A Formal Semantic Model for the Access Specification Language RASP

Mark Evered
School of Science and Technology
University of New England
Armidale, Australia
mevered@une.edu.au

Abstract— **The access specification language RASP extends traditional role-based access control (RBAC) concepts to provide greater expressive power often required for fine-grained access control in sensitive information systems. Existing formal models of RBAC are not sufficient to describe these extensions. In this paper, we define a new model for RBAC which formalizes the RASP concepts of controlled role appointment and transitions, object attributes analogous to subject roles and a transitive role/attribute derivation relationship.**

Keywords: ***security, access control, model, role, attribute***

I. INTRODUCTION

In general, each of the users of an information system needs to be able to view or manipulate only some of the information stored in the system. Ideally, the appropriate access for each user will be specified in the form of an access policy during the analysis phase of the software development and then enforced via access control mechanisms during the execution of the implemented system. As the use of information systems for sensitive data continues to grow in areas such as e-health, it is becoming increasingly important, both for security and for privacy reasons, that the specification of the access control is precise and clear enough to express and satisfy strict minimal (need-to-know) policy requirements. This ensures both that valid users of a system will not misuse their access and that intruders who have illegitimately managed to assume the identity of a valid user will be restricted in what they can do within the system. Both of these factors are vital for the strengthening of cyber-security.

An access control policy can be understood as consisting of two components. The first is control over the membership of the subject groups of interest in the application domain. The second is a mapping from each of these groups to permissions which allow certain operations to be performed on the data by members of the groups. These operations may just be 'read' and 'write' as in traditional database systems or may be based on the methods of object classes as first suggested in [8].

Both components of access control have been approached in a number of different ways. In the simplest case, an access control list (ACL) for each object contains an entry for each subject or group of subjects. The owner of the object (or a system administrator) can assign subjects to groups. More recently, Role Based Access Control (RBAC) models have been defined which allow the first component of access control to be based on the roles played by individuals in the organisations making use of an information system. This means that there is a (dynamic) mapping from subjects to roles and then a (relatively static) mapping from roles to permissions. These models recognise the complex nature of permissions in real organisations and have been shown to subsume both conventional discretionary access control models and mandatory access control models such as Bell-LaPadula [1].

Formally, given:

$\mathbb{S}$ – a set of subjects

$\mathbb{R}$ – a set of roles

$\mathbb{O}$ – a set of objects and

$\mathbb{M}$ – a set of operations (methods) on objects

we can define an RBAC system as consisting of the pair:

(H, X)

where

$H \subseteq \mathbb{S}\times\mathbb{R}$ is a set of role assignments and

$X \subseteq \mathbb{R}\times\mathbb{O}\times\mathbb{M}$ is a set of permissions.

A pair `(s, r)` ∊ H specifies that the subject `s` has the role `r` while a triple `(r, o, m)` ∊ X specifies that a subject with the role `r` can access the object `o` via the method `m`.

While RBAC is an improvement over an ACL approach, case studies such as [2][6][18] have demonstrated that the access control requirements of real-world information systems are considerably more complex than the simple role-based approach described above can handle. For this reason, a number of different RBAC models have been proposed with varying degrees of additional expressive power. These additions include role hierarchies [15], parameterized roles [7] and control over role acquisition [17].

One very useful extension, as implemented in access control systems such as [9], is to allow objects to be labeled with attributes in much the same way that subjects acquire roles. Formally, we introduce the additional set:

$\mathbb{A}$ – a set of attributes with which objects can be labeled.

The permissions in such a system then give access to an object on the basis of it having a particular attribute or set of attributes rather than to objects directly. This is line with the argument in [3] that objects and environments need 'roles' just as subjects do.

The author has defined an access control specification language called RASP (Role and Attribute-based Specification of Protection) [5] which is based on both roles of subjects and attributes of objects and which gives fine-grained control over initial role and attribute acquisition as well as subsequent transitions. In this paper we give a formal definition for an access control model which supports the RASP extensions to RBAC. In particular, it supports:

- Subject roles
- Object attributes
- Control over appointment to roles
- Control over labeling of objects with attributes
- Control over dynamic acquisition of further roles and attributes

The model uses a transitive approach which supports role hierarchies, appointment based on external certificates and role and attribute revocation. No existing RBAC model has the expressive power to support these requirements.

The following section discusses related work on role-based and attribute-based access control while section III gives a brief overview of the access control specification language RASP. Section IV gives a formal definition of the access rules in our model and section V defines the instantaneous state of the RBAC model together with the four operations for transforming the state. Section VI describes the transitive acquisition of roles and attributes and defines the function `allow` for checking whether an operation on an object is permitted. Section VII defines some further useful constructs of RASP and section VIII addresses some issues of efficient implementation. We conclude with a summary of the findings and contributions of the paper.

II. Related Work

Both the object-based access control paradigm [8] and the role-based access control paradigm [14] are well-known approaches as is the combination of the two to define access to an object in terms of the methods which can be invoked by subjects acting in a certain role. A number of significant extensions to the basic RBAC model have been suggested in order to adequately handle the complexities of minimal access control requirements in real-world scenarios. These include role hierarchies [15] and role parameters [7].

A question which has received much less attention is how to group objects so that the access constraints for the whole group can be specified in a single place rather than repeating them for each and every object. The Ponder policy specification language [4] supports a hierarchical structure of domains and sub-domains of objects similar to a file system hierarchy. The leaves of the tree are references to objects rather than the objects themselves so that an object can appear in a number of different domains. This approach assumes that the domains are relatively static and that an administrator will place objects into domains via some mechanism external to the language. Case studies have shown, however, that the domains of an object may depend on object attributes which change in the same way that the role of a subject may change. These transitions require the same level of specification as to who can effect the change as is required for role changes. The approach of Generalized Role-Based Access Control [3] recognizes the need for symmetry between subject roles and object roles but does so on the basis of a very simple model which does not support role parameters or control over role transitions.

Attribute-based access control (ABAC) [19][20] was developed to support access to web services based on provable attributes of a user rather than the identity of the user. This is important for anonymity in using such services but is not appropriate for organizations or systems where fine-grained access-control policies are based on identity and roles. ABAC has been extended to include attributes for resources as well as subjects but does not address attribute transitions.

A further important question concerns the acquisition of access rights. Ponder is a delegation-based system. It provides for delegation policies which limit which access rights a subject can pass to another subject but the basic assumption is that the possessor of a right decides if and when another subject should gain that right. Case studies show that it is often necessary that access rights be granted by someone who does not possess them him/herself. The OASIS Role Definition Language [17] allows for this kind of appointment-based acquisition of access rights and for role acquisition pre-conditions based on external certificates known as auxiliary credential certificates. OASIS RDL does not however allow for a distinction between the case where a new role is replacing a previous role and the case where the new role is additional. This distinction has been found to be useful both for role transitions and for object attribute transitions. OASIS RDL also does not allow for the generation of new credential certificates as a result of operations performed within the system.

Ponder supports both positive and negative authorizations. In fact, it has two forms of negative access control clause: negative authorization policies and refrain policies. So, for example, a set of access rights can be granted to a group of subjects via a positive authorization policy and then one of the rights can later be revoked from a certain member of the group via a negative authorization policy. Negative authorizations lead to the problem of potential inconsistencies and loopholes in an access control system. A more elegant way to express this kind of partial revocation is to use role transition to transfer a subject from one role into a new role which has a more restricted set of rights.

The access control specification languages and mechanisms described in this section represent the state-of-the-art in fine-grained access control. Many of them have no formal definition at all and none of them can support all of the requirements which case studies show to be required. A formal definition of role hierarchies is given in [15] and role parameters are formally defined in [7] but no formal definition of a symmetric approach to role and attribute transitions has been given in the literature to date.

III. Overview of RASP

The three main constructs of the RASP access specification language are the `appoint` clause, the `attribute` clause and the `allow` clause. The `appoint` clause specifies that a subject with a certain role can appoint someone else to have a certain role. The precondition for this is that the person being appointed already possesses a certain role before the appointment. So, for example:

```
appoint manager: staff -> deptHead;
```

expresses the appointment rule that someone who is a manager can appoint someone who is already a staff member to be a head of department.

The `attribute` clause is used to label an object in the system with a certain attribute. This is done by specifying what role a subject must possess to be able to do this and the precondition that the object must already have a certain attribute. So, for example:

```
attribute admin: document -> obsolete;
```

expresses the attribute rule that someone who is an administrator can label a document as being obsolete.

For both the `appoint` and the `attribute` clauses, the transition symbol `'->'` indicates that the old role or attribute should be retained in addition to the new one, whereas the transition symbol `'/->'` can be used to express that the old role or attribute should be relinquished.

The third main construct is the `allow` clause. This specifies that someone with a certain role can invoke a certain operation on objects with a certain attribute or set of attributes. So, for example:

```
allow deptHead!obsolete.delete;
```

expresses the access rule that a head of department can delete an obsolete document.

RASP also provides a `conflict` clause which can be used to express the rule that two roles are in conflict with each other (if possessed by the same subject at the same time) and a `unique` clause which expresses the rule that a certain role may only be possessed by one subject at a time.

This overview of RASP will suffice for the purposes of this paper but for more detail on the rationale for and the design of the RASP language, the reader is referred to [5]. A summary of the syntax of the constructs discussed in this paper can be found in Appendix A.

IV. Access Rules

We now extend the RBAC formalism sketched in the introduction to a more powerful model which is capable of expressing the semantics of the RASP constructs described above. In this section, we define the relatively static aspect of our access control model, i.e. what access does a subject with a certain role have to an object with a certain set of attributes, who has the authority to appoint subjects to roles and who has the authority to label objects with attributes.

We define this as the 5-tuple:

(X, P, T, L, U)

where

$X \subseteq \mathbb{R} \times 2^{\mathbb{A}} \times \mathbb{M}$ is a set of permissions

$P \subseteq \mathbb{R}^3$ is a set of appointment rules

$T \subseteq \mathbb{R}^3$ is a set of role transition rules

$L \subseteq \mathbb{R} \times \mathbb{A}^2$ is a set of attribute labeling rules and

$U \subseteq \mathbb{R} \times \mathbb{A}^2$ is a set of attribute transition rules.

A permission triple $(r, A, m) \in X$, $A \subseteq \mathbb{A}$ specifies that a subject with the role r can access an object via the method m if that object has all of the attributes in the set A. Examples are:

(admin, {thisFacility, patientPersonalDetails}, update)

(secretCleared, {secret}, read)

These express the semantic value of the RASP syntax:

```
allow admin!
  {thisFacility, patientPersonalDetails}.
  update;    and
```

```
allow secretCleared!secret.read;
```

respectively (given the obvious mapping from an identifier 'admin' to the role $r_{admin} \in \mathbb{R}$ etc.).

An appointment triple $(r_1, r_2, r_3) \in P$ specifies that a subject with the role r_1 can appoint a subject with the role r_2 to additionally have the role r_3. In this context, we denote the null role (always possessed by all subjects) as $\emptyset$. So, for example we can have:

(manager, ∅, employee)

(manager, employee, admin)

(manager, doctor, doctorAtThisFacility)

These express the semantic value of the RASP syntax:

```
appoint manager: someone -> employee;
appoint manager: employee -> admin;
appoint manager: doctor ->
  doctorAtThisFacility;
```

where the identifier 'someone' is used to denote the null role .

Similarly, an attribute labeling triple $(r, a_1, a_2) \in L$ specifies that a subject with the role r can label an object with the attribute a_1 as also having the attribute a_2. Again, we denote a null attribute as $\emptyset$. For example:

(sysadmin, ∅, thisFacility)

(sysadmin, thisFacility, patientPersonalDetails)

These express the semantic value of the RASP syntax:

```
attribute sysadmin:
  something -> thisFacility;

attribute sysadmin: thisFacility ->
  patientPersonalDetails;
```

A role transition triple $(r_1, r_2, r_3) \in T$ specifies that a subject with the role r_1 can cause a subject with the role r_2 to lose that role and take on the role r_3 instead. For example,

(manager, traineeEmployee, employee)

(clearanceOfficer, secretCleared, topSecretCleared)

(manager, employee, ∅)

These express the semantic value of the RASP syntax:

```
appoint manager: traineeEmployee /->
  employee;
appoint clearanceOfficer: secretCleared
  /-> topSecretCleared;
appoint manager: employee /-> someone;
```

Note that in the last example, this kind of role transition is used to remove a role from a subject.

Finally, an attribute transition triple $(r, a_1, a_2) \in U$ specifies that a subject with the role r can cause an object with the attribute a_1 to lose that attribute and take on the attribute a_2 instead. For example:

(admin, draftReport, report)

(manager, thisFacility, thatFacility)

(declassificationOfficer, secret, unclassified)

These express the semantic value of the RASP syntax:

```
attribute admin:
  draftReport /-> report;

attribute manager: thisFacility /->
  thatFacility;

attribute declassificationOfficer:
  secret /-> unclassified;
```

V. Access State

We now define the second part of the model, which determines for some point in time, which subject has which roles and which object has which attributes. This is represented via a set of *role appointment certificates* and a set of *attribute labeling certificates*. Formally, the state of the access control system is given by:

(C, D)

where

$C \subseteq \mathbb{S}\times\mathbb{R}^2$ is a set of appointment certificates and

$D \subseteq \mathbb{O}\times\mathbb{A}^2$ is a set of label certificates.

The certificate $(s, r_1, r_2) \in C$ specifies that if the subject s has the role r_1, then that subject also has the role r_2. Thus:

(Fred, ∅, traineeEmployee)

(Fred, employee, admin)

Similarly, the certificate $(o, a_1, a_2) \in D$ specifies that if the object o has the attribute a_1, then that object also has the attribute a_2.

We define four functions which update the state of the access control system. Function $\mathtt{addRole}(C, s, r_1, r_2)$ is used to add an appointment certificate to C and is defined as:

$$\mathtt{addRole}(C, s, r_1, r_2) = C \cup \{(s, r_1, r_2)\}$$

Function $\mathtt{modRole}(C, s, r_1, r_2)$ is used to change some of the appointment certificates in C and can be defined recursively as:

```
if C contains an appointment certificate of the form $(s, r, r_1)$, for some $r \in \mathbb{R}$ then
    $\mathtt{modRole}(C, s, r_1, r_2) =$
    $\{(s, r, r_2)\} \cup \mathtt{modRole}(C \setminus \{(s, r, r_1)\}, s, r_1, r_2)$
otherwise
    $\mathtt{modRole}(C, s, r_1, r_2) = C$
```

So, for example, if C contains the certificate:

(Fred, ∅, traineeEmployee)

then $\mathtt{modRole}$(C, Fred, traineeEmployee, employee) will instead contain the certificate:

(Fred, ∅, employee)

Function $\mathtt{addAttr}(D, o, a_1, a_2)$ is used to add a label certificate to D and is defined as:

$$\mathtt{addAttr}(D, o, a_1, a_2) = D \cup \{(o, a_1, a_2)\}$$

Finally, function $\mathtt{modAttr}(D, o, a_1, a_2)$ is used to change some of the label certificates in D and is defined as:

```
if D contains a label certificate of the form $(o, a, a_1)$, for some $a \in \mathbb{A}$ then
    $\mathtt{modAttr}(D, o, a_1, a_2) =$
    $\{(o, a, a_2)\} \cup \mathtt{modAttr}(D \setminus \{(o, a, a_1)\}, o, a_1, a_2)$
otherwise
    $\mathtt{modAttr}(D, o, a_1, a_2) = D$
```

Note that a subject s may invoke $\mathtt{addRole}(C, s_1, r_1, r_2)$ only if s has a role r such that $(r, r_1, r_2) \in P$. Similarly, s can invoke $\mathtt{modRole}(C, s_1, r_1, r_2)$ only with a role r such that $(r, r_1, r_2) \in T$. Likewise, s can invoke $\mathtt{addAttr}(D, o, a_1, a_2)$ only if s has a role r where $(r, a_1, a_2) \in L$ and $\mathtt{modAttr}(D, o, a_1, a_2)$ only with a role r such that $(r, a_1, a_2) \in U$. The exact definition of 'having a role' is given in the next section.

VI. Determining Roles and Attributes

From the definitions in the previous section, it can be seen that, rather than just representing the set of roles possessed by a subject at some point in time, our model represents the role from which each role is derived. We define the notation $\langle r, r' \rangle_s$ to represent that the subject s has the role r' *conditional on having* the role r, i.e.:

$\exists r_1 \ldots r_n \in \mathbb{R}.\ (s, r, r_1) \in C \wedge (s, r_1, r_2) \in C \ldots \wedge (s, r_{n-1}, r_n) \in C \wedge (s, r_n, r') \in C$

It can be seen that this conditional possession of roles is then a transitive relationship, i.e.

$\langle r_1, r_2 \rangle_s \wedge \langle r_2, r_3 \rangle_s \Rightarrow \langle r_1, r_3 \rangle_s$

The actual possession of a role can then expressed as:

$\langle \emptyset, r \rangle_s$

Similarly, for attributes of objects, we define $\langle a, a' \rangle_o$ to mean that the object o has the attribute a' conditional on having the attribute a. So, an object actually possesses an attribute if:

$\langle \emptyset, a \rangle_o$

Finally, we can define the allow function which determines whether a subject s can access an object o via a method m as:

$\text{allow}(s, m, o) = \exists r \in \mathbb{R}, A \subseteq \mathbb{A}.$
$\langle \emptyset, r \rangle_s \wedge \forall a \in A. \langle \emptyset, a \rangle_o \wedge (r, A, m) \in X$

The fact that the model represents the possession of a role or attribute as conditional on possession of another role or attribute is very important for an adequate level of access control in real-world information systems. Suppose, for example, the set C contains the appointment certificates:

(Fred, ∅, doctor) and

(Fred, doctor, doctorAtThisFacility)

If Fred were to lose the role of 'doctor' (for example by being 'struck off' the medical register for some reason), we would want him to also automatically lose the role of 'doctorAtThisFacility' with all its associated permissions. This is only possible if the model represents the derivation of the second role from the first. Similarly, for the labeling certificates:

(DocumentAbc, ∅, Australian) and

(DocumentAbc, Australian, Sydney)

we want the document to automatically lose the attribute 'Sydney' if it loses the attribute 'Australian'. This illustrates that the transitive nature of our model can be used to support specialization hierarchies of roles and attributes. An example for roles is:

(Fred, ∅, sysadmin) and

(Fred, sysadmin, linuxSysadmin)

A further advantage of our approach is that the order of adding roles becomes more flexible. So, for example, if we have the certificates:

(Fred, ∅, traineeEmployee) and

(Fred, employee, admin)

then this represents the fact that "Fred does not yet have the 'admin' role but will acquire that role as soon as he becomes a (fully fledged) employee". The operation:

modRole(C, Fred, traineeEmployee, employee)

will then make him an 'admin' as well as an 'employee'.

Lastly, our representation of role appointment certificates supports explicit certificates which represent a precondition (e.g. for employment) which is imported from, or accessed at, an external source. For example, the certificate:

(Fred, ∅, doctor)

should ideally be maintained by an external body such as a national medical association rather than in the organization where the doctor is working. Our model provides for an explicit representation of such an external qualification certificate. (Of course, in an implementation which transfers or accesses this from an external site, it would need to be secured by a mechanism such as public-key cryptography, digital signatures and unique subject identifiers.)

VII. FURTHER FEATURES OF RASP

The main constructs of RASP are the appoint, the attribute and the allow clauses as defined above but we can also use the formal model to define the semantics of two other constructs which can be important for restricting role appointments in the information systems of real organizations.

The first of these is a clause which specifies that it is a conflict for someone to be fulfilling two certain roles in the organization at the same time. So, for example, it may be considered a conflict for someone to be both a student and a staff member of a university at the same time. The syntax for expressing this in RASP is:

```
conflict staff, student;
```

We can formally describe the semantics of this by defining functions $\text{addRoleCheckConflict}(C, s, r_1, r_2)$ and $\text{modRoleCheckConflict}(C, s, r_1, r_2)$ which extend $\text{addRole}(C, s, r_1, r_2)$ and $\text{modRole}(C, s, r_1, r_2)$ by adding a check for a breach of the constraint whenever the set of appointment certificates is updated. The definitions of these functions for the conflict roles role_id1 and role_id2 are then:

```
$\text{addRole}_{\text{checkconflict}}(C, s, r_1, r_2) =$
    if $\exists s \in \mathbb{S}. \langle \emptyset, r_{\text{role\_id1}} \rangle_s \wedge \langle \emptyset, r_{\text{role\_id2}} \rangle_s$ in
        $\text{addRole}(C, s, r_1, r_2)$ then:
      error
    otherwise:
      $\text{addRole}(C, s, r_1, r_2)$
```

and

```
modRole_checkconflict(C, s, r_1, r_2) =
    if $\exists s \in \mathbb{S}. \langle \emptyset, r_{role\_id1} \rangle_s \wedge \langle \emptyset, r_{role\_id2} \rangle_s$ in
        modRole(C, s, r_1, r_2) then:
      error
    otherwise:
      modRole(C, s, r_1, r_2)
```

The second construct is a clause that specifies that only a single subject may have a certain role at one time. So, for example, we can specify that these can only be one subject with the role `manager` at one time by the clause:

```
unique manager;
```

Again, we can formally define this construct by defining the functions addRoleCheckUnique(C, s, r_1, r_2) and modRoleCheckUnique(C, s, r_1, r_2) which check for a breach of the constraint whenever the set of appointment certificates is updated. The definitions for a unique role `role_id` are:

```
addRole_checkunique(C, s, r_1, r_2) =
   if $\exists s1 \in \mathbb{S}, s2 \neq s1 \in \mathbb{S}$ .
         $\langle \emptyset, r_{role\_id} \rangle_{s1} \wedge \langle \emptyset, r_{role\_id} \rangle_{s2}$ in
         addRole(C, s, r_1, r_2) then:
      error
   otherwise:
      addRole(C, s, r_1, r_2)
```

and

```
modRole_checkunique(C, s, r_1, r_2) =
   if $\exists s1 \in \mathbb{S}, s2 \neq s1 \in \mathbb{S}$ .
         $\langle \emptyset, r_{role\_id} \rangle_{s1} \wedge \langle \emptyset, r_{role\_id} \rangle_{s2}$ in
         modRole(C, s, r_1, r_2) then:
      error
   otherwise:
      modRole(C, s, r_1, r_2)
```

One concept of RASP that the model presented in this paper does not yet support is that of role parameters. We have deliberately excluded this concept, not because we consider it to be unnecessary or unimportant, but for the sake of brevity and of clearly describing the basic model without this complicating factor. Role parameters can however be integrated into our model and a future paper will discuss this. Existing models for role parameters such as in [7] are not sufficient for RASP since they do not describe role transitions or transitive role relationships and also do not relate the role parameters to attributes of the protected objects.

A summary of the mappings from RASP syntax to their semantics as expressed in the formal model is given in Appendix B.

VIII. IMPLEMENTATION CONSIDERATIONS

While this paper is concerned with a general model rather than a specific implementation, it is nevertheless important that any access control scheme be implementable with realistic overheads for the checking of permissions. If the definition of the allow(s, m, o) function in the previous section were to be evaluated in that form for every attempted invocation of a method on an object, then unacceptable delays would be incurred. Similarly, if the rules were to be preprocessed to a central access control matrix for all subjects and all objects then that would incur a high overhead each time a certificate was added or changed.

Fortunately, neither of these extremes is necessary. Firstly, most subjects will be interested in only a small fraction of the total number of objects and secondly, the system need only be concerned with the subjects who are currently using it. Thirdly, although the number of subjects and objects in an organization may be large, the number of roles and attributes and therefore the number of rules will generally be fairly small, even for a fine-grained access scheme. Also the kinds of operations to which the scheme is applied will generally be high-level operations like, 'read', 'edit' or 'update' on documents or databases and so will not be extremely frequent.

Finally, rather than calculate the entire set of roles allowed for a subject, it is actually preferable for each subject to acquire only the $\emptyset$ role when they start a session and then explicitly request any further role they wish to adopt for that session. This means that only those roles need be checked against the rules rather than all possible roles for that subject. The reason this is preferable is that it allows a log to be maintained of exactly who is acting in which role at what time.

An implementation could thus work along the following lines:

- assign the role $\emptyset$ to a subject who starts a session
- when a subject requests to act in a further role:
    - check for a certificate which allows this
    - if allowed, determine the attribute sets associated with this role in the permission rules
- allow a subject to searches for objects with those sets of attributes:
- when a subject selects a certain object
    - use the permission rules for the current roles to determine which operations can be performed on the object

Given appropriate index tables for the rule and certificate information, none of these individual steps need incur an unacceptable overhead.

IX. CONCLUSION AND FUTURE WORK

Case studies show that information systems often require a degree of access control which cannot be expressed simply as a static mapping from subjects to roles and from roles to operations on objects.

In this paper, we have formally defined a role-based access control model which has a much greater expressive power and which, in particular, can be used to formally describe the semantics of the RASP access specification language.

The model supports controlled dynamic acquisition of new roles, transitions from one role to another and role revocation. It also supports labeling of objects with attributes in a way analogous to appointing subjects to roles and defines permissions in terms of roles and attribute sets.

We have defined the access control model in two parts. The first represents the rules for role appointment and attribute labeling as well as role and attribute transitions and access permissions. The second part of the model represents the instantaneous state of the access system in terms of a set of appointment certificates and a set of labeling certificates. We have defined four functions for updating these sets.

A significant aspect of the model is the use of transitive relationships whereby a certificate represents the fact that the possession of a role or attribute may be conditional on the possession of another role or relationship. This allows the model to support role and attribute specialization hierarchies, controlled revocation of derived roles/attributes and flexibility in the addition of roles.

No existing formal model for role-based access control supports all the concepts captured in our model.

References

[1] D.E. Bell and L.J. La Padula, "Secure computer systems: unified exposition and Multics interpretation", MTR-2997, The MITRE Corporation, 1975.

[2] B. Blobel, "Authorisation and access control for electronic health record systems", International Journal of Medical Informatics, 73, 2004.

[3] M.J. Covington, M.J. Moyer and M. Ahamad, "Generalized role-based access control for securing future applications", Proc. 23rd National Information Systems Security Conference, Baltimore, 2000.

[4] N. Damianou, N. Dulay, E. Lupu and M. Sloman, "Ponder: A language for specifying security and management policies for distributed systems", The Language Specification Version 2.3, Imperial College Research Report DoC 2000/1, 2000.

[5] M. Evered, "Rationale and Design of the Access Specification Language RASP", Intl. Journal of Cyber-Security and Forensics, 1, 1, 2012.

[6] M. Evered and S. Bögeholz, "A case study in access control requirements for a health information system", Proc. Australasian Information Security Workshop, Dunedin, 2004.

[7] J.H. Hine, W. Yao, J. Bacon and K. Moody, "An architecture for distributed OASIS services", Proc. Middleware 2000, Lecture Notes in Computer Science, Vol. 1795, Springer-Verlag, Heidelberg/New York, 2000.

[8] A. Jones and B. Liskov, "A language extension for expressing constraints on data access". Communications of the ACM, 21(5):358-367, May, 1978.

[9] T. Moses (Ed.), Extensible Access Control Markup Language (XACML) Version 2.0, OASIS Consortium, 2005.

[10] Object Management Group, Resource Access Decision Facility Specification, Version 1.0, 2001.

[11] Object Management Group, Object Constraint Language Specification Version 2.0, 2006.

[12] G. Russello, C. Dong and N. Dulay, "Authorisation and conflict resolution in hierarchical domains", Proc. 8th IEEE Workshop on Policies for Distributed Systems and Networks, Bologna, 2007.

[13] J.H. Saltzer, "Protection and the control of information sharing in Multics", Symposium on Operating System Principles, Yorktown Heights, NY, 1973.

[14] R. Sandhu, E.J. Coyne, H.L. Feinstein and C.E. Youman, "Role based access control models", IEEE Computer 29 (2), 1996.

[15] R. Sandhu, "Role activation hierarchies", Proc. 3rd ACM Workshop on Role-Based Access Control, Fairfax, 1998.

[16] M.C. Tschantz and S. Krishnamurthi, S. "Towards reasonability properties for access conrol policy languages", Proc. 11th ACM Symposium on Access Control Models and Technologies, Lake Tahoe, 2006.

[17] W. Yao, K. Moody and J. Bacon, "A model of OASIS role-based access control and its support for active security", ACM Transactions on Information and System Security, 5, 4, 2001.

[18] P. Yu and H. Yu, H., "Lessons learned from the practice of mobile health application development", Proc. 28th Annual International Computer Software and Applications Conference, Hong Kong, 2004.

[19] T. Yu, X. Ma and M. Winslett, "Prunes: an efficient and complete strategy for automated trust negotiation over the internet", Proc. 7th ACM conference on Computer and communications security.ACM Press, 2000.

[20] E. Yuan and J. Tong, "Attributed based access control (ABAC) for web services", Proc. IEEE International Conference on Web Services, 2005.

Appendix A – Concrete syntax of relevant RASP constructs

```
clause: appoint_clause |
        attribute_clause |
        allow_clause |
        conflict_clause |
        unique_clause


appoint_clause: 'appoint'
        role_id ':'
        role_id transition role_id ';'
```

```
transition: '->' | '/->'


attribute_clause: 'attribute'
       role_id ':'
       attribute_id transition
       attribute_id ';'


allow_clause: 'allow' role_id '!'
        action ';'
```

```
action: attribute_id '.' operation_id


action: '{' attribute_list '}' '.'
        operation_id


attribute_list: attribute_id
         { ',' attribute_id }


conflict_clause: 'conflict'
       role_id ',' role_id ';'


unique_clause: 'unique' role_id ';'
```

Appendix B – Summary of semantic mappings

```
'appoint' role_id1 ':'
   role_id2 '->' role_id3  ⇒
```

$P' = P \cup \{ (r_{role\_id1}, r_{role\_id2}, r_{role\_id3}) \}$

```
'appoint' role_id1 ':'
   role_id2 '/->' role_id3  ⇒
```

$T' = T \cup \{ (r_{role\_id1}, r_{role\_id2}, r_{role\_id3}) \}$

```
'attribute' role_id ':'
   attr_id1 '->' attr_id2  ⇒
```

$L' = L \cup \{ (r_{role\_id}, a_{attr\_id1}, r_{attr\_id2}) \}$

```
'attribute' role_id ':'
   attr_id1 '/->' attr_id2  ⇒
```

$U' = U \cup \{ (r_{role\_id}, a_{attr\_id1}, r_{attr\_id2}) \}$

```
'allow' role_id '!'
   attr_id '.' op_id ';' ⇒
```

$X' = X \cup \{ (r_{role\_id}, \{a_{attr\_id}\}, m_{op\_id}) \}$

```
'allow' role_id '!'
   '{' attr_id1 ','
       attr_id2 ',' …
       attr_idn '}'
   '.' op_id ';' ⇒
```

$X' = X \cup \{ (r_{role\_id}, \{a_{attr\_id1}, a_{attr\_id2}, \ldots a_{attr\_idn}\}, m_{op\_id}) \}$

```
'conflict' role_id1 ',' role_id2 ;'  ⇒
```

```
addRole_checkconflict(C, s, r1, r2) =
   if ∃s∈𝕊.⟨∅, r_role_id1⟩s ∧ ⟨∅, r_role_id2⟩s in
        addRole(C, s, r1, r2) then:
      error
   otherwise:
      addRole(C, s, r1, r2)

and

modRole_checkconflict(C, s, r1, r2) =
   if ∃s∈𝕊.⟨∅, r_role_id1⟩s ∧ ⟨∅, r_role_id2⟩s in
        modRole(C, s, r1, r2) then:
      error
   otherwise:
      modRole(C, s, r1, r2)
```

```
'unique' role_id ';'  ⇒
```

```
addRole_checkunique(C, s, r1, r2) =
   if ∃s1∈𝕊, s2≠s1∈𝕊 .
        ⟨∅, r_role_id⟩s1 ∧ ⟨∅, r_role_id⟩s2 in
        addRole(C, s, r1, r2) then:
      error
   otherwise:
      addRole(C, s, r1, r2)

and

modRole_checkunique(C, s, r1, r2) =
   if ∃s1∈𝕊, s2≠s1∈𝕊 .
        ⟨∅, r_role_id⟩s1 ∧ ⟨∅, r_role_id⟩s2 in
        modRole(C, s, r1, r2) then:
      error
   otherwise:
      modRole(C, s, r1, r2)
```