INTEGRATION OF ACCESS CONTROL IN INFORMATION SYSTEMS: FROM ROLE ENGINEERING TO IMPLEMENTATION

Romuald Thion, Stéphane Coulondre
LIRIS / INSA
University of Lyon
20 Av. Albert Einstein
69621 Villeurbanne Cedex, France
Tel: +33 472 43 70 55; fax: +33 472 43 87 13
e-mail: romuald.thion@insa-lyon.fr

ABSTRACT

Pervasive computing and proliferation of smart gadgets make organizations open their information systems, especially by extensive use of mobile technology: information system must be available any-time, any-where. This cannot be performed reasonably without thorough access control policies. Such an access control must be able to deal with user's profile, time and eventually with other complex contexts like geographical position.
This paper shows that it is possible to take into account confidentiality constraints straight into the logical data model in a homogeneous way, for various aspects treated independently (user profile, time, geographical position, etc.). We propose a language called LORAAM which includes a way to express authorizations at the class level. We first present the syntactical aspects, then the semantics of such a language, based on the object-oriented paradigm.

1 INTRODUCTION

Companies and public interest for new technologies keeps growing, either for mobile use (laptops, Wi-fi, pocket-PC, GPS, UMTS, Java technology in GSM, etc..) or for "traditional" use. Information systems now become open and online, and their security must be guaranteed. These new technologies lead to the concept of context: a new constraint to consider in access control to the information system services. From now on, access control mechanisms tend towards integration of user profile, time, state of the computing environment and even geographical position.

In this paper, we show how to take into account general context data (user roles, spatio-temporal environment, etc.) in a homogeneous way, straight in the object data model (and more generally in Information Systems, Objects, Web Services, etc). Indeed security management, and especially access control, is often postponed until the end of the design cycle and is implemented at the end of the process. The software is therefore often developed without taking confidentiality constraints into account. We think that confidentiality must also be present throughout the whole development cycle. Our proposal provides a logical data model in which contextual role-based access control is integrated: we thus provide a support to upstream design methods [1,2] which rely on it.

Section 2 presents the original Role-Based Access Control that our proposal uses for the organisation of privileges within an information system, and surveys previous work in attempting to integrate the role concept in logical object data models for security purpose. Section 3 details syntactical and functional aspects of the LORAAM language we propose, together with an illustrative example in the medical area. Section 4 finally concludes the paper and discusses some perspectives.

2 THE RBAC MODEL

2.1. An Access Control Model

The RBAC Model [3] was defined in the 90's and has been extended in many ways since (temporal, geographical extensions, etc). It was introduced in order to tackle the weaknesses of DAC (Discretionary Access Control) and MAC (Mandatory Access Control) models: the former is difficult to implement whith a large number of users, and the latter is too rigid for modern applications.

The basic RBAC philosophy is based on the observation that most of the access permissions are determined by a person authority or function, inside an organisation. This defines the central concept of role. The introduction of role concept in access control policies as an intermediate layer between subjects and permissions, really facilitates and simplifies the system administration task. The RBAC definition of a role is "a job function within the organization with some semantics regarding the authority and responsibility conferred on the member of the role".

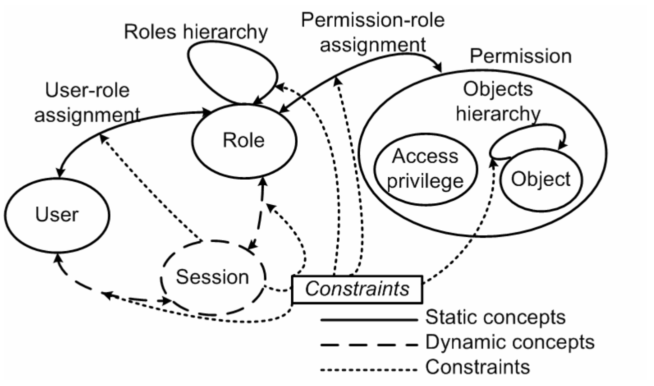

Figure 1. *The RBAC model [3]*

The RBAC model family is based on the identification of a certain number of roles [4], each of them representing a set of actions and responsibilities within the system. Thus in the RBAC model (figure 1):

- No permission is granted directly to the subject (ex: user, process, object...), permissions are only granted to roles
- The subjects endorse the roles which are given by the administrator (it is only possible to specify positive authorisations, no prohibitions).
- Roles are defined and organised in a hierarchy: a child role has the permissions granted to his/her parents.

An example of confidentiality policy in a hospital would be:

- A nurse can only read the patient prescriptions. But she can write the last care date and time, provided it takes place during her working time.
- A doctor can only prescribe if he/she is geographically located in the hospital. He has access to the whole medical record, but he/she cannot write the last care date and time.
- A head nurse has read access to prescriptions and cares history without conditions of time.

Permissions associated to roles allow expressing access authorisation in a generic way. Therefore we do not specify that "Dr. Johnson" has access to "Mr. Rabot" record. Instead we only specify that doctors have write access to patient records. Note that in this paper we only take into account the static aspect (i.e. not related to individuals) of RBAC access control. Thus is is not possible to specify that only "Dr. Johnson" has access to "Mr. Rabot" record. The RBAC roles, their hierarchical organisation and the associated permissions constitute the organisation confidentiality policy.

2.2. Related work

The object paradigm is a very expressive framework, largely used. However, implementing object roles is a difficult task. Indeed, the multiplicity of roles and their lifecycle (creation, deletion) is incompatible with the hard constraints of class-based models: object identity, strong typing, etc. Very few work focused on integrating of RBAC within logical data models. Therefore, confidentiality constraints are unfortunately taken into account at the end of the development process, by mean of various techniques added on top of the applications.

This problem could be partly solved with multiple inheritance (figure 2a) in an object programming language. But each combination of role must lead to create a new class, which leads to an explosion of the number of necessary classes. Moreover, their existence is only motivated by technical reasons and not by a modelling need. Another solution is to create a structure of "handles" [5] (figure 2b) which corresponds to the desired multiple-role instances. The handle references several OIDs, each of them corresponding to a role played by this instance. This leads to a referencing problem and involves the use of message delegation. Moreover, "Jacques" would be only a "handle", loosing its encapsulation, and therefore not an object anymore.

A review of role-based object models in the programming object and database areas can be found in [6,7]. However, these models are intended mainly to take into account the evolutive part of the objects during their life, but either they do not propose in general any access control primitive or they do not totally respect the standard paradigms of object programming [8].

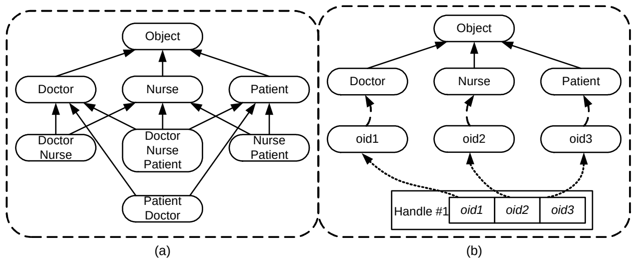

Figure 2. *Empirical solutions for role implementation*

3. THE LORAAM LANGUAGE

In order to tackle the problems of RBAC integration within object data models, we propose a generic language LORAAM allowing the expression of RBAC authorisations and integrating an access control mechanism. The declarative part of the language is composed of:

- The body, which relies on C++ syntax (on a purely illustrative basis, as any class-based language could have been used: Java, Python, etc.) while adding access authorisations formulae to methods.
- The header, which defines the roles which are to be used in the definition of access authorisations.

3.1. The Header

The header is used to specify:

- Various categories of roles to be taken into account. In this example we included the categories of [9] which are adapted to organisations: functional, seniority and context. These categories, freely chosen by the developer, form groups of roles. These groups represent transverse role aspects, which are combined to form complex roles. It would be possible to add some other groups such as "ward", (ex: cardiology, radiology, etc. which remains static), or "classification" (ex: white, grey, black information according to the sensitivity of data) which can be used for example to simulate a MAC access control.
- Hierarchical relations between roles [10]. For example *head* << *assistant* means that the head has (at least) all the privileges of the assistant. Thus, the conjunction of these roles with a functional role "doctor" makes it possible to specify complex roles, for example "head doctor", who would have more privileges than a “simple" doctor, but fewer privileges than the manager (who is also a doctor).
- The various contexts in which the access authorisations are defined. These contexts can be geographical (by using the predicate "position") or temporal (with the predicate "hour"). We suppose that the position of the user is obtained by reliable mechanisms which are not in the scope of this paper. We suppose we can get an absolute reference as a couple of (X, Y) co-ordinates, indicating the user position from where he/she invokes the service. In practice, space modelling by mean of linear constraints is sufficient for many cases [11]. Within the header, we can for example restrict access only if the user is located within the hospital or the building.

All simple roles defined in the header are combinable via conjunctions and disjunctions, in order to create complex roles, modelling access control constraints based on the transverse aspects of the profile, time and space at the same time.

```
Functional Roles {
Roles : nurse, doctor, day_nurse, night_nurse;
Hierarchy : day_nurse << nurse, night_nurse <<
nurse ;
}

Seniority Roles {
Roles : manager, head, assistant;
Hierarchy : manager << head << assistant;
}

Contextual Roles {
Hospital_enclosure = (position(X,Y) and X>10 and
X<50 and Y<10 and Y>30);
First_shift = (hour(H) and H>=4 and H<12)
Second_shift = (hour (H) and H>=12 and H<20);
Third_shift = ((hour (H) and H>=20) or (hour(H)
and H<4));
}
```

3.2. The Body

In LORAAM, the body part allows the expression of access authorisations at the method level. This is made possible using the *auth* keyword, followed by an appropriate logical formula. The authorisation logical formulae condition access to each method, according to the roles defined in the header. These access authorizations can model access control rules defined in the confidentiality policy.

```
Class CElectronicPatientRecord {
Public:
contact getPatientContact()
auth (doctor or nurse);
string getLastPrescription()
auth (doctor or nurse);
string getPrescriptionHistory()
auth (doctor or (nurse and head));
string getCareHistory()
auth (doctor or (nurse and head));
void setPrescription(string prescription)
auth (doctor and Hospital_enclosure);
void setLastCare(hour h, string care)
auth ((day_nurse and first_shift)
or (day_nurse and second_shift)
or (night_nurse and third_shift));
/* This authorization prevent day nurse from
filling the LastCare field of the e-Patient
record during night and night nurse during the
day */
}
```

3.3. Functional Aspects

As the access control we propose is defined at the class level, the following statements hold :

- For confidentiality-critical applications, access control authorisations should be taken into account from the very start of an information system design cycle [1]. We do think that it does not have to be postponed until the end of the cycle.
- Roles must be defined as soon as the requirement engineering stage.
- Roles and authorizations can only be static, as the class structure is modified, therefore recompiling is necessary. We consider that this is not necessarily a major problem, as the set of information defined in the header and authorizations are very static (ex: hierarchical levels, internal organisation, administrative responsibilities, etc.). However, no recompiling is necessary for dynamic user role assignment or revocation. Moreover, privilege delegation is possible between users.

The principle of access control decision is as follows: when a method call is detected, the LORAAM engine checks if the dynamic user profile logical formula implies the method authorisation. The dynamic user profile is constructed as follows: each role r is defined within a category c, and is associated to a logical first-order atom $c(r)$. The profile is obtained by conjunction of all played roles and their parents roles. Contextual information is obtained by mean of software/hardware tools such as LDAP, GPS, time clock, etc. and also translated in a logical formula. If the implication is valid, the method is invoked, else an catchable exception is raised.

3.4. Example

Let us suppose that a user, John, wants to access the *setLastCare()* method from his mobile device. John, who has previously identified himself on the information system, has a profile *functional(nurse) and functional(night_nurse) and position(150,45) and hour(23)*. The functional part can be extracted from a LDAP directory for example, and the spatio-temporal part can be added by a time and position server.

The authorisation formula associated with the *setLastTreatment()* method is specified within the LORAAM body, as *((day_nurse and first_shift) or (day_nurse and second_shift) or (night_nurse and thrid_shift))*. The LORAAM engine replaces these role names by logical predicates, as defined in the header:

- *day_nurse* is replaced by *functional(nurse) and function(day_nurse)*. Indeed, *day_nurse* has at least all the privileges of *nurse*. The same hold for *night_nurse*.
- *first_shift* is replaced by *hour(H) and H>=4 and H<12.* The same holds for *second_shift* and *third_shift*.

The resulting formula (under disjunctive form) is *(functional(nurse) and functional(night_nurse) and hour(H) and H<4) or (functional(nurse) and functional(night_nurse) and hour(H) and H>=20) or (functional(nurse) and functional(day_nurse) and hour(H) and H>=4 and H<12) or (functional(nurse) and functional(day_nurse) and hour(H) and H>=12 and H<20)*. The LORAAM engine checks if the dynamic user profile logical formula implies this formula. As the user profile is *functional(nurse) and functional(night_nurse) and position(150,45) and hour(23),* we can see that the implication holds. Therefore, access is granted.

4. DISCUSSION

Our proposal makes it possible to take into account RBAC access control to information systems straight into the logical object data model. We presented the generic LORAAM language, which contains two parts. The header allows specification of roles categories and hierarchies. The body part allows specification of authorisations at the method level, by use of logical connectors in order to build more complex ones. We also presented the functional part of LORAAM, which relies on a first-order logic engine.

Software quality best practises recommend the specification of access control constraints at the very beginning of the design process. Thanks to LORAAM, the information system architect can thus directly integrate authorizations in his logical data model in a declarative way, without worrying about the corresponding underlying mechanisms.

This methodology implies that the architect must conduct the role engineering process prior to specifying the information system data model, thus auditing the target internal organization, as well as contextual information system usage (temporal constraints, mobile access, etc.).
We currently work on automatic translation into LORAAM of UML diagrams expressed in specific security models [1,2]. LORAAM can indeed be used as a target language for a CASE supporting a RBAC-based design method, such as SecureUML. We currently plan to validate this approach using our prototype, a LORAAM to C++ preprocessor, with the Foundstone SecureUML Visio template [12].

References

[1] Kim D.K., Ray I., France R., Li N., Modeling Role-Based Access Control Using Parameterized UML Models, *Proceedings of Fundamental Approaches to Software Engineering (FASE/ETAPS)*, Springer-Verlag, p. 180-193, 2004.

[2] Lodderstedt T., Basin D., Doser J., SecureUML: A UML-Based Modeling Language for Model-Driven Security, *Proceedings of the 5th International Conference on The UML*, Springer, p. 426-441, 2002

[3] Sandhu R., Coyne E., Feinstein H., Youman C., Role-Based Access Control Models, *IEEE Computer*, vol. 29, num. 2, p. 38-47, 1996.

[4] Roeckle H., Schimpf G., Weidinger R., Process-Oriented Approach for Role-Finding to Implement Role-Based Security Administration in a large industrial organization, *Proc. of the 5th ACM Workshop on RBAC*, p 103-110, 2000

[5] Thomsen D., O'Brien D., Bogle J. Role Based Access Control Framework for Network Enterprises, *14th Annual Computer Security Applications Conference*, 1998.

[6] Kappel G., Retschitzegger W., Schwinger W., A Comparison of Role Mechanisms in Object-Oriented Modeling, *Modellierung '98, Proceedings des GI-Workshops*, Munster, 1998.

[7] Coulondre S., Libourel T., An Integrated Object-Role oriented Model, *Data and Knowledge Engineering*, Vol 42, num 1, p 113-141, Elsevier Science, 2002.

[8] Wong R., RBAC Support in Object-Oriented Role Databases, *Proceedings of the 2nd ACM Workshop on Role-Based Access Control*, p 109-120, 1998

[9] Crook R, Ince D., Nuseibeh B., Modelling Access Policies using Roles in Requirements Engineering, *Information and Software Technology*, issue 45, p. 979-991, 2003

[10] Moffett J.D., Lupu E.C., The Uses of Role Hierarchies in Access Control, *Proceedings of the 4th ACM Workshop on Role-Based Access Control*, p 153-160, 1999

[11] Grumbach S., Rigaux P., Segoufin L., Spatio-Temporal Data Handling with Constraints, *GeoInformatica*, vol. 5, num 1, Kluwer Academic Publishers, p. 95-115, 2001

[12] Araujo R., Gupta S., Design Authorisation Systems Using SecureUML, Foundstone Inc. Whitepaper, 2005.