Integration of Access Control in Information Systems: From Role Engineering to Implementation

Thion Romuald and Coulondre Stéphane
LIRIS / INSA University of Lyon
20 Av. Albert Einstein
69621 Villeurbanne Cedex, France
romuald.thion@insa-lyon.fr and stephane.coulondre@insa-lyon.fr

Keywords: role-based access control, object-oriented models, role engineering, security

Received: November 11, 2005

Pervasive computing and proliferation of smart gadgets lead organizations to open their information systems, especially by extensive use of mobile technology: information systems must be available any-time, any-where, on any media. This cannot be done reasonably without thorough access control policies. Such access control must be able to deal with user profile, time and even with more complex contexts including 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 generally treated independently (user profile, time, geographical position, etc.). We propose a language called RAPOOL which allows the expression of authorizations at the class level. We first present the syntactical aspects, then the semantics of the language, based on the object-oriented paradigm.

Povzetek: Članek opisuje mobilne informacijske sisteme.

1 Introduction

Companies and general 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 legacy use. Information Systems (IS) now become open and online. Therefore, security is in great demand, particularly for IS containing confidential data. New mobile and pervasive technologies introduce the concept of context. The need to include it in access control mechanisms arises [16, 13]. Thus, from now on, access control tends towards integration of user profile, time, state of the computing environment and even geographical position [8].

1.1 Motivations

Security is considered as a non-functional requirement in software engineering. Contrary to other non-functional requirements, such as efficiency, modularity or usability, confidentiality has been unconsidered for long. Thus, access control management is often postponed until the end of the design cycle and is implemented at the very end of the design process. The software is therefore developed without taking confidentiality constraints into account. This usual approach often leads to serious design challenges (e.g. integration of roles) and problems (e.g. software vulnerabilities, information leakage) [17].

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) to provide an object-oriented language which allows the expression of authorizations at the class level. We do think that security must be present throughout the whole development cycle. Our proposal describes a logical data model in which contextual role-based access control is integrated. We thus provide a support to upstream design methods [9, 14] which can rely on it.

1.2 Our approach

We do think that security concerns must be considered all the development cycle long. As user interface is incorporated in software architecture (e.g. the model-view-controller architecture separates an application's data model, user interface and its control logic), we argue that access control, and in a broader sense security, must be considered in the software development cycle, and not neglected until the very end of this process.

To bridge this gap between the security will and its implementation, our approach is to provide a consistent logical data model including role-based access control policies. Since Role-Based Access Control (RBAC) is currently one of the most seducing security models and since extending Object-Oriented (OO) systems with roles has been amply studied in the literature, we chose to integrate role-based authorization policies at the class level. With such embedded authorizations in an OO language, developers can now integrate their security policies in their code in a declarative manner. As inheritance is used by programmers without worrying about how polymorphism or dynamic linking are implemented, authorization policies can be used without worrying about the mechanisms involved in the authorization decision.

The rest of the paper is organized as follows. Section 2 presents the original Role-Based Access Control on which our proposal relies for describing privileges organization within an IS. Section 3 details syntactical aspects of the RAPOOL (for Role-based Authorizations Policies Object-Oriented Language) language we propose. Section 4 details functional aspects with an illustrative example in the medical area. This section also describes how to implement RAPOOL. Section 5 surveys attempts in integrating the role concept in object data models and compare our approach to related work on security integration within object-oriented models. Section 6 finally concludes the paper and discusses some perspectives on security integration within the software design process.

2 Modelling authorizations with roles

Role modelling has been introduced into many computer sciences areas: databases, programming languages, ontologies or agent oriented modelling. For security purposes, roles have been introduced to make access control policies administration easier: this is the main idea of the *Role-Based Access Control* (RBAC) model.

2.1 An Access Control model

Access control denotes the fact of determining whether a *user* (not necessarily an human user: process, computer, etc.) is able to perform an *operation* (read, write, execute, delete, search, etc.) on an *object* (more generally: a tuple in a database, a table, an object, a file, etc.). An operation right on an object is called *permission*. An access control model define how to organize the permissions of users.

The RBAC Model [21] was defined in the 90's and has been extended in many ways (temporal, geographical extensions, etc. [8, 13]). 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 with a large number of users, and the latter is too rigid for modern applications. We focused on RBAC rather than other recent access control models because it is currently the most seducing access control paradigm, as shown by its use in major databases management systems such as Oracle Enterprise Server v.8 or Sybase Adaptive Server v.11.5. Even for legacy systems which are not role- based, the use of RBAC may simplify management [18].

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 organization. This defines the central concept of role. The introduction of role concept in access control policies as an intermediate layer between users 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"([21]).

The RBAC model family is based on the identification of a certain number of roles [20], each of them representing a set of actions and responsibilities within the system (roles can be seen as a collection of permissions). Thus in the RBAC model (figure 1):

- no permission is granted directly to the user, permissions are only granted to roles,
- the users endorse the roles which are given by the administrator (it is only possible to specify positive authorizations, no prohibitions),
- roles are defined and organized in a hierarchy: a child role has the permissions granted to his/her parents.

In OO systems, the concept of permission is related to objects methods. A permission is an access privilege on a ressource. In OO systems, ressources are objects (or attributes of objects) and access privileges are objects methods (e.g. `getAttribute()` is a read-access, `setAttribute()` is a write-access). Thus, granting an access privilege to an object consists in authorizing the object method call. The rules defining permission assignments to roles are *access control policies*.

2.2 Access control policies

Access control policies define the users rights on objects, in order to enforce the security of an organization. In the RBAC model, policies define which permissions are granted to roles (*permission-role assignment* in figure 1). Thus users are granted permissions through role-assignment (*user-role assignment* in figure 1).

An example of role-oriented access control policies in health sector 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 the expression of access authorizations in a generic way. Therefore we do not specify that *Dr. Johnson* has access to *Mr. Rabot* records. Instead we only specify that doctors have write access to

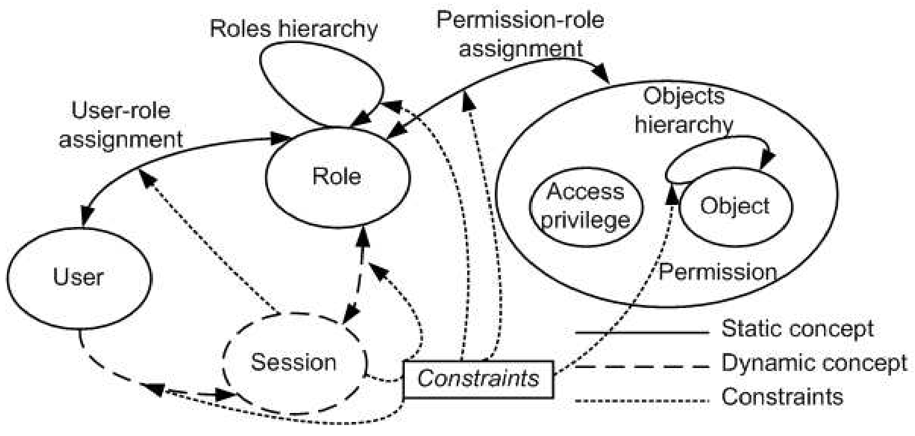

Figure 1: The RBAC Model

patient records. According to RBAC principle “permissions are only granted to roles", our proposal do not include policies related to individuals. Thus is is not possible to specify that only *Dr. Johnson* has access to *Mr. Rabot* record. The RBAC roles, their hierarchical organization and the associated permissions make up the organization confidentiality policy.

The language we chose to express access control policies has been heavily influenced by [8] which formalize authorization policies, including temporal aspects, in first-order logic (FOL). Thus we use a tractable fragment of FOL (no function symbol, no negation, only conjunctive and disjunctive connectors) suitable to express role-based authorization policies at the class level.

2.3 A suitable subset of RBAC

Our approach intends to include role-based authorizations into class models. Thus, authorizations policies are common to every objects instantiated from a class. As classes are most-of-the-time static in OO systems, we are not able to express dynamic aspects of RBAC. The session concept for example, cannot be included into the class model: each session is related to exactly one user, and it represents the roles its user is actually endorsing.

Moreover, our proposal is an OO language designed for secured software. Its goal is not to integrate the whole RBAC model into class models, but only a subset describing authorizations policies related to the application which is to be developed. Thus user-role assignments, sessions or delegations are outside the scope of this work: we only model the authorization policies related to the application. For example, let us suppose that a hospital is developing an intranet web-portal. User-role assignments are stored in a dedicated directory (which is used by other applications), not in web-portal itself. Only the policies describing “which role can access a given method of a given class of the web-portal" are included straight into the code.

Thus, the concepts of user, user-role assignment, sessions and context retrieval are not included in our proposal and will be referred as *user profile* (section 4).

3 The RAPOOL Language

In order to tackle the problems of RBAC integration within object data models, we propose a generic language RAPOOL allowing the expression of RBAC authorizations 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 authorizations formulae to methods,
- *the header*, which defines the roles which are to be used within the definition of access authorizations.

3.1 The Header

The header is used to specify (BNF grammar is given in annex):

- various categories of roles to be taken into account. In the example we included the categories of [3] which are adapted to organizations: *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 *sensitivity* (ex: white, grey, black information according to the sensitivity of data) which can be used to simulate a MAC access control,
- hierarchical relations between roles [15]. For example *head* $<<$ *assistant* means that the head has (at least) all the privileges of the assistant. Thus, the conjunction of seniority roles with the functional role *doctor*

makes it possible to specify complex roles, for example *head doctor*, who would have more privileges than a doctor, but fewer privileges than the *manager doctor*,

- the various contexts in which the access authorizations 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 [7]. 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 RAPOOL, the body part allows the expression of access authorizations at the method level. This is made possible using the *auth* keyword, followed by an appropriate logical formula. The authorization logical formulae are used to condition access to each method, according to the roles defined in the header. These access authorizations model access control rules defined in the confidentiality policy (section 2.2).

```
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 prevents a day nurse from
 filling the LastCare field of the e-Patient
 record during night, and a night nurse during the day */
}
```

4 Functional aspects

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

- for confidentiality-critical applications, access control authorizations should be taken into account from the very beginning of an information system design cycle [17]. 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 [20, 11],
- roles and authorizations can only be static [3], 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 organization, administrative responsibilities, etc.). However, no recompiling is necessary for dynamic user role assignment or revocation. Moreover, privilege delegation is possible between users. In the case of developing a wrapper (for accessing legacy application through web services for instance), recompilation does only involve the wrapper, not the wrapped applications.

4.1 The authorization decision

The principle of access control decision is as follows: when a method call is detected, the RAPOOL engine checks if the dynamic user profile fulfills the method authorization policy. As described in section 2.3, our proposal does not include the management of user profiles: we suppose that a system storing role assignments, running sessions and providing contextual information (e.g. time) exists. Once this information is retrieved (a cache mechanism can be used to improve retrieval efficiency), the RAPOOL engine can check if the requested access is granted. An architecture for such a context repository is described in [16].

The basic idea of the access control decision is based on logical implication. The user profile and the authorization policy of the requested method needs to be translated into first-order logic formulae:

1. each role is replaced by itself and the conjunction of all its parents roles. If two roles are set to be mutually inherited, they are considered as a same role,
2. each category c act as a predicate symbol. Each role r defined within c is replaced by atom $c(r)$,

3. if a role is equivalent to a formula, then it is replaced by this formula,

Once theses transformations are applied to both user profile and requested authorization formula, we need to add contextual information to the user profile. This information is obtained by mean of software/hardware tools such as LDAP, GPS, time clock, etc. and are also translated into atoms. E.g. $hour(18)$ or $position(10, 23)$. Then if the user profile (plus context) implies the authorization formula of the requested method, the method is invoked, otherwise a catchable exception is raised.

4.2 Example of authorization decision

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 the following profile: $functional(nurse) \wedge functional(night\_nurse) \wedge position(150, 45) \wedge hour(23)$ 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 authorization policy associated with the *setLastCare()* method is specified within the RAPOOL body, as $((day\_nurse \wedge first\_shift) \vee (day\_nurse \wedge second\_shift) \vee (night\_nurse \wedge third\_shift))$ The RAPOOL engine replaces these role names by logical predicates, as defined in the header:

- day_nurse is replaced by $functional(nurse) \wedge functional(day\_nurse)$. Indeed, day_nurse has at least all the privileges of $nurse$. The same hold for $night\_nurse$,
- $first\_shift$ is replaced by $hour(H) \wedge H \geq 4 \wedge H < 12$. The same holds for $second\_shift$ and $third\_shift$.

The resulting formula (under disjunctive form) is: $(functional(nurse) \wedge functional(night\_nurse) \wedge hour(H) \wedge H < 4) \vee (functional(nurse) \wedge functional(night\_nurse) \wedge hour(H) \wedge H \geq 20) \vee (functional(nurse) \wedge functional(day\_nurse) \wedge hour(H) \wedge H \geq 4 \wedge H < 12) \vee (functional(nurse) \wedge functional(day\_nurse) \wedge hour(H) \wedge H \geq 12 \wedge H < 20)$

The RAPOOL engine checks if the dynamic user profile logical formula implies this formula. As the user profile is $functional(nurse) \wedge functional(night\_nurse) \wedge position(150, 45) \wedge hour(23)$. Implication holds, therefore access is granted.

4.3 Implementation in an object-oriented framework

The first approach to implement RAPOOL is to add a layer over an existing object-oriented language. Such a layer has to retrieve user profile and contextual information from role-assignment database. This layer needs to implement the profile and authorization policy transformation into logical formulae. This framework allows software designers to integrate access control in a declarative manner, without worrying about the mechanisms involved in authorization decision.

A proof-of-concept pre-processor RAPOOL to C++ has been implemented. For a developer, the RAPOOL to C++ pre-processor is a black-box transforming his code into C++. The basic steps of the pre-processor are:

- the input is a RAPOOL source file, as written by developers according to RAPOOL grammar,
- the pre-processor includes the C++ framework files to the source code,
- the pre-processor parse the header of RAPOOL (according to its grammar) and suppress it from the source file,
- the pre-processor analyse the body of RAPOOL and transform authorization policies into logical formulae (section 4.1) according to the previously parsed header,
- the pre-processor add a call to the authorization decision mechanism (included from C++ framework files) at the beginning of each method,
- the output is a C++ source file, obtained from RAPOOL source file once header and authorization policies are translated into calls to the framework.

4.4 Implementation in a role-based object-oriented model

In many class-based object-oriented systems the association between an instance and a class is exclusive and permanent. Therefore these systems have serious difficulties in representing dynamic evolution of objects over time. The problem is the most severe for OO databases in which objects are stored over long periods during which the entities evolve. Object-role oriented models intend to fill this shortcomings of object-oriented models by adding an orthogonal concept to classes: roles.

The authors of [23] describe an RBAC framework organized into 7 layers, as the OSI (Open Systems Interconnect) network stack is (from *physical* layer #1 to abstract *application* layer #7). The least abstract layer is the *object* layer, used by the directly higher one: *objects handles*. This second layer is used to keep the association between objects and roles. An object-role oriented model integrate directly such a layer: handles are no more needed, associations between roles and objects are handled in declarative manner in the object-role oriented paradigm.

The implementation of RAPOOL in a role-based object-oriented model is possible if:

- the model integrates a role hierarchy,
- the role hierarchy is independent of the class hierarchy,
- the model respects the Object Data Management Group (ODMG) standard (e.g. encapsulation, inheritance, polymorphism, etc.) to be compliant with an existing object-oriented language.

Section 5.2 survey previous works on integration of role mechanisms into object-oriented models. According to previous surveys [2, 10], Samovar is the most suitable model for hosting the RAPOOL language. This model respects the ODMG standard and integrate roles into object-oriented model. In this model, class can be seen as *abstract role containers*: no method is directly associated to a class. Attributes are only associated to roles or combinations of roles (including conjunction and disjunction). These combinations are expressed in first-order logic formulae (the same fragment of FOL used to express authorization policies in RAPOOL).

Thus, to implement RAPOOL in a role-based object-oriented model, we must carry a prior transformation step on access control policies. In RAPOOL, each authorization is associated to an object method, roughly said as “policies are organized by permissions". In order to implement our language, we must infer on these policies to organize them by roles, rather than by permissions. Note that this process can be performed automatically without human intervention.

For example, assuming we are working with the example from section 3.2. Policies of the `CElectronicPatientRecord` class are organized by permissions:

- permission `getPatientContact()` is granted to roles `(doctor or nurse)`,
- permission `getLastPrescription()` is granted to roles `(doctor or nurse)`,
- permission `getPrescriptionHistory()` is granted to roles `(doctor or (nurse and head))`.

In order to be implemented in a role-based object-oriented model, policies must be organized by roles. Thus, the above example will be organized as follows:

- role `(doctor)` is granted access to `getPatientContact()`, `getLastPrescription()` and `getPrescriptionHistory()`,
- role `(nurse)` is granted access to `getPatientContact()` and `getLastPrescription()`
- role `(nurse and head)` is granted access to `getPrescriptionHistory()`.

Everybody who is assigned to several roles is granted all permissions assigned to each role, as permitted by role hierarchy. Thus, `(nurse and head)` is also granted access which `(nurse)` is granted.

5 Related work

Very few work focused on integrating of access control models within logical data models. This section survey related work on introduction of security in object-oriented models and on the role-object oriented paradigm.

5.1 Integration of access control in object-oriented systems

Many papers have described how to implement security mechanisms involving roles and contexts (e.g. [16, 3]) in a role-oriented system. Our goal is not to describe how role-based access control can be implemented *with* classes, but is to describe how (and which subset of) role-based access control mechanisms can be implemented *in* classes.

Integration of access control into OO systems has already been studied. The authors of [24] describe how to implement Mandatory Access Control (MAC) in OO database systems. Roughly stated, MAC is a military-oriented model, in which users and resources are associated to labels. Access is granted if and only if the user label is as least as high as the requested resource label. Commonly used labels are *unclassified*, *confidential*, *secret* and *top-secret*. However, MAC has been shown to be too rigid for current applications, particularly when multiple users with different profiles are working on the system.

A more recent approach in [5] integrate MAC to UML diagrams. Their framework bridges the gap between software engineers and an organization security. A very interesting contribution which is not limited to class diagram and intends to integrate MAC in use cases and sequence diagrams. This paper describes a logic data model, but we are working on a conceptual model integrating role-based authorizations (section 6).

5.2 Extending object-oriented systems with roles

The object paradigm is a very expressive framework, largely used. According to [22], 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.

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* [23] (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.

The implementation of RBAC models in OO systems clearly points out that maintaining association between roles and classes can be a tough design challenge, particularly when dealing with role hierarchies. For example, [3, 4] describes a, UML class-diagram to implement RBAC. Their framework includes *role* and *role instance* classes. Thus, software designers have to implement a mechanism to ensure that an object instance of *role* is linked with another object instance of *role instance*.

A review of role-based object models in the programming and database areas can be found in [10, 2, 6]. However, these models are intended mainly to take into account the dynamic 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 (e.g. [25]).

5.3 Integration of security in role-oriented systems

To the best of our knowledge, the closest paper to our is [25]. This approach intends to integrate a subset of RBAC into a role-oriented system: DOOR. This model permits modelling an owner relationship between roles and objects, but this approach does not entirely respect the standard paradigms of object programming. Section 5.2 explains how RAPOOL can be implemented in role-object oriented systems, nevertheless in such implementation, roles can still be used for non role-based authorization purposes. The Samovar model [2] is well suited to include a basic form of RBAC because it includes role definition in a logical manner. These role formulae can easily capture permission-role assignments of RBAC.

The aspect-oriented paradigm can be seen as an alternative to the role-oriented one. Both of them aim at a more flexible use of objects in OO systems. The authors of [19] describe their approach based on an "aspect-oriented modelling (AOM) technique that allows system developers to isolate cross-cutting design structures in aspects to support controlled evolution of the structures". In this approach, design structures that represent access control policies are treated as aspects. Policies are integrated into system by merging aspects with the system design model in which acces control concerns are not adresses. This composition results in a *woven model*. This approach is interesting for its dynamic perspectives on access control modelling but do not consider role hierarchy. We chosed to exclude dynamic concerns in our approach to provide a less flexible but easier to use language. In our approach the basic subset of RBAC is incorporated directly, thus do not impose the composition of two models. We are investigating whether RAPOOL can be as easily implemented in an aspect-based model as it is in a role-based model.

6 Discussion

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

Security is often divided into confidentiality, reliability and integrity. Confidentiality is the least considered non-functional requirements of security. Access control models, and nowadays role-based ones, are designed to enhance confidentiality. Integration of Mandatory Access Control [5] in UML diagrams and security extensions for UML [12, 14] are promising. They will bridge the gap between a security will and its implementation. These conceptual and logical propositions can be core models for methodologies considering security as in integral part of the whole software design process such as [17]. We are currently working on automatic translation into RAPOOL of UML diagrams expressed in specific security models. RAPOOL 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 RAPOOL to C++ pre-processor, with the Foundstone SecureUML Visio template [1].

References

[1] R. Araujo and S. Gupta (2005) Design authorisation systems using SecureUML, *Foundstone*, Technical report.

[2] S. Coulondre and T. Libourel (2002) An integrated object-role oriented database model, *Data Knowl. Eng.*, 42(1):113–141.

[3] R. Crook, D. C. Ince, and B. Nuseibeh (2003) Modelling access policies using roles in requirements engineering, *Information & Software Technology*, 45(14):979–991.

[4] J. P. Davis and R. D. Bonnell (1999) Role-playing: A mechanism for bridging the object-oriented design-level gap, *OOPSLA-97: Workshop on Object Technology, Architectures and Domain Analysis*.

[5] T. Doan, S. Demurjian, T. C. Ting, and A. Ketterl (2004) Mac and uml for secure software design, *FMSE '04: Proceedings of the 2004 ACM workshop*

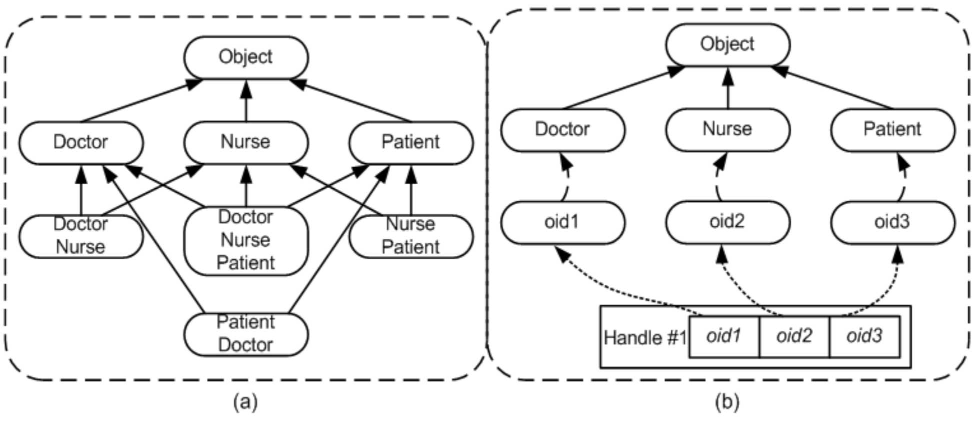

Figure 2: Empirical solutions for role implementation

on Formal methods in security engineering, ACM Press, 75–85.

[6] G. Gottlob, M. Schrefl, and B. Rock (1996) Extending object-oriented systems with roles, *ACM Trans. Inf. Syst.*, 14(3):268–296.

[7] S. Grumbach, P. Rigaux, and L. Segoufin (2001) Spatio-temporal data handling with constraints, *GeoInformatica*, 5(1):95–115.

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

[9] J. Jürjens (2002) UMLsec: Extending UML for secure systems development, *UML 2002 - The Unified Modeling Language. Model Engineering, Languages, Concepts, and Tools. 5th International Conference*, Springer, Dresden Germany, 412–425.

[10] G. Kappel, W. Retschitzegger, and W. Schwinger (1998) A comparison of role mechanisms in object-oriented modeling, *Modellierung CEUR Workshop Proceedings*, CEUR-WS.org.

[11] A. Kern, M. Kuhlmann, A. Schaad, and J. D. Moffett (2002) Observations on the role life-cycle in the context of enterprise security management, *SACMAT*, 43–51.

[12] D.-K. Kim, I. Ray, R. B. France, and N. Li (2004) Modeling role-based access control using parameterized uml models, *FASE*, Lecture Notes in Computer Science, 180–193.

[13] A. Kumar, N. Karnik, and G. Chafle (2002) Context sensitivity in role-based access control, *SIGOPS Oper. Syst. Rev.*, 36(3):53–66.

[14] T. Lodderstedt, D. A. Basin, and J. Doser (2002) Secureuml: A uml-based modeling language for model-driven security, *UML '02: Proceedings of the 5th International Conference on UML*, Springer-Verlag, London UK, 426–441.

[15] J. D. Moffett and E. Lupu (1999) The uses of role hierarchies in access control, *ACM Workshop on Role-Based Access Control*, 153–160.

[16] G. K. Mostéfaoui and J. Pasquier-Rocha (2003) Deterministic context-based security policies: An object-oriented approach, *SNPD*, ACIS, 160–165.

[17] H. Mouratidis, P. Giorgini, and G. A. Manson (2003) Integrating security and systems engineering: Towards the modelling of secure information systems, *CAiSE*, Lecture Notes in Computer Science, 63–78.

[18] S. L. Osborn, Y. Han, and J. Liu, (2003) A methodology for managing roles in legacy systems, *SACMAT*, ACM, 33–40.

[19] I. Ray, R. B. France, N. Li, and G. Georg (2004) An aspect-based approach to modeling access control concerns, *Information & Software Technology*, 46(9):575–587.

[20] H. Roeckle, G. Schimpf, and R. Weidinger (2000) Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization, *ACM Workshop on Role-Based Access Control*, 103–110.

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

[22] F. Steimann (2000) On the representation of roles in object-oriented and conceptual modelling, *Data Knowl. Eng.*, 35(1):83–106.

[23] D. Thomsen, D. O'Brien, and J. Bogle (1998) Role-based access control framework for network enterprises, *ACSAC '98: Proceedings of the 14th Annual Computer Security Applications Conference*, IEEE Computer Society, Washington DC USA.

[24] M. B. Thuraisingham (1989) Mandatory security in object-oriented database systems, *OOPSLA '89: Conference proceedings on Object-oriented programming systems, languages and applications*, ACM Press, New York USA, 203–210.

[25] R. K. Wong (1997) Rbac support in object-oriented role databases, *RBAC '97: Proceedings of the second ACM workshop on Role-based access control*, ACM Press, New York USA, 109–120.

Annex: BNF Grammar of RAPOOL header

Note: grammar of non–terminal symbol *logical_formula* is not included.

RAPOOL_header := groups_list
groups_list := groups_list group
|
group := group_identifier 'Roles' '{' definitions_list '}'
group_identifier := IDENTIFIER

definitions_list := definitions_list definition
|
definition := role_definition
| hierarchy_definition
| equivalency_definition

role_definition := 'Roles' ':' roles_list ';'
roles_list := role_identifier roles_list_suite
roles_list_suite := ',' role_identifier roles_list_next
|
role_identifier := IDENTIFIER

hierarchy_definition := 'Hierarchy' ':' hierarchical_relations_list ';'
hierarchical_relations_list := hierarchical_relation hierarchical_relations_list_suite
hierarchical_relations_list_suite := ',' hierarchical_relation hierarchical_relations_list_suite
|
hierarchical_relation := role_identifier '<<' role_identifier

equivalency_definition := equivalency_identifier ':=' logical_formula ';'
equivalency_identifier := IDENTIFIER