $r_i \in \mathcal{R}$ is a senior role (or more powerful role) than role $r_j \in \mathcal{R}$. Role hierarchies are important for specifying implicitly the inheritance of access privileges on resources.

Example 2: Suppose that the users u_1 and u_2 are assigned to the roles r_2 and r_1 respectively, and that write (w) permission on object o_1 is assigned to r_1 and read (r) permission on o_1 is assigned to r_2. Moreover, suppose that r_1 is directly senior to r_2 in an role hierarchy. Then, using the notation introduced above, this policy is represented by the relations:
$ura(u_1, r_2), ura(u_2, r_1), pra(w, o_1, r_1), pra(r, o_1, r_2),$
$senior\_to(r_1, r_2)$.

User-role and permission-role assignments are related via the notion of an *authorisation*. An authorisation is a triple (u, a, o) that expresses that the user u has the a access privilege on the object o.

For example, consider a user u that may exercise the a access privilege on object o if u is assigned to the role r_1, r_1 is senior to a role r_2 in a role hierarchy, and r_2 has been assigned the a access privilege on o. The corresponding set of authorisations $\mathcal{AUTH}$ may be expressed thus:
$(u, a, o) \in \mathcal{AUTH} \Leftrightarrow$
$\exists r_1, r_2 . ura(u, r_1) \ \wedge \ senior\_to(r_1, r_2) \ \wedge \ pra(a, o, r_2)$

Example 3: By inspection of the user-role assignments, permission-role assignments, and the role seniority relationships that are specified in Example 2, it follows that the set of authorisations that are included in $\mathcal{AUTH}$ is:

$$\{(u_2, w, o_1), (u_2, r, o_1), (u_1, r, o_1)\}.$$

The RBAC model presented so far can be extended to take the *location* of the user, and the *time* of the request, into account for access request evaluation. The extension of the RBAC model with time and location constraints will be called TLRBAC. In the TLRBAC model:

- A physical location $l \in \mathcal{L}$ is a collection of points in a three dimensional geometric space. A logical location is an abstract notion that characterises possibly many physical locations, like patient's home or city.
- Time is seen as a sequence of discrete time points t on the time line. An interval int is a set of time instances, for example from 9am to 5pm.
- Locations are assigned to users at any time point.
- Permissions are assigned to roles during a time interval and according to the user location.
- A user u may exercise an access privilege a on a resource o if and only if u can activate a role r to which the access privilege a on o is also assigned. To activate a role r, the user must be in a location and during a time interval where the activation can take place (as specified below).

We have now the following relations involving roles.

- user-role assignments: we write $uralt(u, r)$ if and only if user $u \in \mathcal{U}$ is assigned to role $r \in \mathcal{R}$.
- permission-role assignments: we write $pralt(p, r, l, t)$ if and only if the access privilege $p \in \mathcal{P}$ is assigned to the role $r \in \mathcal{R}$ at location l and time t.

- role enabling: we write $renab(r, l, t)$ if the role r is enabled at time t and location l.

Location and time information can be used to define where a role is enabled. The set of authorisations $\mathcal{AUTH}$ is composed by tuples (u, a, o, l, t) that express the fact that the user u has the a access privilege on the object o at time t and location l.

Example 4: Suppose that the users u_1 and u_2 are at location l at time t, and that at that time and location they are assigned to the roles r_2 and r_1 respectively, with write (w) permission on object o_1 assigned to r_1 and read (r) permission on o_1 assigned to r_2. Suppose that role r_1 is enabled at that location and time, but role r_2 is not enabled at time t in location l. Then, using the notation introduced above, this information is represented by the relations:
$uralt(u_1, r_2), uralt(u_2, r_1)$,
$pralt(w, o_1, r_1, l, t), pralt(r, o_1, r_2, l, t)$,
$renab(r_1, l, t)$.
Then it follows that (u_2, w, o_1, l, t) is included in $\mathcal{AUTH}$.

III. The TLRBAC model as a rewrite system

In this section we define the TLRBAC model as a term rewriting system.

We begin by defining the sets of constants in the signature that are used in the formulation of TLRBAC policies.

- A countable set $\mathcal{O}$ of *objects*, written $o_1, o_2, \ldots$
- A countable set $\mathcal{A}$ of *access privileges* $a_1, a_2, \ldots$
- A countable set $\mathcal{U}$ of *user identifiers*, written $u_1, u_2, \ldots$
- A countable set $\mathcal{L}$ of *locations* $l_1, l_2, \ldots$
- A countable set $\mathcal{T}$ of *time points* $t_1, t_2, \ldots$.

In our TLRBAC model access to resources is defined in the following way:

> *A user $u \in \mathcal{U}$ is permitted to perform an action $a \in \mathcal{A}$ on an object $o \in \mathcal{O}$ if and only if u at time $t \in \mathcal{T}$, being in location $l \in \mathcal{L}$, can activate a role $r \in \mathcal{R}$ to which a access on o has been assigned at time t and location l.*

We will use the function roles: $\mathcal{U} \rightarrow List(\mathcal{R})$ to represent the assignment of roles to users (note that a user may be assigned to several roles). For example, the function roles can be specified for any user u by the rewrite rules:

$$\mathsf{roles}(u) \rightarrow [r_1, \ldots, r_i]$$

To represent the assignment of privileges to roles we use a function Priv from roles to lists of tuples Priv: $\mathcal{R} \rightarrow List(\mathcal{A} \times \mathcal{O} \times \mathcal{L} \times \mathcal{T})$. For example, the function Priv for a role r can be specified by the following rewrite rule:

$$\mathsf{Priv}(r) \rightarrow [(a_1, o_1, l_1, t_1), \ldots, (a_i, o_i, l_i, t_i)]$$

Finally we use a function enable: $\mathcal{R} \rightarrow List(\mathcal{L} \times \mathcal{T})$ to define the times and locations at which the roles can be possibly activated.

$$\mathsf{enable}(r) \rightarrow [(l_1, t_1), \ldots, (l_i, t_i)]$$

The definition of the funtion enable can be easily modified to consider intervals of time instead of simple time instants (see for example the hospital case study below).

Example 5: The user-role and permission-role assignments described in Example 4 may be specified by the rules:

$$\begin{array}{ll} \mathsf{roles}(u_1) \rightarrow [r_2] & \mathsf{roles}(u_2) \rightarrow [r_1] \\ \mathsf{Priv}(r_1) \rightarrow [(w, o_1, l, t)] & \mathsf{Priv}(r_2) \rightarrow [(r, o_1, l, t)] \\ \mathsf{enable}(r_1) \rightarrow [(l, t)] & \end{array}$$

The main functions used for evaluating access requests from users are defined by the rewrite rules we present next. The idea is similar to the standard RBAC model presented in [2], but we have extra conditions about location and time. Intuitively, given a user u asking for an access privilege a on an object o at time t and location l, we check which roles are active for u at that particular moment and position and also whether the requested action is permitted on object o at time t and location l according to the user's role privileges.

$$\begin{array}{lll} \mathsf{Access}(u, a, o, l, t) & \rightarrow & \mathsf{Check}(\mathsf{Occurs}((a, o, l, t), \\ & & \mathsf{Privl}(\mathsf{ActiveRoles}(\mathsf{roles}(u), l, t)))) \\ \mathsf{Check}(\mathsf{True}) & \rightarrow & \mathsf{Grant} \\ \mathsf{Check}(\mathsf{False}) & \rightarrow & \mathsf{Deny} \end{array}$$

Occurs checks whether the first parameter (i.e., the tuple (a, o, l, t)) is present in the list given as second parameter, that is, the list of privileges associated to the active roles of the user. The set of active roles is computed by the function ActiveRoles (defined below), which goes through the list of roles assigned to the user and discards those that are not enabled at time t and location l. The list of privileges of the active roles is computed by the function Privl, recursively, using the function Priv, as follows:

$$\begin{array}{lll} \mathsf{Privl}(\mathsf{Nil}) & \rightarrow & \mathsf{Nil} \\ \mathsf{Privl}(\mathsf{Cons}(r, lst)) & \rightarrow & \mathsf{Priv}(r) \cup \mathsf{Privl}(lst) \end{array}$$

$$\begin{array}{lll} \mathsf{ActiveRoles}(\mathsf{Nil}, l, t) & \rightarrow & \mathsf{Nil} \\ \mathsf{ActiveRoles}(\mathsf{Cons}(r, lst), l, t) & \rightarrow & \\ & & \mathsf{if}\ \mathsf{Occurs}((l, t), \mathsf{enable}(r)) \\ & & \mathsf{then} \\ & & \mathsf{Cons}(r, \mathsf{ActiveRoles}(lst, l, t)) \\ & & \mathsf{else} \\ & & \mathsf{ActiveRoles}(lst, l, t) \end{array}$$

Example 6: Consider the access control policy characterised by the user-role and permission-role assignments specified by the rewrite system in Example 5. The following access requests are evaluated using the rewrite rules defining Access, and return answers Grant and Deny respectively. Note that, according to the function enable defined in Example 5, only the role r_1 is active at that time and location:

$$\begin{array}{lll} \mathsf{Access}(u_2, w, o_1, l, t) & \rightarrow^* & \mathsf{Grant} \\ \mathsf{Access}(u_1, r, o_1, l, t) & \rightarrow^* & \mathsf{Deny} \end{array}$$

IV. Implementation Techniques

In order to increase the flexibility of the software application, it is convenient to separate the security data from the raw data, by designing a layer-structured system. Such a division allows us to handle each layer separately, assuming of course that common formats, structures and interfaces are agreed beforehand.

In a real case design, several issues need to be taken into account:

- Reliability: the rewrite-based core of our framework makes it easier to ensure the consistency and correctness of the security policy (see [2], [6], [20] for a description of the use of rewriting techniques to check consistency, totality, and correctness of access control policies).
- Flexibility: this can be achieved through modular structure and taking advantage of the powerful type systems available in rewrite-based programming languages or in functional languages.
- User-friendliness is a requirement as an argument for future users to adopt the system.
- Interface: in the context of a highly mobile and distributed environment, a software application should be accessible from several locations, and should be able to deal with access requests that are issued concurrently. Nowadays, Java is one of the most popular languages in industry for the development of web applications.

Term rewriting is a simple, declarative framework well suited to the specification of access control policies, and rewrite based or functional programming languages are particularly fit for the implementation of rewrite-based policies. Once the policy has been defined as a term rewriting system, it can be executed directly in a rewrite-based programming language such as Maude [10], or it can be easily translated to a functional language such as CAML [8]. However, in the context of a highly mobile and distributed system, a programming language with a rich collection of libraries to facilitate the development of web applications is also needed. For Maude and OCAML, interfaces with Java are available, and it is therefore possible to develop the security layer in Maude or CAML and to integrate this layer in a Java application. Another alternative is to use the TOM language [23], which is an extension of Java with rewriting features (in particular, pattern-matching is fully supported).

In the next section we describe two case studies, which have been implemented using Maude and OCAML respectively.

V. Design of web applications with access control

A. Case Study: Hospital

Consider a hospital environment and its medical information system. The basic elements composing the system are: a set of users (Renaud, Liva,...), a set of electronic patient records (EPRs) which are the objects a user can access, a set of actions composed by the read and the write actions. We consider a set of roles as follows: Doctor, Patient, OrganisationalStaff (OS), VoluntaryCaringAgency (VCA), EnvironmentalHealthOfficer (EHO). Moreover, each EPR is composed of several subrecords, such as ID, Name, Age, Sex, ClinicalData and different permissions on these records are assigned to each role. For example, a doctor can read and write on all EPRs belonging to patients he is responsible for, a patient can read all records of his own EPR, a member of OS can can read Name and ID of all EPRs, etc.

Each user can be assigned to more than one role. For example:

$$\mathsf{roles}(Renaud) \rightarrow [Doctor]$$
$$\mathsf{roles}(Clara) \rightarrow [Doctor, Patient]$$
$$\mathsf{roles}(Liva) \rightarrow [VCA, OS]$$

The policy is implemented using the programming language Maude. Given an access request of the form

```
Maude>frew in POLICY :
access(Liva,Read,EPR1,Name) .
```

the system returns a grant or deny answer according to the user-role assignments and permission-role assignments specified by the administrator.

This basic RBAC model has been extended with a hierarchy of roles. Thus, authorisation is granted if the user's role or one of its subordinate roles permits the access. For instance, OS is subordinate to Patient which is in turn subordinate to Doctor.

A time parameter has also been included, in order to evaluate requests according to the instant of time they are made. New roles such as NightDoctor and DayDoctor are introduced together with an associated time interval during which they can be activated; e.g. 9pm to 9am for the NightDoctor and 9am to 9pm for the DayDoctor. In this application, time points are represented in the format Day-HH.MM, for instance a request made on Monday at 11.30pm will have a time stamp Monday-23.30. Assuming Renaud is a NightDoctor, a request

`access(Renaud,Write,EPR1,Monday-11.30)`

will be evaluated to a denial, since the role NightDoctor cannot be activated at 11:30am. On the other hand

`access(Renaud,Write,EPR1,Monday-23.30)`

will be evaluated to grant, supposing Renaud is in charge of the patient related to EPR1.

Similarly, an additional parameter concerning location can be introduced in the model. For example, a doctor can be associated to different privileges depending on its location (hospital, or in an emergency situation in the ambulance). For example, we can assign to a doctor in an ambulance the privilege to write on all EPRs, even if the person being treated is not his patient. In this case

`access(Renaud,Write,EPR2,ambulance)`

will evaluate to Grant, while

`access(Renaud,Write,EPR2,hospital)`

evaluates to Deny if Renaud is not the doctor responsible for EPR2.

The time and location components can be easily combined in the same model simply by adding the corresponding parameters in the access request and updating the assignments defining the policy.

```
access(Renaud,Write,EPR1,Monday-11.30,
ambulance)
```

evaluates to Deny if Renaud has the NightDoctor role.

The complete description of the system, and its specification in Maude, are available at www.dcs.kcl.ac.uk/staff/clara/.

B. Case Study: Museum

We describe below a simple model of a museum, its visitors, its staff, its administrators and their rights of access to the museum rooms. Rooms are the resources in this system; each room has its own storage space, called the back room. Each type of user (staff members, visitors, etc) has different rights over rooms and back rooms.

The following data structures are used in the implementation of the access control system:

- Rooms are specified by a number, status (open or close), back room status (open or close), and a room list, which represents the rooms that have to be traversed in order to reach this room.
- Users are represented by their identification fields and assigned roles (as a list of roles); a login field and password are added in the extension of the basic model to include virtual visits.
- Actions: these are associated to rooms. To simplify the description, we assume they are the same for all rooms.
- Roles are actually associated with types of tickets (not with visitors, who are considered anonymous in the basic model), in addition to: Room manager, Security staff, Technical staff, Curator.

The system has been implemented in CAML, and is available from www.dcs.kcl.ac.uk/staff/clara/. Below we give an extract of the code defining the types of users and roles and the access function (following the rewrite-based specification given above).

```
type role = { rname: string ;
 authorizations: (room * privilege) list };;
type rbacuser = { user: user ;
                         roles: role list } ;;
(*Access Function*)
let access_check rbacuser privilege room=
    let rec searchroles l=
     match l with
    | []   -> Deny
    | t::q -> if
(member (room,privilege) t.authorizations)
            then Grant
            else searchroles q;
    in searchroles rbacuser.roles ;; }
```

An extension of the system has been proposed to deal with virtual visits. In this extension, users can use the museum's website to request viewing virtual rooms from the outside, and in addition potential visitors can use screens available in the entrance hall of the museum to browse rooms. The access control policy takes into account the location of the user in order to grant or deny access to the virtual expositions.

VI. Conclusions

We have described a generalisation of the RBAC model that addresses certain shortcomings of these models when used in a mobile computing context. TLRBAC takes into account the time and location of an access request in order to decide whether the access is permitted or denied. We have specified the model using a term rewriting framework, and discussed implementation techniques, which have been put into practice in two case studies.

The development of a framework for the smooth integration of rewrite-based access control policies in web applications is a promising research area. One of the first attempts to do this is reported in [21], using TOM. In this paper we have explored another direction, integrating rewrite-based or functional languages with Java. Further work is needed to develop tools that can aid in the specification and implementation of such systems, and for the formal verification of policy properties in the context of web applications.

References

[1] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, Great Britain, 1998.

[2] S. Barker and M. Fernández. Term rewriting for access control. In *Proc. of DBSec'06*, volume 4127 of *LNCS*, pages 179–193. Springer, 2006.

[3] G. Barthe, G. Dufay, M. Huisman, and S. Melo de Sousa. Jakarta: a toolset to reason about the JavaCard platform. In *Proc. of e-SMART'01*, volume 2140 of *LNCS*. Springer-Verlag, 2002.

[4] M. Y. Becker C. Fournet and A. D. Gordon. Design and semantics of a decentralized authorization language. In *Proc. of CSF'07*, pages 3–15, IEEE Comp. Society, 2007.

[5] E Bertino, P A Bonatti, and E Ferrari. Trbac: a temporal role-based access control model. In *Proceedings of RBAC'00*, pages 21–30, New York, NY, USA, 2000. ACM.

[6] C. Bertolissi and M. Fernández. A Rewriting Framework for the Composition of Access Control Policies. In *Proc. of PPDP 2008*, ACM Press.

[7] C. Bertolissi, M. Fernández, and S. Barker. Dynamic event-based access control as term rewriting. In *Proc. of DBSEC'07)*, volume 4602 of *LNCS*. Springer-Verlag, 2007.

[8] CAML. A functional programming language. http://caml.inria.fr/

[9] S. M. Chandran and J. B. D. Joshi. Lot-rbac: A location and time-based rbac model. In A. H. H. Ngu, M. Kitsuregawa, E. J. Neuhold, J.-Y. Chung, and Q. Z. Sheng, editors, *:Proceedings of WISE'05*, volume 3806 of *LNCS*, pages 361–375. Springer, 2005.

[10] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The Maude 2.0 system. In *Proc. of RTA'03*, number 2706 in LNCS, pages 76–87. Springer-Verlag, 2003.

[11] A Corradi, R Montanari, and D Tibaldi. Context-based access control for ubiquitous service provisioning. In *Proceedings of COMPSAC'04*, pages 444–451, Washington, DC, USA, 2004. IEEE Computer Society.

[12] M J. Covington, W Long, S Srinivasan, A K. Dev, M Ahamad, and G D. Abowd. Securing context-aware applications using environment roles. In *Proceedings of SACMAT'01*, pages 10–20, New York, NY, USA, 2001. ACM.

[13] M L Damiani, E Bertino, B Catania, and P Perlasca. Geo-rbac: A spatially aware rbac. *ACM Trans. Inf. Syst. Secur.*, 10(1), 2007.

[14] D. J. Dougherty, C. Kirchner, H. Kirchner, and A. Santana de Oliveira. Modular Access Control via Strategic Rewriting. In *Proc. of ESORICS'07*, LNCS, pages 578–593, 2007.

[15] R. Echahed and F. Prost. Security policy in a declarative style. In *Proc. of PPDP'05*. ACM Press, 2005.

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

[17] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role-based access control model. *Knowledge and Data Engineering, IEEE Transactions on*, 17(1):4–23, 2005.

[18] D. Sannella S. Kahrs and A. Tarlecki. The definition of Extended ML: A gentle introduction. *TCS*, 173(2):445–484, 1997.

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

[20] A. Santana de Oliveira. Rewriting and modularity of security policies. PhD Thesis, University Henri Poincaré, Nancy, 2008.

[21] A. Santana de Oliveira, E. Ke Wang, C. Kirchner, and H. Kirchner. Weaving Rewrite-Based Access Control Policies. Proceedings of FMSE'07, ACM Press, 2007.

[22] Song Fu and Cheng-Zhong Xu. Coordinated access control with temporal and spatial constraints on mobile execution in coalition environments. *Future Generation Comp. Syst.*, 23(6):804–815, 2007.

[23] TOM. A software environment for defining transformations in Java. http://tom.loria.fr/soft/release-2.6/manual-2.6