Representation and Reasoning on Role-Based Access Control Policies with Conceptual Graphs

Romuald Thion and Stéphane Coulondre

LIRIS: Lyon Research Center for Images and Intelligent Information Systems,
Bâtiment Blaise Pascal (501), 20, avenue Albert Einstein,
69621 Villeurbanne Cedex, France
prenom.nom@liris.cnrs.fr

Abstract. This paper focused on two aspects of access control: graphical representation and reasoning. Access control policies describe which operations on resources are granted to users. Role-based access control is the model which introduces the concept of role to design user' permissions. Actually, there is a lack of tools allowing security officers to describe and reason on their policies graphically. Thanks to conceptual graphs, we can provide a consistent graphical formalism for role-based access control policies, able to deal with specificities of this model such as role hierarchy and constraints. Moreover, once a policy modelled within CGs, graph rules and chainings can be used to reason on it. Thus, allowing SOs to understand why (through wich role assignment) user' permissions are granted and to find constraints violated by assignments.

1 Introduction

Opening our information systems to the world-wide web is really seducing, but highlights a problem which becomes more and more crucial: security. Nowadays, every on-line computer must be equipped with security update agent, firewall, anti-virus software and even anti-spyware, otherwise within a few minutes information system can become the target of automated scanning scripts, hackers, malicious websites, etc.: security is now of main importance. In this paper, we are dealing with a particular aspect of security process: access control.

1.1 Reasoning on access control policies

Access control denotes the fact of determining whether a *user* (process, computer, human user, etc.) is able to perform an *operation* (read, write, execute, delete, search, etc.) on an *ressource* (a tuple in a database, a table, an object, a file, etc.). An operation right on a ressource is called *permission*. Access control policies describes define the user' rights on ressources, in order to enforce security of an organization.

As a matter of fact, a large part of flaws in information systems are due to administration mistakes or security misconceptions: number of users is increasing, rules are more and more complex, constraints are introduced to deal with

border-line. As policies engineering is considered to be of high pratical importance [3], we do think there is a need for tools facilitating design and maintenance of security policies, to avoid mistakes and misconceptions.

Such tools need to meet several requirements:

- have an appropriate graphical interface,
- be able to capture access control model mechanisms and constrainted policies,
- be able to check consistency of policies,
- be able to answer queries for particular permissions or relation holdings in the policies,
- have comprehensible inference mechanism, even by non-logicians.

According to [4], we do think that such functionnalities cannot be designed without a proper formal framework.

1.2 Why conceptual graphs ?

Rather than tailoring a dedicated fragment of first-order logic [7] or using a pure logic-based approach [4, 3] with traditional resolution methods (such as Robinson principle), we focused on conceptual graphs (CGs) and their dedicated chaining of rules.

The CGs framework meets the foresaid requirements in section 1.1:

- CGs is a formal system (using the function Φ allow direct comparison with logic-based approaches),
- support can model complex structures such as hierarchies,
- individual markers, graph rules are expressive engough to capture authorisation mechanisms of access control models,
- chaining allow direct graphical-based inference on graphs.

This paper presents how to use the conceptual graphs framework to design and infer on role-based access control (RBAC) policies. CGs applied on RBAC allow graphical representation of policies, can capture underlying authorisation mechanisms and chaining of graph rules allows inference on policies.

2 Fundamentals

2.1 Role Based-Access Control Policies

Role based access control (RBAC) models constitute a family in which permissions are associated with roles (roles can be seen as collections of permissions), and users are made members of appropriate roles. The definition of role is quoted from [12]:

> A role is a job function or job title within the organization with some associated semantics regarding the authority and responsibility conferred on a member of the role.

Figure 1 is common representation of the RBAC model. In this figure, "URA" is a short for "user-role assignment" and "PRA" for "permission-role assignment". In the RBAC model, permissions are not directly assigned to users. Thus, a user is access granted to a ressource if he is assigned to a role which is assigned to this permission. The role hierarchy is a way to minimize the number of roles in RBAC policies: a role inheriting another one inherits all its permission.

As the major part of access control decisions is based on the subjects' functions or job, introducing roles greatly simplifies the management of the system. Since roles in an organization are relatively persistent with respect to user turnover and task reassignment, RBAC provides a powerful mechanism for reducing the complexity, cost, and potential for error in assigning permissions to users within the organization [1]. RBAC was found to be among the most attractive solutions for providing access control in e-commerce, e-government or e-health, and is also a very active research field.

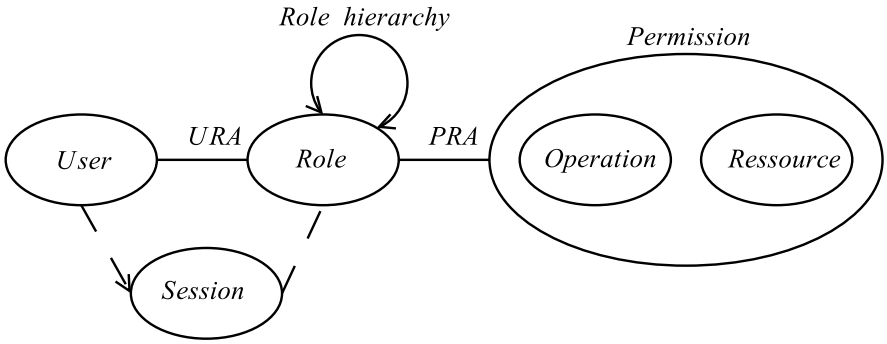

Fig. 1. RBAC Model (no specific formalism)

RBAC greatly simplifies the management of access control policies, however, organization may involve thousands of roles [10] and administration of RBAC policies can still be complex. Graphical based modelling and reasoning is a step beyond easy-to-use policy administration interface and better comprehension of policies.

2.2 Representation and reasoning based on conceptual graphs

!!!!Attention plagiat!!!![11, 6]

A rule $R : G_1 \Rightarrow G_2$ is composed of two CGs G_1 and G_2, respectively called *hypothesis* and *conclusion*. There may be co-reference links between concepts of G_1 and G_2. There are no constraints on types for individual markers common to G_1 and G_2: the type considered in operations is the smallest type that can be associated with the marker.

We call a *support* the structure that represents the ontology for a specific domain application, it is composed a concept type lattice and a relation type set. A *knowledge base* is composed of a support S, a set of simple CGs (facts), and a set of rules.

Forward chaining is typically used in order to enrich knowledge base with new facts, implicitly present in the base. It is used to prove a goal graph G_{goal} is implied by a knowledge base KB. Main idea is to calculate the closure of the set of facts of KB by the set of rules of KB. Basic outline of procedure is:

- chose a rule R from KB,
- let be G a fact from KB. If G fullfills the hypothesis of R, then the conclusion of R can be added to KB, if it is not already present (to avoid redundant informations),
- repeat until no new facts can be produced,
- if there exists a projection from G_{goal} to facts from KB, then G_{goal} is implied by the knowledge base.

This knowledge addition principle is the graph dual of bottom-up resolution approach in first-order logic.

Backward chaining is typically used to prove a goal (a request). Whereas forward chaining acts from facts to goal, backward chaining is top-down: from conclusion (G_{goal}) to hypothesis (the facts from KB). Basic outline of procedure is:

- find piece,
- produce subgoal,
- repeat until subgoal is empty.

3 Representation and reasonings on RBAC policies with conceptual graphs

In section 2, we described the role-based access control model and basic outlines of grph rules chaining. In this section we will present how a knowledge base can model a RBAC policy. A knowledge base is composed of a support S, a set of simple CGs (facts), and a set of rules. In this section, sample code are written in the *BCGCT* format for CGs.

3.1 Modelling basic concepts of RBAC

The support is composed a concept type lattice. For RBAC security modelling purposes, this lattice models the role-hierarchy. A user will be assigned to a role by defining an individual wich concept type is the assigned role. The main idea is: !!! A ecrire en langage math ? ca fait plus pro ??? :-)

- Add a *Role* concept type inherinting from the universal type,
- for each role defined within the policy (from the least privilegied ones to the most ones),
- create a concept type in support with same name,
- for each inheritance relation, add the equivalent edge in the concept type lattice. If no inheritance relation exist, then add an edge between current concept type and the Role concept type.

Figure 2 is a sample role hierarchy (based on french universities organisation). For example, a user assigned to role *Sr. Lecturer* will be granted every permissions granted to role *Lecturer*, and every permissions granted to both roles *Teacher* and *esearcher* by transitivity. For sake of readability, the universal concept and the absurd concept are not present.

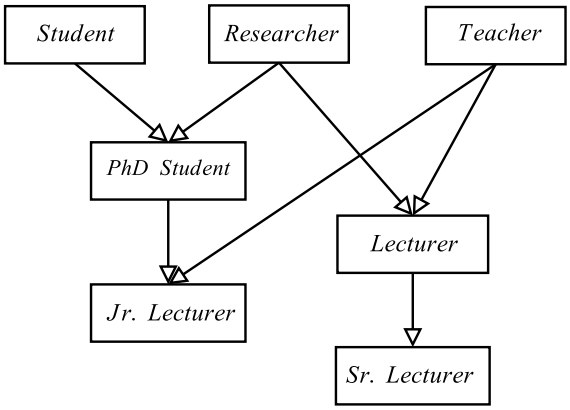

Fig. 2. A sample role hierarchy

After modelling role hierarchy, we can model other core elements of the RBAC model:

- add the concept type *Ressource* to the support,
- add the concept type *Operation* to the support,
- add the concept type *User* to the support,
- add the relation type *permitted* of arity 3:User,Operation,Ressource.
- for each user ressource the policy, add a conformity relation $ressource_id$: $Ressource$,
- for each operation within the policy, add a conformity relation $operation_id$: $Operation$,
- for each user within the policy, add a conformity relation $user_id : User$,

The following sample code is a support modelling a RBAC policy. Role hierarchy from figure 2 is translated into concept type lattice, three users are defined, three operations and two ressources:

```
{BCGCT:3}
Begin
Support:RBAC;
      TConSet:
            ConceptTypes:
                  Universal;

                  Role;
                  User;
                  Operation;
                  Ressource;

                  Student;
                  Researcher;
                  Teacher;
                  PhD_student;
                  Jr_Lecturer;
                  Lecturer;
                  Sr_Lecturer;
            EndConceptTypes;
            Order:
                  Student < Role;
                  Researcher < Role;
                  Teacher < Role;
                  PhD_student < Student;
                  PhD_student < Researcher;
                  Lecturer < Researcher;
                  Lecturer < Teacher;
                  Sr_Lecturer < Lecturer;
                  Jr_Lecturer < PhD_student;
                  Jr_Lecturer < Teacher;

                  Operation < Universal;
                  Ressource < Universal;
                  User < Universal;
            EndOrder;
      EndTConSet;
      TRelSet:
            RelationTypes:
                  permitted{Signature:3,User,Operation,Ressource};
            EndRelationTypes;
      EndTRelSet;
      Conf:
            romuald, User;
            robert, User;
            stephane, User;
            read, Operation;
            write, Operation;
            execute, Operation;
            examen, Ressource;
            projet, Ressource;
      EndConf;
EndSupport;
End
```

3.2 Modelling access control rules

```
Begin
        Rule: rEtudiant;
                Hypt: Graph: hypothesis;
                Concepts :
                      cUser=[Student:*];
                EndGraph;

                Conc: Graph: conclusion;
                Concepts :
                      cUserBis=[Student:*];
```

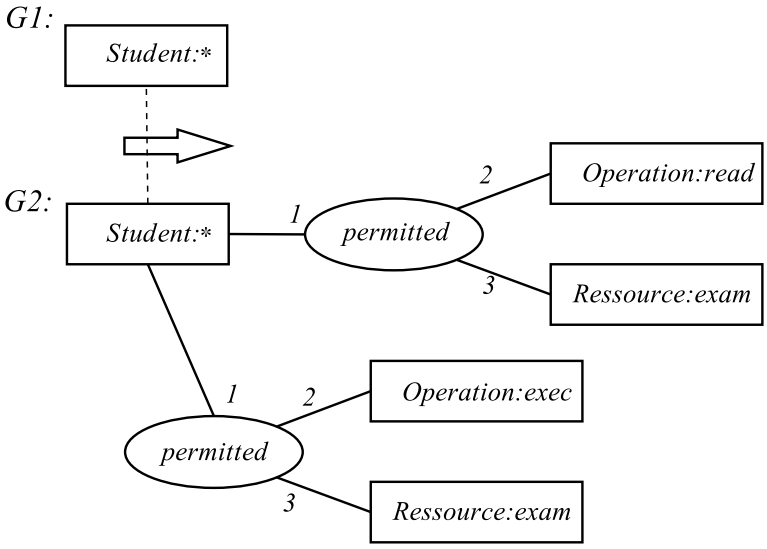

Fig. 3. A graph rule granting read and exec permission on exams to students

```
                cOperationR=[Operation:read];
                cOperationX=[Operation:execute];
                cRessource=[Ressource:examen];
        Relations :
                rPermittedR=(permitted);
                rPermittedX=(permitted);
        Edges :
                rPermittedR,cUserBis ,1;
                rPermittedR, cOperationR,2;
                rPermittedR, cRessource,3;
                rPermittedX,cUserBis ,1;
                rPermittedX, cOperationX,2;
                rPermittedX, cRessource,3;
        EndGraph;

        ConnectionPoints:
        (cUser,cUserBis);
EndRule;
End;
```

3.3 Reasoning on policies

```
Begin
        Graph : queryTest;
        Nature : fact;
        Concepts :
                c1=[Universal:*];
                c2=[Ressource];
                c3=[Operation];
        Relations :
                rPermit=(permitted);
        Edges :
                rPermit,c1,1;
```

```
            rPermit,c3,2;
            rPermit,c2,3;
      EndGraph;
End;
```

4 Conclusion

4.1 Graphical representation of Role-Based Access Control Policies

4.2 Reasoning on Role-Based Access Control Policies

sacmat

4.3 Representation of constraints

4.4 Requetage flou sur les bases de rles gigantesques

References

1. G.-J. Ahn and R. S. Sandhu. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Secur.*, 3(4):207–226, 2000.
2. S. Barker and P. J. Stuckey. Flexible access control policy specification with constraint logic programming. *ACM Trans. Inf. Syst. Secur.*, 6(4):501–546, 2003.
3. E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. *ACM Trans. Inf. Syst. Secur.*, 6(1):71–127, 2003.
4. P. A. Bonatti and P. Samarati. Logics for authorization and security. In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases*, pages 277–323. Springer, 2003.
5. M. Chein and M.-L. Mugnier Représenter des connaissances et raisonner avec des graphes *Revue d'Intelligence Artificielle*, 10(1):7–56, 1996.
6. S. Coulondre and E. Salvat. Piece resolution: Towards larger perspectives. In M.-L. Mugnier and M. Chein, editors, *ICCS*, volume 1453 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 1998.
7. J. Y. Halpern and V. Weissman. Using first-order logic to reason about policies. In *CSFW*, pages 187–201. IEEE Computer Society, 2003.
8. S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26(2):214–260, 2001.
9. M. Koch and F. Parisi-Presicce. Visual specifications of policies and their verification. In M. Pezzè, editor, *FASE*, volume 2621 of *Lecture Notes in Computer Science*, pages 278–293. Springer, 2003.
10. H. Roeckle, G. Schimpf, and R. Weidinger. Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization. In *ACM Workshop on Role-Based Access Control*, pages 103–110, 2000.
11. E. Salvat and M.-L. Mugnier. Sound and complete forward and backward chaining of graph rules. In P. W. Eklund, G. Ellis, and G. Mann, editors, *ICCS*, volume 1115 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 1996.
12. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

13. R. S. Sandhu and Q. Munawer. The arbac99 model for administration of roles. In *ACSAC*, pages 229–. IEEE Computer Society, 1999.
14. K. Sohr, G.-J. Ahn, M. Gogolla, and L. Migge. Specification and validation of authorisation constraints using uml and ocl. In S. D. C. di Vimercati, P. F. Syverson, and D. Gollmann, editors, *ESORICS*, volume 3679 of *Lecture Notes in Computer Science*, pages 64–79. Springer, 2005.