MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE
LA RECHERCHE SCIENTIFIQUE
UNIVERSITE ABDELHAMID IBN BADIS - MOSTAGANEM

Faculté des Sciences Exactes et de l'Informatique
Département de Mathématiques et d'Informatique
Filière : Informatique

MEMOIRE DE FIN D'ETUDES
Pour l'Obtention du Diplôme de Master en Informatique
Option : Ingénierie des Systèmes d'Information

THEME :

Profils UML de spécification de politiques RBAC

« Représentation graphique et cohérence »

Etudiant : **CHAOUCHE Ahmed Zakaria**

Encadrant : CHEHIDA Salim

Année Universitaire 2016/2017

Résumé

Dans ces dernières années, la définition de politiques de contrôle d'accès dès les premières étapes de cycle de développement logiciel est une préoccupation majeure. Le modèle RBAC (Role-Based Access Control) est le modèle de contrôle d'accès le plus adopté par les chercheurs et les industriels.

UML est un langage standard de l'OMG utilisé pour l'analyse et la conception d'un système. Ce langage présente l'avantage de pouvoir se décliner sous forme de « profils » permettant l'extension de ces diagrammes pour spécifier un aspect particulier d'un système.

Plusieurs travaux ont proposé des profils UML utiles pour la spécification des politiques de contrôle d'accès basées sur le modèle RBAC. Parmi ces profils nous pouvons citer :

- ✓ SecureUML : qui étend le diagramme de classes.
- ✓ BAAC@UML : qui étend le diagramme d'activités.

Ce projet vise à étudier les profils UML de spécification de politiques RBAC notamment SecureUML et BAAC@UML . Nous proposons également la mise en œuvre de ces profils dans outil permettant la représentation graphique de leurs modèles et la vérification de leur cohérence

Abstract

In recent years, the definition of access control policies from the early stages of the software development cycle is a major concern. The RBAC (Role-Based Access Control) model is the most adopted access control model.

UML is an OMG standard language used for analysis and design of a system. This language proposes "profile" mechanism witch allowing the extension of these diagrams to specify a particular aspect of a system.

Several works have proposed profiles useful for specifying access control policies based on the RBAC model. Among these profiles, we can cite:

- ✓ SecureUML : Which extends the class diagram .
- ✓ BAAC@UML : Which extends the activity diagram .

This project deals the SecureUML and BAAC@UML profiles. We propose tool allowing the graphical design of SecureUML and BAAC@UML models and the detection of the inconsistencies of these models.

Mots-clés : Contrôle d'accès, RBAC, UML, Profil, Cohérence.

Remerciement

*En premier lieu, je remercie **ALLAH** de m'avoir donné la force pour la réalisation de ce projet de fin d'étude.*

Au terme de la rédaction de ce mémoire, je tiens à remercier mon encadreur monsieur CHEHIDA SALIM pour l'orientation, la confiance, la patience qui ont constitué un apport considérable sans lequel ce travail n'aurait pas pu être mené au bon port. Qu'il trouve dans ce travail un hommage vivant à sa haute personnalité.

Je tiens à remercier de tout cœur mes parents qui m'ont toujours soutenus dans cette aventure et n'ont reculés devant aucune épreuve. Je leur serai toujours reconnaissante pour tout ce qu'ils ont bien voulu faire de moi. Je salue ici toutes leurs déterminations, leurs efforts et leurs sens du sacrifice.

Mes vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à notre humble PFE, en acceptant de l'examiner et de l'enrichir par leurs remarques.

Je tiens à exprimer mes sincères remerciements à tous les professeurs de l'université ABD ELHAMID IBN BADIS, FSEI qui m'ont enseigné et qui par leurs compétences j'ai soutenu dans la poursuite de mes études.

Enfin, je remercie tous ceux qui, de près ou de loin, ont contribué à la réalisation de ce travail.

Merci à toutes et à tous.

Dédicace

L'eau coule grâce à sa source
L'arbre pousse grâce à ses racines

A Mes parents

Pour les sacrifices déployés à mes égards ; pour leur patience Leur amour et leur confiance en moi
Ils ont tout fait pour mon bonheur et ma réussite.
Qu'ils trouvent dans ce modeste travail, le témoignage de ma
Profonde affection et de mon attachement indéfectible.
Nulle dédicace ne puisse exprimer ce que je leur dois
Que dieu leur réserve la bonne santé et une longue vie.

A mes sœurs

En témoignage de mon affection fraternelle, de ma profonde tendresse et reconnaissance, je leur souhaite une vie pleine de bonheur et de succès et que Dieu, le tout puissant, les protège et les garde pour moi.

A mes grands parents

Je leur dédie cette thèse en témoignage de gratitude d'estime et d'attachement. Puisse dieu leur accorder santé, longue vie et prospérité

A mes oncles et tantes

Veuillez trouver dans ce travail un modeste témoignage de mon admiration et toute ma gratitude, de mon affection la plus sincère et de mon remerciement le plus profond pour votre hospitalité et votre générosité.

A Nos amis

En témoignage de mes sincères reconnaissances pour les efforts
Qu'ils ont consentis pour me soutenir au cours de mes études.
Que dieu nous garde toujours unis

A Toute personne qui m'a aidé à faire mon projet.

Table de matières

Liste des Figures

Liste des Abréviations

UML: Unified Modeling Language

OMG: Object Management Group

RBAC: Role-Based Access Control

OCL: Object Constraint Language

DAC: Discretionary Access Control

MAC: Mandatory access control

OPS: Operation

PRMS: Permission

PA : Permission Assignment

UA : User Assignment

OBS : Objet

DSD: Static Separation of Duty

SSD: Dynamic Separation of Duties

EJB : Entreprise Java Beans

RH : hiérarchique de rôles

GACPreCondition : Global Acces Control PreCondition

BAAC@UML: Business Activity Access Control with UML

Introduction Générale

Aujourd'hui, la sécurité est un enjeu majeur pour les entreprises ainsi que pour l'ensemble des acteurs qui l'entourent. Elle n'est plus confinée uniquement au rôle de l'informaticien. Sa finalité sur le long terme est de maintenir la confiance des utilisateurs et des clients. Sur le court terme, l'objectif est que chacun ait accès aux informations dont il a besoin. [18]

L'une des mesures importantes de la sécurité consiste à contrôler les accès possibles au système et à autoriser ou non un certain nombre d'action en fonction de l'utilisateur. Il s'agit de limiter l'accès aux ressources du système d'information uniquement aux utilisateurs, programmes, procédés ou systèmes autorisés.

Plusieurs types de modèles de contrôle d'acces ont été proposées , comme par exemple , DAC (Discretionary Access Control)[1] , MAC (Mandatory Access Control) [1], et RBAC (Role Based Access Control) [1]. Dans ce dernier, chaque utilisateur est assigné à un rôle qui lui permet d'accéder à des permissions sur des ressources.

Toutefois, il est indispensable de considérer le contrôle d'accès dès les premières phases du cycle de développement d'un système. La notation UML [9] est un langage standard de l'OMG [4] constitué d'un ensemble de diagrammes, qui donnent chacun une vision différente du système. Cependant, les diagrammes d'UML restent incapables de représenter des aspects particuliers du système comme les aspects de sécurité. Le concept de Profil propose des mécanismes permettant de spécialiser les diagrammes d'UML. Plusieurs travaux ont proposé des profils utiles pour la spécification des politiques de contrôle d'accès basées sur le modèle RBAC. Parmi ces profils nous pouvons citer UMLsec [11], SecureUML [14] et BAAC@UML [16].

Notre projet vise à étudier ces différentes extensions d'UML. Nous présentons d'abord le modèle RBAC dans le premier chapitre. Dans le second chapitre, nous définissons les différentes extensions de diagrammes UML pour la spécification de politiques RBAC notamment le profil SecureUML qui exprime la vue statique d'une politique RBAC et le profil BAAC@UML qui exprime la vue dynamique d'une politique RBAC. Dans le troisième chapitre, nous allons traiter la cohérence des modèles SecureUML et BAAC@UML. Nous réalisons un outil qui permet la vérification de la conformité de ces modèles par rapport à leurs méta-modèles.

Chapitre 1

Le modèle RBAC

1. Introduction

Le contrôle d'accès est utilisé pour limiter les possibilités d'utilisation du système d'information. Les concepts de base de contrôle d'accès sont :

-Sujets : entités actives (utilisateurs ou processus) qui demandent des droits d'accès au système.
-Objets : entités passives du systèmequireprésente les informations ou les ressources à protéger.
- Actions : moyens permettant aux sujets de manipuler les objets du système.

L'objectif du contrôle d'accèsestde préserver la confidentialité, l'intégrité et la disponibilité des données. Plusieurs modèles de contrôle d'accès ont été proposés. Dans ce chapitre, nous présentons le modèle RBAC (RoleBased Access Control) [1] qui est le modèle le plus répondu pour contrôler l'accès aux systèmes.

2. Présentation générale du modèle RBAC

Dans le modèle RBAC, chaque décision d'accès est basée sur le rôle auquel l'utilisateur est attaché. Un rôle découle généralement de la structure d'une entreprise. Les utilisateurs exerçant des fonctions similaires peuvent être regroupés sous le même rôle. Un rôle associe à un utilisateur des autorisations d'accès sur un ensemble d'objets [1].Le changement des règles de contrôle d'accès n'est pas nécessaire chaque fois qu'un utilisateur rejoint ou quitte une organisation. Par cette caractéristique, RBAC est considéré comme un système idéal pour les entreprises dont la fréquence de changement du personnel est élevée.

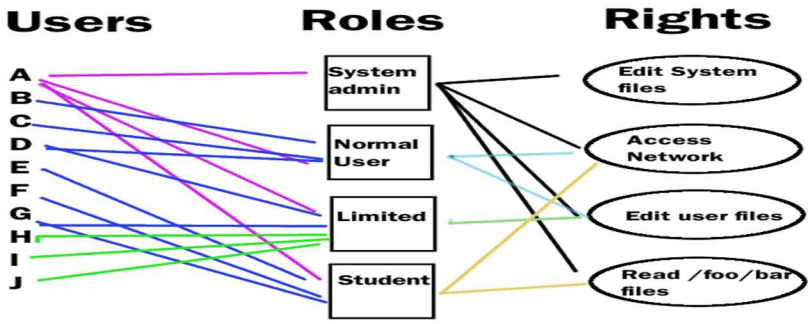


Figure1 : Représentation de la relation entre les rôles et les utilisateurs [3]

Comme montre la figure 1, au sein d'une organisation, des rôles sont crées pour diverses fonctions. Les autorisations d'accès sont affectées à ces rôles. Les utilisateurs du système sont affectés à un ensemble de rôles et, par l'intermédiaire de ces affectations, ils acquièrent des autorisations pour effectuer certaines opérations. La gestion des droits d'accès de l'utilisateur devient une simple attribution de rôles appropriés à l'utilisateur. Cela simplifie les opérations courantes, comme l'ajout d'un utilisateur ou le changement du service de l'utilisateur. Le modèle de référence RBAC est organisé en trois niveaux qui sont le noyau, la hiérarchie de rôles et les contraintes de rôles.

3. Noyau RBAC

La figure 2 présente le noyau du modèle RBAC qui englobe les éléments de base du modèle et les relations entre ces éléments. Ce niveau est défini par les concepts utilisateur (USERS), rôle (ROLES), session (SESSIONS), permission (PRMS), opération (OPS) et objet (OBS) ainsi que les relations PA (Permission Assignment), UA (User Assignment), session_users et session_roles.

- Un utilisateur est défini comme un être humain. Bien que le concept d'utilisateur puisse être étendu à des machines, des réseaux… [3]
- Un rôle est une fonction dans le contexte d'une organisation avec une certaine sémantique associée concernant l'autorité et la responsabilité conférée aux utilisateurs affectés à cette fonction.
- Les objets représentent des conteneurs d'informations (par exemple : des fichiers, des répertoires, dans un système d'exploitation ou des lignes, des tables et des vues dans un système de gestion de base de données).
- Une permission est une approbation à l'utilisateur pour effectuer une opération sur un ou plusieurs objets protégés du système. [3]
- Une opération est action invoquée par un utilisateur sur les objets du système.les opérations peuvent inclure lire, écrire et exécuter. Dans un système de gestion de base de données, les opérations peuvent inclure insérer, supprimer, ajouter et mettre à jour, etc.
- UA est une relation qui permet d'affecter des utilisateurs aux rôles (UA $\subseteq$ USERS $\times$ ROLES).

  Formellement: Utilisateur_affecté (r) = {u Є USERS / (u, r) Є UA}.
- PA (Permission Assignment) : exprime l'affectation des permissions aux rôles. Un seul rôle peut bénéficier de nombreuses permissions.
- **Formellement** : Permission_affectée (r) = {p Є PRMS / (p, r) Є PA}.
- Session : Session représente une instance de la boîte de dialogue d'un utilisateur avec le système. Un utilisateur peut créer ou supprimer une session et activer ou désactiver un rôle dans une session. Plusieurs rôles peuvent être activés au sein d'une même session [3] et un utilisateur peut avoir exécuté plusieurs sessions simultanément.
- sessions-rôle nous donne les rôles activés dans une la session.
  Formellement : session- rôles (s_i) = {r Є ROLES / (session_ utilisateur (s_i), r) Є UA}.
- sessions-user est une relation qui nous donne l'ensemble des sessions associées à un utilisateur.
  Formellement : Avail_Session_Perms(s) = $\bigcup_{r \in \text{session\_ rôles (s)}}$ Permission_ affecté (r).
  - Avail_Session_Perms(s)==> les permissions qui sont disponible dans une session.
  - U Permission_ affecté (r)==> tous les permissions qui ont été affecté avec des rôles dans cette session.

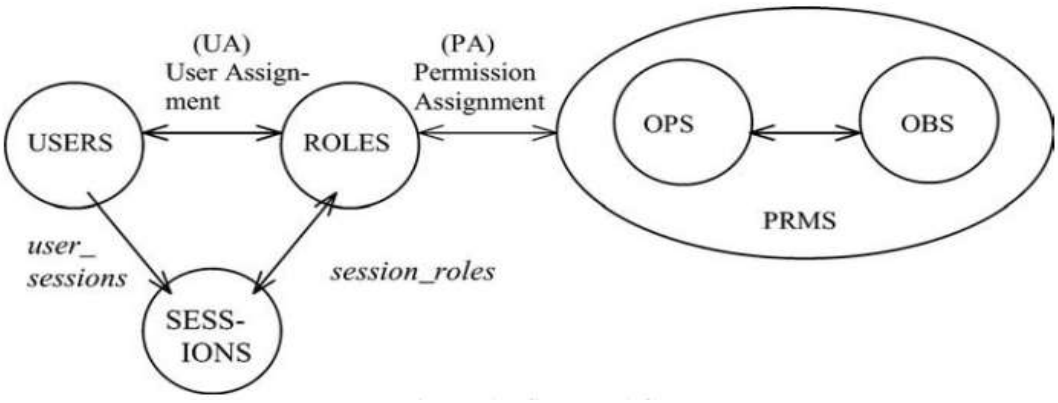


Figure 2 : core RBAC [3]

Dans RBAC, les rôles sont assignés à des utilisateurs, les permissions sont assignées à des rôles, et les utilisateurs acquièrent des permissions en étant membres des rôles. Le noyau du RBAC inclut la condition que l'affectation utilisateur-rôle et permission-rôle peut être many to many (n à n). Ainsi, le même utilisateur peut se voir assigner plusieurs rôles et un rôle peut être attribué à plusieurs utilisateurs. De même, pour les permissions, une seule permission peut être assignée à plusieurs rôles, et un seul rôle peut être assigné à plusieurs permissions.

4. La relation hiérarchique de rôles

Ce niveau ajoute la relation hiérarchique (RH) entre les rôles au noyau du modèle RBAC comme le montre la figure 3. Il existe deux types de hiérarchie de rôles : générale et limité.

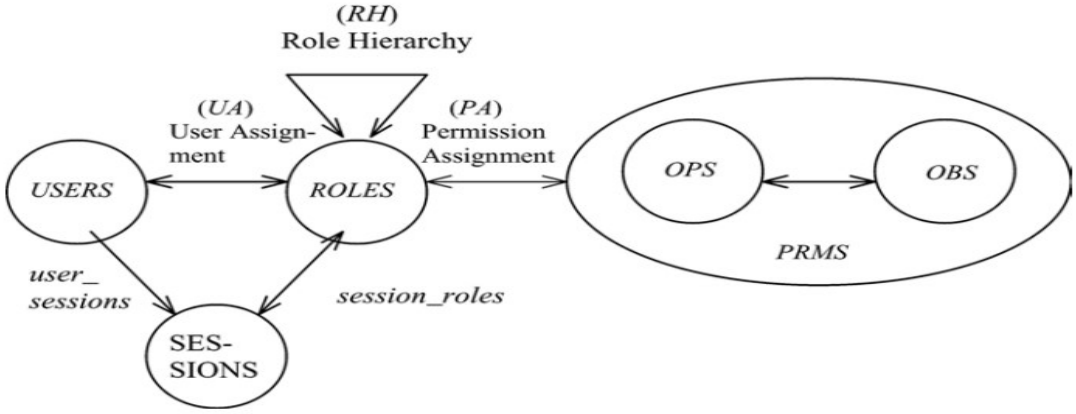


Figure 3 : Hiérarchie RBAC [3]

4.1 Hiérarchie générale

Les hiérarchies générales de rôle soutiennent le concept d'héritage multiple qui permet à un rôle d'hériter des permissions de deux ou plusieurs rôles ou d'héritent de bénéficier de l'appartenance de ses utilisateur à deux ou plusieurs rôles.

Formellement

RH $\subseteq$ ROLES x ROLES

Soit l'ordre partiel $\geq$ tel que r1 $\geq$ r2:

[auth_permis (r2) $\subset$ auth_permis (r1)]

[auth_user (r1) $\subset$ auth_user (r2)]

Tel que:

auth_user (r) = {u $\in$ USERS / r' >r, (u, r') $\in$ UA}

auth_permis (r) = {p $\in$ PRMS / r' >r, (p, r') $\in$ PA}

La représentation graphique des hiérarchies de rôles se fait par des structures comme le montrent les figures 4, 5 et 6.

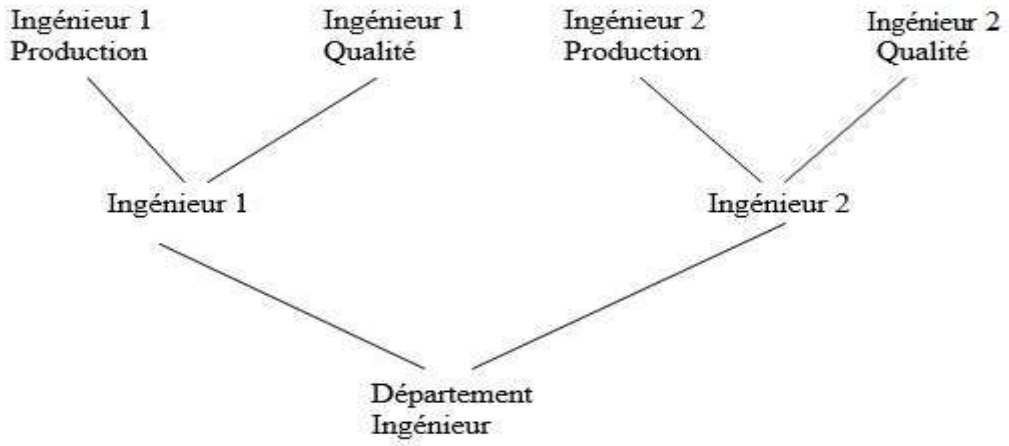


Figure 4 :Arbre Inversé

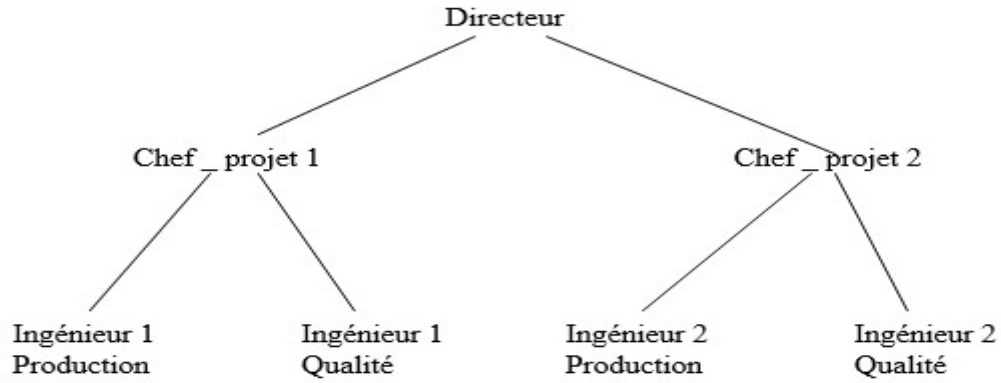


Figure 5 : Arbre

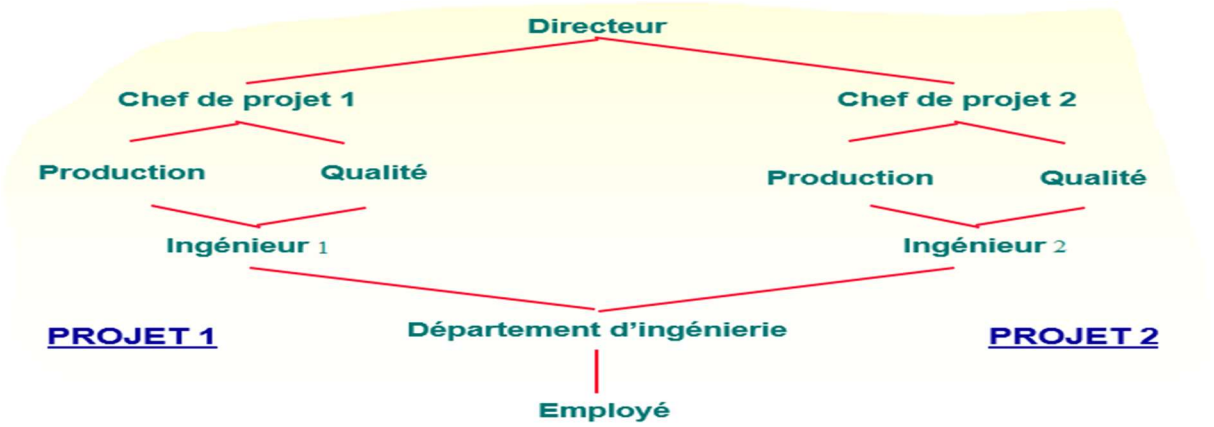


Figure 6 : Treillis.

Dans les organisations, les patrons ont plus de pouvoir que les subordonnés (ils héritent leurs permissions). Comme le montre la figure 5, le chef du projet hérite les permissions des ingenieur de production et des ingenieur de qualité, et le directeur hérite les permissions des deux chefs de projets .
Si l'ingenieur de production 1 a une permission, alors le Chef de projet 1 et le Directeur l'ont aussi .[5]

4.2. hiérarchie limitée

Les rôles dans une hiérarchie de rôle limitée sont limités à un seul descendant immédiat. Ce type ne prend pas en charge l'héritage multiple.

Formellement

$\forall r, r1, r2 \in ROLES, r \geq r1 \wedge r \geq r2 \Rightarrow r1 = r2$

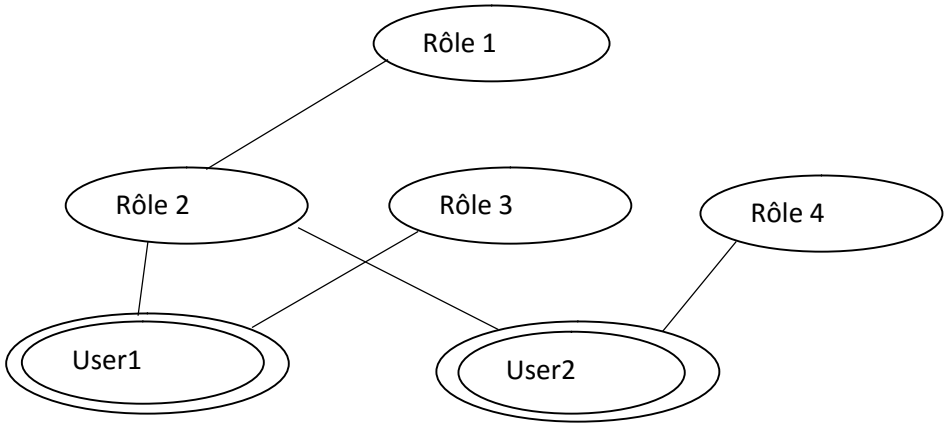


Figure 7 : exemple d'une hiérarchie limitée

Le rôle R1 hérite seulement du rôle R2 => hiérarchie limité.
Le rôle R1 ne peut pas hériter des rôles R3 et R4 en plus de R2, sinon => hiérarchie générale.

5. *Les contraintes de rôles*

Ces contraintes sont utilisées pour éviter des conflits d'intérêt dans les organisations. Elles permettent d'empêcher les utilisateurs de dépasser un niveau raisonnable de pouvoir [8]. Il existe deux principaux types de contraintes : la séparation statique des devoirs (figure 5) et la séparation dynamique des devoirs (figure 6).

5.1. La Séparation statique des devoirs (SSD)

La contrainte de séparation statique des devoirs (SSD) vise à empêcher leconflit d'inté rêts qui apparaît lorsque des rôles en conflit sont associés à un même utilisateur.Le conflit d'i ntérêts dans un rôle peut surgir comme le résultat d'un utilisateur obtenant une autorisation pour des permissions liées avec les rôles en conflit. [7]

exemple :

l'admin crée les rôles comptables et contrôleur, ils sont indépendants .

l'administrateur donne au rôle comptable la permission d'émettre des chèques.

l'administrateur donne au rôle contrôleur la permission de contrôler les chèques.

l'administrateur décide que Alice a les deux rôles.

- ? conflit d'intérets [6]

Il y a malheureusement plusieurs manières pour arriver à une situation de conflit.

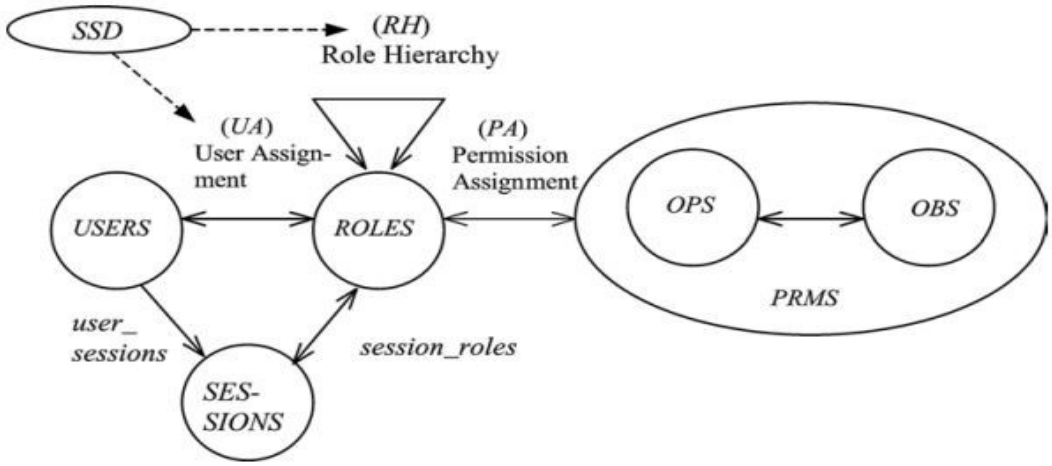


Figure 8 : Séparation statique des devoirs [3]

Formellement :

SSD⊆ (2ROLES×N) est un ensemble de paires (rs, n) en Statique Séparation de droit, où chaque rs est un ensemble de rôles, n est un nombre naturel ≥ 2, avec la propriété qu'aucun utilisateur n'est affecté à n ou plus de rôles à partir de l'ensemble rs dans chaque (rs, n) ∈SSD.

la séparation statique des devoirs peut être définie en absence et en présence d'une hiérarchie.

Absence d'une hiérarchie

$$\forall (r_s, n) \in SSD,\ t \subset r_s :\ |t| \geq n \Rightarrow \bigcap_{r \in t} \text{Utilisateur\_affecté}(r) = \emptyset.$$

Présence d'une hiérarchie

$$\forall (r_s, n) \in SSD,\ t \subset r_s :\ |t| \geq n \Rightarrow \bigcap_{r \in t} \text{auth\_user}(r) = \emptyset.$$

5.2 La Séparation dynamique des devoirs (DSD)

Les relations Dynamic Separation of duty (DSD), comme les relations SSD, visent à limiter les autorisations qui sont disponibles pour un utilisateur. Cependant, les relations DSD diffèrent des relations SSD par le contexte dans lequel ces limitations sont imposées. Les relations SSD définissent des contraintes sur l'espace d'autorisation total d'un utilisateur .Par contre le DSDdéfinit des contraintes sur les rôles activés par l'utilisateur dans le cadre d'une session[3].

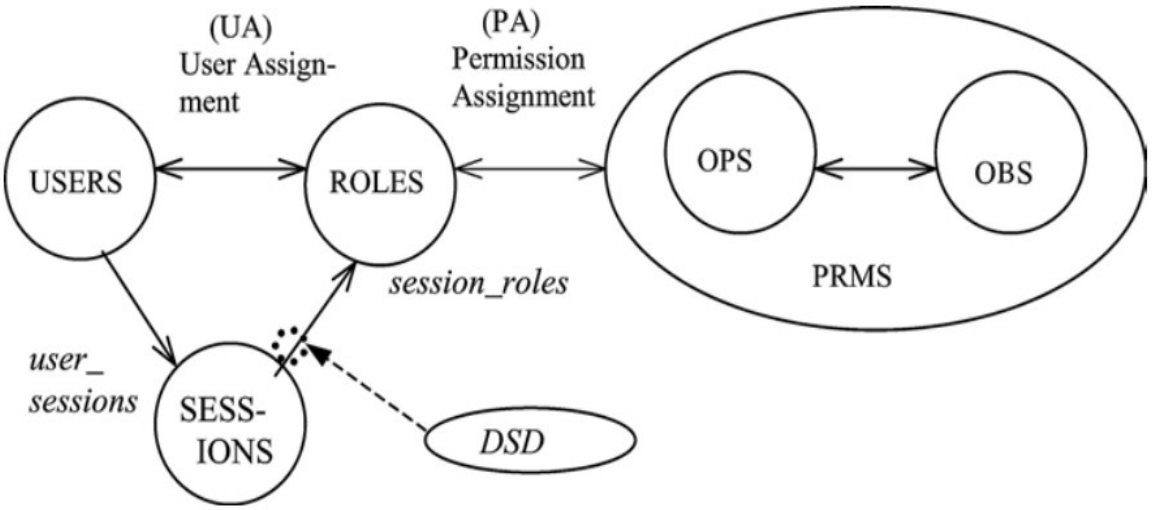


Figure 9 : La Séparation dynamique des devoirs [3]

DSD⊆ (2ROLES×N) est la collection de paires (rs, n), où chaque rs est un ensemble de rôles, n est un nombre naturel ≥2, Avec la propriété qu'aucun user-session ne peut activer n ou plus de rôles de l'ensemble rs dans chaque dsd ∈DSD.

Formellement

$\forall r_s \subset ROLES, n \in N, (r_s, n) \in DSD \Rightarrow n \geq 2 \wedge |r_s| \geq n$, et

$\forall s \in SESSION, \forall r_s \in ROLES, \forall Sous\_Role \subset ROLES,$

$\forall DSD, Sous\_Role \subset r_s, Sous\_Role \subset \text{session- rôles}(s) \Rightarrow |Sous\_Role| < n.$

6. Conclusion

Le modèle RBAC, est devenu le modèle de contrôle d'accès le plus employé. Il est supposé fournir une structure viable pour adresser à grande échelle les besoins de contrôle d'accès pour un très grand nombre d'entreprise.

Dans ce chapitre nous avons présenté le modèle RBAC, son fonctionnement et de l'importance à respecter les différentes notions qui le compose afin d'optimiser la sécurité des systèmes. Une bonne configuration et une bonne gestion des rôles nous permettront d'éviter bien des problèmes de sécurité.

Chapitre 2

Extensions d'UML pour la spécification de politiques RBAC

1. Introduction

UML (Unified Modeling Language) [9] est un langage standard de l'OMG [10] utilisé pour l'analyse et la conception d'un système. Ce langage présente l'avantage de pouvoir se décliner sous forme de « profils » permettant l'extension de ces diagrammes pour spécifier un aspect particulier d'un système.[9] Plusieurs travaux ont proposé des profils utiles pour la spécification des politiques de contrôle d'accès basées sur le modèle RBAC. Dans ce chapitre, nous présentons brièvement les différents diagrammes d'UML puis nous décrivons des extensions de certains de ces diagrammes pour la spécification de politiques RBAC.

2. Les diagrammes UML 2

UML est une notation graphique destinée à la modélisation orienté objet du système. La version 2.0 comporte ainsi plusieurs types de diagrammes représentant autant de *vues* distinctes du système d'information. Ils se répartissent en deux grands groupes :

2.1 *Diagrammes structurels* [9]

- Diagramme de classes : le but d'un diagramme de classes est d'exprimer de manière générale la structure statique d'un système en termes de classes et de relations entre ces classes. Une classe a des attributs, des opérations et des relations avec d'autres classes.
- Le diagramme d'objet : il sert à représenter les instances de classes (objets). Il montre également les liens entre ces objets.
- Diagramme de composants : il montre les composants du système du point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques,..). Il permet également de mettre en évidence les dépendances entre les composants.
- Diagramme de déploiement : il montre la disposition physique du matériel qui compose le système (ordinateurs, périphériques, réseaux...) et la répartition des composants logiciels sur ces matériels. Les ressources matérielles sont représentées sous forme de nœuds, connectés par des supports de communication.
- Diagramme des paquetages : un paquetage est un conteneur logique permettant de regrouper et d'organiser les éléments des modèles UML. Il sert également à représenter les dépendances entre ces paquetages.
- Diagramme de structure composite : est un ensemble d'éléments interconnectés collaborant lors de l'exécution d'une tâche. Il est représenté par un ensemble de pièces (rôles) liées par des connecteurs.

2.2 *Diagrammes comportementaux* [9]

- Diagramme des cas d'utilisation : il permet à représenter les besoins des utilisateurs du système. Il montre les relations entre les acteurs et les cas d'utilisation du système.
- Diagramme d'activité : il permet de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation. L'élément principal de ce diagramme est l'activité. Une activité est un comportement qui peut être représenté par un

enchainement d'étapes. Le comportement d'une activité est modélisé par des nœuds (nœuds d'activité) reliés par des arcs (transitions).

- Diagramme états-transitions : il permet de décrire sous forme de machine à états finis des comportements du système ou de ses composants. Il est composé d'un ensemble d'états, reliés par des arcs orientés qui décrivent les transitions.
- Diagramme de séquence : Il représente séquentiellement le déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs. Il sert à modéliser les aspects dynamiques du système et des scénarios complexes. Dans ce type de diagramme, l'accent est mis sur des envois de messages.
- Diagramme de communication : c'est une représentation simplifiée d'un diagramme de séquence, en se concentrant sur les échanges de messages entre les objets.
- Diagramme global d'interaction : il fournit une vue générale des interactions décrites dans le diagramme de séquence et des flots de contrôle décrits dans le diagramme d'activités.
- Diagramme de temps : il permet de présenter l'interaction entre les objets actifs et leurs changements d'état sur un axe de temps. Il décrit également les variations d'une donnée au cours du temps.

3. La notion du profil UML

UML fournit des mécanismes permettant aux concepteurs d'étendre les diagrammes d'UML pour traiter des aspects particuliers d'un système. Les extensions sont organisées en profils. Un profil est un ensemble cohérent d'extensions applicables à un domaine. Il permet de spécialiser le méta-modèle UML [11].L'OMG définit trois mécanismes d'extension : les stéréotypes, les étiquettes et les contraintes.

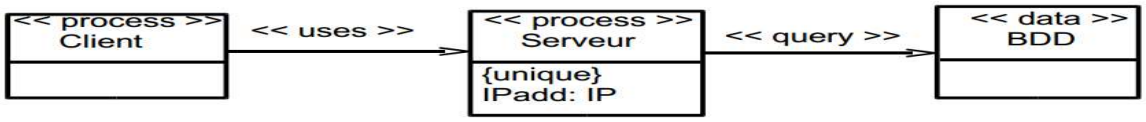
3.1 Stéréotype

Les stéréotypes permettent aux concepteurs d'étendre les éléments des diagrammes d'UML.Un stéréotype introduit une nouvelle classe dans le méta-modèle par dérivation d'une classe existante. Comme le montre la figure 10, un stéréotype est rendu comme un nom encadré par des guillemets <<>>et placé au-dessus du nom de l'élément stéréotypé.

Figure 10 : stéréotype

3.2 Etiquette (valeur marquée)

Une valeur marquée est un méta-attribut que l'on définit comme attribut d'un stéréotype. Elle ajoute une nouvelle propriété à un élément de modélisation. Une valeur marquée a un nom et un type indiqué entre accolades**.**



Valeur marquée : {unique} id : int

Figure 11 : valeur marquée

3.3 Une contrainte

Dans UML2, une contrainte ajoute une sémantique à un stéréotype. Pour exprimer des contraintes de manière formelle, le langage de contraintes OCL (*Object Constraint Language*) peut être utilisé. Chaque contrainte est indiquée entre accolades {}. Une contrainte peut être associée à plusieurs éléments par des relations de dépendance **[**12**]**. La figure 12 présente un exemple d'une contrainte associée au stéréotype *Bean*.

La figure 12définit un profil UML spécifiant une architecture EJB (Entreprise JavaBeans). Ce profil regroupe les cinq stéréotypes (Bean, Session, Entité, Jar, Eloigné, Accueil). Les flèches noires représentent des extensions. Enumération est un type de données dont les instances proviennent d'une liste de valeurs littérales nommées.

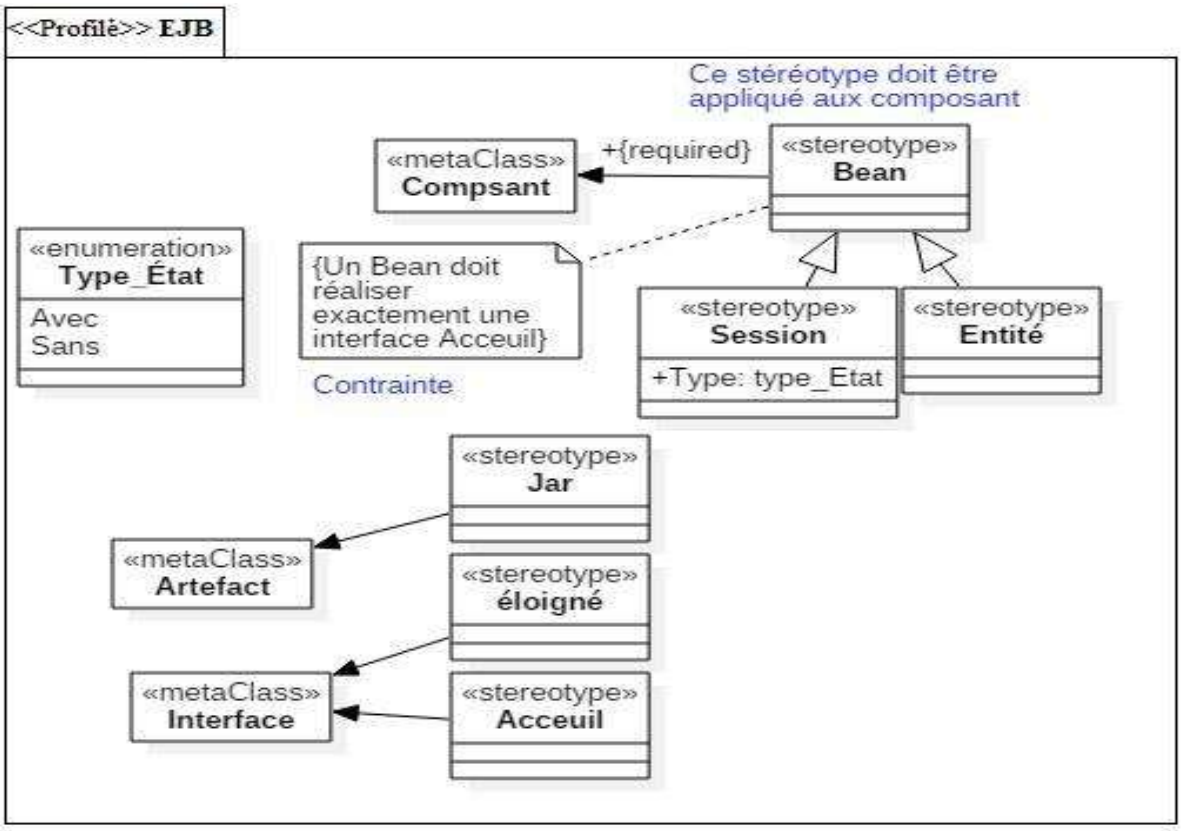


Figure 12 : Un exemple du profil UML qui utilise une contrainte OCL. [19]

4. Les profils RBAC

Dans cette section, nous présentons trois profils UML de spécification de politiques RBAC. SecureUML qui présente un profil pour étendre le diagramme de classe, UMLsec qui présente un profil pour étendre le diagramme d'activités et BAAC@UML qui présente un profil pour étendre le diagramme d'activités également.

.4.1 SecureUML

La figure 13 présente le méta-modèle SecureUML [17]. Il reprend intégralement les concepts User, Rôle et Permission de modèle RBAC. La permission porte sur un ensemble

d'actions (ActionType). Chaque *ActionType* représente des opérations pertinentes définies sur un type particulier de ressources. Les actions lire, écrire et exécuter sont des exemples d'ActionType. *AutorisationConstraint* exprime une condition imposée à chaque appel à une opération sur une ressource particulière. La contrainte d'autorisation est attachée à un élément de modèle particulier représentant une ressource protégée. Notez que les actions peuvent être *Atomic* ou *Composite*. Une action atomique spécifie une seule opération concrète comme lecture, écriture, etc. Une action spécifie un ensemble d'actions.

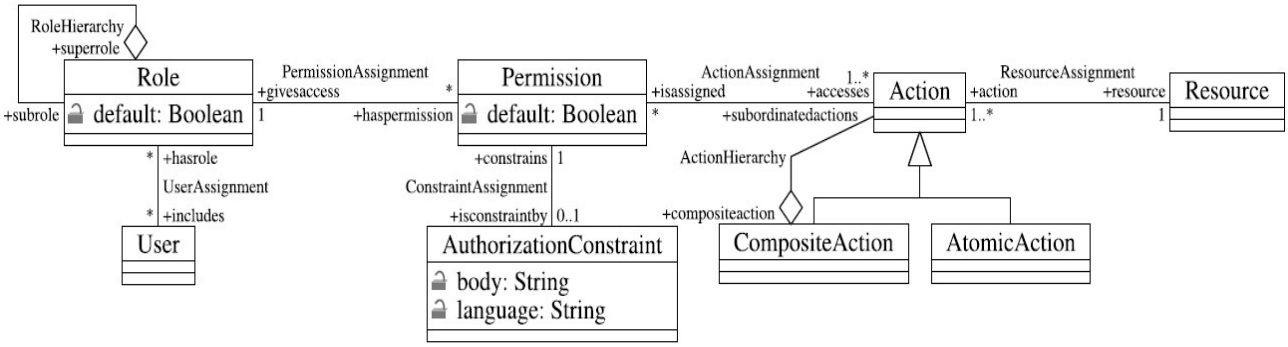


Figure 13 : Méta-modèle SecureUML [17]

SecureUML permet d'étendre tout langage de modélisation pour exprimer le contrôle d'accès aux systèmes. Cette extension est réalisée à travers le Dialecte (figure 14) qui met en relation le méta-modèle SecureUML et le méta-modèle du langage de modélisation à étendre. Le dialecte fournit une passerelle définissant le lien entre les points d'intégration des deux modèles, par exemple les éléments modélisés dans le langage de modélisation des systèmes sont classés comme des ressources à protéger dans le modèle de sécurité [13].

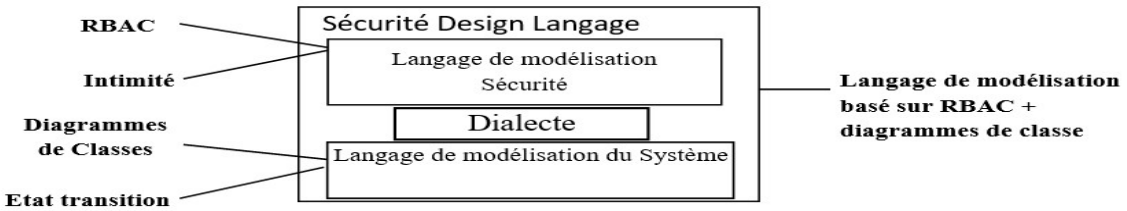


Figure 14 : Schéma d'un langage de conception de sécurité.[13]

La figure 15 montre un de diagramme de classe etendu avec le profil secureUML :

La permission InitiatorPermissions authorise le role MeetingInitiator d'éxécuter les actions *enterAgreementdetails* et *changeMeetingInfo*.

La permission *ParticipantPermissions* autorise le role Meetingparticipant d'éxécuter l'action *getAgreementInformation*.

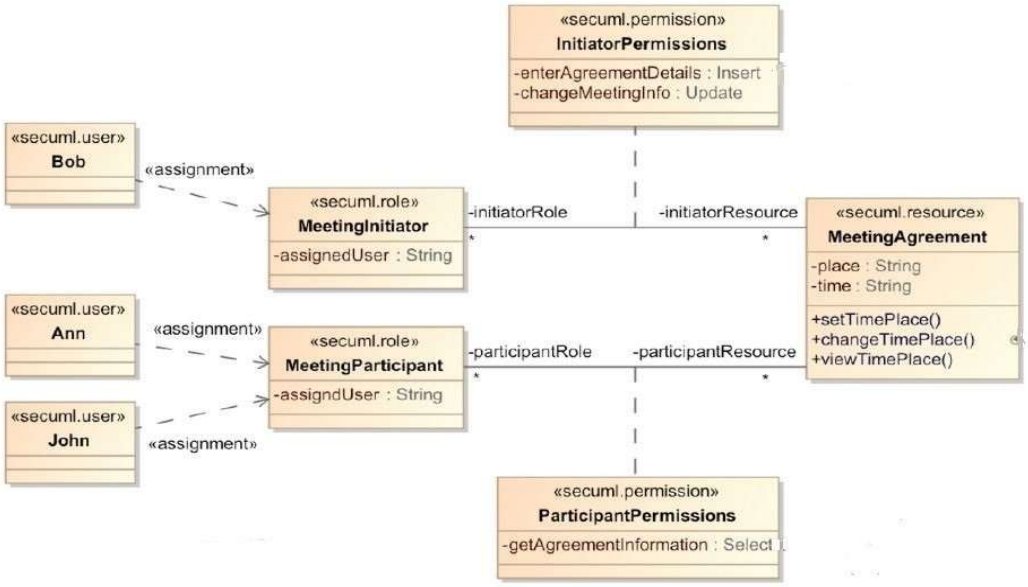


Figure 15 : Un exemple d'un diagramme de classes étendu avec le profil SecureUML[14]

4.2 UMLsec

UMLsec [11] propose une extension du diagramme d'activités d'UML. Ce profil représente un diagramme d'activités dans un package stéréotypé par « rbac ». Le profil *rbac* d'UMLsec utilise trois types de tags : *protected*, *role*, et *right*.

Protected : l'étiquette *protected* a pour valeurs les activités dans le diagramme d'activité dont l'accès doit être contrôlé.

Role : l'étiquette rôle peut avoir comme valeur une liste de paires (acteur, rôle) où l'acteur est un utilisateur qui exécute des actions dans le diagramme d'activité et le rôle présente une responsabilité assignée à l'utilisateur.

Right : L'étiquette *right* a comme valeur une liste de paires (role, right) où right représente le droit d'accès à une ressource protégée. La contrainte associée exige que les acteurs du diagramme d'activité n'exercent que des actions pour lesquelles ils ont les droits appropriés.

Exemple

La figure 16 donne un exemple de modèle SecureUML. Il décrit une partie simplifiée d'un processus métier où un crédit est établi pour un client d'une banque. Dans l'exemple, les gérants de banque traitent les dossiers de demandes de crédits des clients. Dans le cas des crédits importants (ceux qui dépassent le montant de 10.000), des superviseurs doivent autoriser le crédit avant le transfère de l'argent aux clients. Dans l'exemple donné, la ressource protégée est donc l'activité d'autorisation de crédit. Le superviseur doit être connecté au rôle *creditapprover*pour autoriser des crédits.

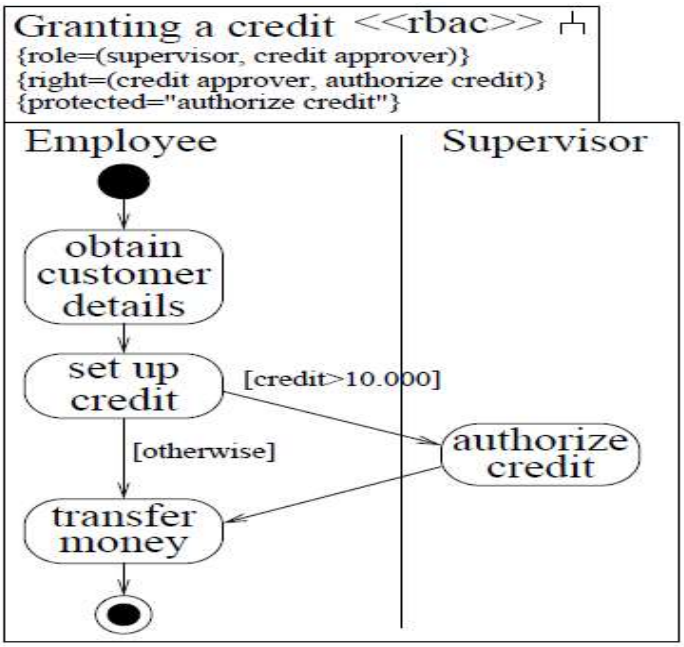


Figure 16 : Exemple de modèle UMLsec [11]

4. 3 BAAC@UML

BAAC@UML [16] (Business Activity Access Control with UML) introduit les concepts de précondition et de rôle pour protéger l'accès des utilisateurs aux activités d'un processus métier. Les préconditions de contrôle d'accès expriment des contraintes contextuelles qui permettent d'accorder des autorisations en fonction de l'état du système. Ils permettent de spécifier des politiques de contrôle d'accès qui dépendent des aspects dynamiques du système tels que les valeurs d'attributs.

BAAC@UML commence à partir des cas d'utilisation du système. Chaque cas d'utilisation encapsule un comportement qui représente les interactions entre le système et ses acteurs. Le profil BAAC @ UML détaille ces interactions en tenant compte d'une politique RBAC. [15]

BAAC@UML définit deux niveaux pour visualiser le comportement d'un cas d'utilisation. Un niveau abstrait qui ne détaille pas les autorisations RBAC et un niveau concret où les préconditions de contrôle d'accès sont associées aux actions.

BAAC@UML définit deux mécanismes pour spécifier une politique RBAC au niveau d'activités concrètes :

Le premier attribue l'activité concrète à un ou plusieurs rôles. Ces rôles correspondent aux acteurs du cas d'utilisation décrit par l'activité. Seuls les utilisateurs affectés à ces rôles peuvent exécuter les actions d'activité. L'attribution des rôles à une activité est formalisée par une précondition d'activité stéréotypée comme <<assignedRoles>>. La précondition d'activité reflète le fait que toutes les actions d'activité doivent être exécutées par un même utilisateur dans un rôle donné. [15]

Le second associe l'activité concrète et ses éléments aux préconditions de contrôle d'accès. Ces préconditions expriment des contraintes contextuelles qui accordent des autorisations en fonction de l'état du système ainsi que les utilisateurs et leurs rôles. BAAC@UML définit deux types de préconditions de contrôle d'accès : global et local. La précondition de contrôle d'accès globale doit être vérifiée lors de l'exécution de toute l'activité. Elle est exprimée comme une précondition d'activité stéréotypée comme <<GACPreCondition>> en utilisant le langage OCL. La précondition de contrôle d'accès locale est associée à une tâche d'activité ou à une action d'activité. La précondition locale est exprimée par une contrainte stéréotypée comme <<LACPreCondition>> en utilisant le langage OCL. [15]

La figure 17 présente un exemple du modèle BAAC@UML :

1-La contrainte <<assignedRoles>> spécifie que seuls les utilisateurs affectés au rôle Initiateur peuvent exécuter l'activité.

2-La contrainte <<GACPreCondition>> garantit que l'utilisateur exécutant les actions de l'activité concrètes est le créateur de la réunion (l'instance Mx de la classe Meeting).<<GACPreCondition>> doit être préservé dans l'exécution de toute l'activité.

3-La précondition de contrôle d'accès local *FA-InvitationAC* associée aux actions *Ix_getAnswer*, *Ix_setConfirmation (ok)* et *Ix_delete* vérifie que le créateur de la réunion peut uniquement lire, modifier ou supprimer une invitation de sa réunion.

4-La précondition de contrôle d'accès local *FA-ChangeAC* attachée à la tâche *reply to change* vérifie que le créateur de la réunion ne peut répondre que aux changements associées à l'invitation de sa réunion.

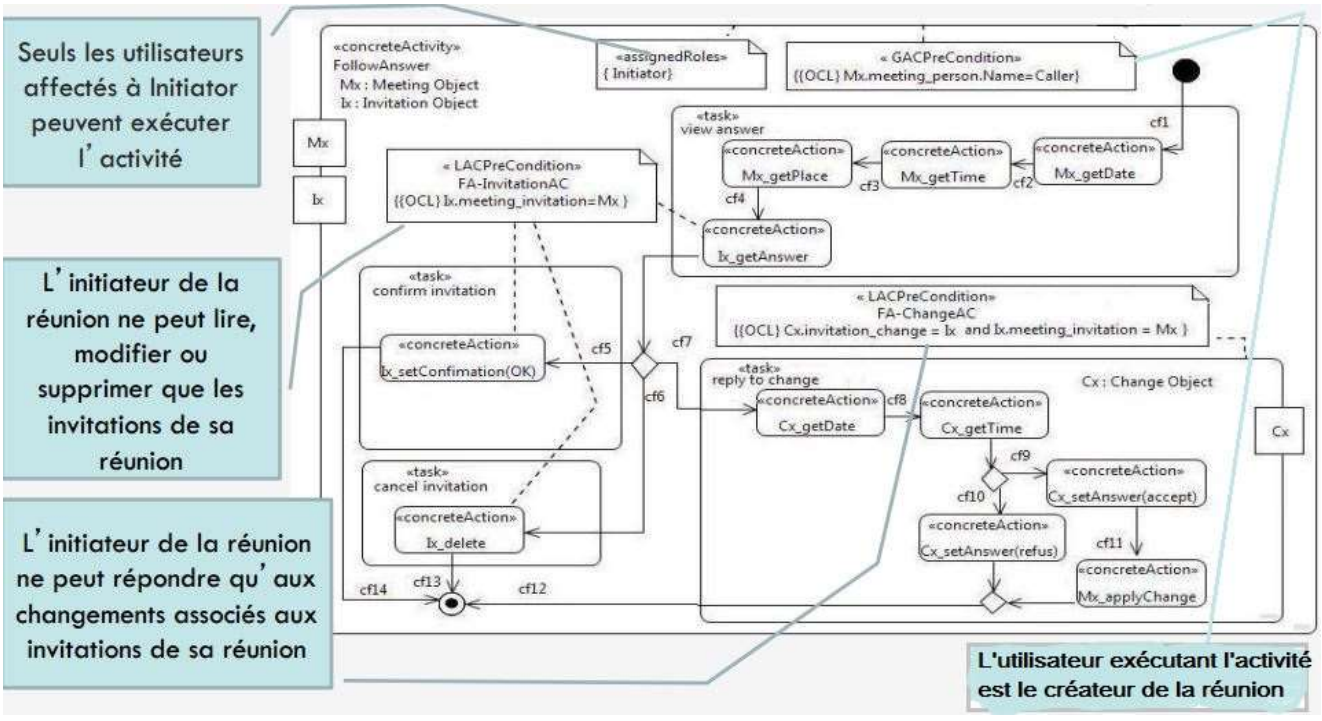


Figure 17 : Exemple de modèle BAAC@UML[16]

5. *conclusion*

Dans ce chapitre, nous avons défini le langage UML et la notion du profil, ensuite nous avons montré comment les profils SecureUML, UMLsec et BAAC@UML peuvent aider à définir des politiques de sécurité basées sur le modèle RBAC.

Les trois profils SecureUML, UMLsec et BAAC@UML intègre le concept de rôle qui permet de structurer les utilisateurs du système. SecureUML et BAAC@UML considèrent, en plus, des contraintes contextuelles d'autorisation (exprimées en OCL) qui dépendent de l'état fonctionnel du système, des utilisateurs et leurs rôles.

SecureUML spécifie la vue statique d'une politique RBAC (diagramme de classes) alors qu'UMLsec et BAAC@UML sont utilisés pour modéliser la vue dynamique d'une politique RBAC (diagrammes d'activités). Tenant compte du fait que la modélisation du système doit être abordée à partir de différentes perspectives de modélisation , cela signifie que les trois approches ne sont pas concurrents, mais elles se complètent en fournissant des points de vue différents au système sécurisé.

Chapitre 3

Mise en œuvre des modèles BAAC@UML et SecureUML dans un outil

1. *Introduction*

Un système logiciel est souvent modélisé par un ensemble de diagrammes UML qui spécifient les différents vues statiques et dynamiques de ce système. Cependant, il est nécessaire de vérifier et d'assurer la cohérence de ces diagrammes. Par conséquent, il est crucial de fournir des outils qui nous permettent de détecter les incohérences. Dans ce chapitre, nous allons se concentrer sur la réalisation d'un outil permettant la représentation graphique et la vérification de la cohérence des modèles SecureUML qui spécifient la vue statique d'une politique RBAC et des modèles BAAC@UML qui spécifient la vue dynamique de cette même politique. Nous vérifions la conformité de ces modèles par rapport à leurs méta-modèles.

2. Cohérence des diagrammes UML

Les outils contemporains pour UML fournissent un support insatisfaisant pour maintenir la cohérence de différents diagrammes UML.

2.1. Pourquoi la cohérence ?

Une modélisation comprend donc un ensemble de modèles décrivant des points de vue différents exprimés dans divers formalismes, par exemple, ceux inclus dans la notation UML [20]. En plus des éléments de modélisation et de leur représentation graphique, se pose le problème de la cohérence de modèles. Pour arriver à un consensus général de tous les acteurs de la modélisation, il est nécessaire de s'appuyer sur des démarches méthodologiques qui décrivent l'articulation des différents points de vue et l'utilisation correcte des éléments de modélisation. Par exemple, les descriptions statiques et dynamiques sont-elles compatibles, un même élément de modélisation at- il toujours la même interprétation ? Actuellement, en l'absence d'outils de vérification évolués, les incohérences ne peuvent être détectées que tardivement ce qui implique un retour aux phases initiales de développement. [26] De ce fait, les problèmes de cohérence doivent être traités dans les premières phases d'analyse et de conception.

2.2. Type de cohérences

Selon [27] et [28], il existe deux types de cohérence. Le premier intra-modèle, est concerné par la cohérence entre les artefacts dans un modèle donné. Le deuxième inter-modèle assure la cohérence entre deux modèles en relation mais de types différents.

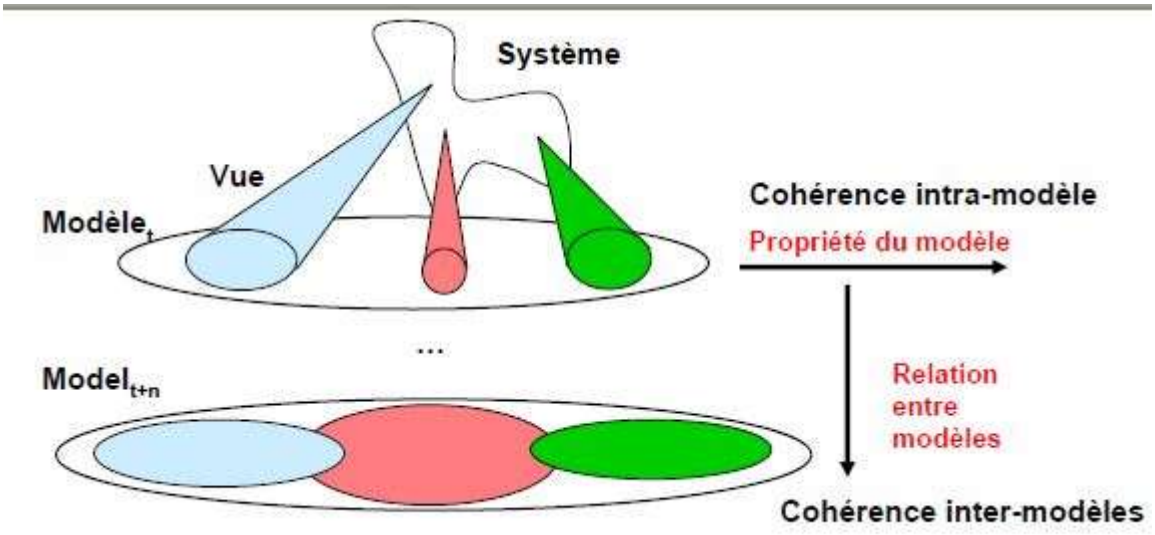


Figure 18 : problèmes de cohérence de modèles [21]

2.2.1 Cohérence intra-modèle

Une règle intra-modèle exprime la cohérence des différents éléments de modélisation au sein d'un même point de vue. La présence d'un acteur dans un diagramme de cas d'utilisation, par exemple, ne se justifie que s'il interagit avec le système, c'est-à-dire s'il participe au moins à un cas d'utilisation. [26]

2.2.2 Cohérence inter-modèles

Une règle inter-modèle vérifie la cohérence des différents éléments de modélisation au sein de différents points de vue. Par exemple, deux objets d'un diagramme de collaboration ne peuvent échanger des messages que si les classes dont ils sont instances sont connectées via une association navigable. [26]

2.3 Vérification de la cohérence

Allaki et al [29] présentent les approches existantes qui permettent de vérifier les incohérences dans les modèles UML. Certaines de ces approches utilisent des techniques formelles comme les diagrammes d'états-transitions, la logique et l'algèbre de processus. Ces techniques consistent à transformer les modèles UML et leurs règles de cohérence dans un langage formel et ensuite à détecter les incohérences à l'aide de mécanismes d'inférence du langage formel. [25] D'autres approches comme [30] et [31] proposent l'utilisation du langage OCL [22]. Il s'agit ici d'exprimer les règles de cohérence directement en OCL sur le méta-modèle d'UML. Notre travail porte sur cette approche qui est basée sur un langage de contraintes riche, intégré dans UML et supporté par plusieurs outils. [25]

3. Cohérence des modèles BAAC@UML et SecureUML

Notre travail propose l'introduction d'un ensemble de règles pour assurer la cohérence intra-modèle BAAC@UML et intra-modèle SecureUML.

3.1 Règles de cohérence intra-modèle BAAC@UML

Dans cette section, nous allons introduire des règles pour vérifier la cohérence des modèles BAAC@UML.Ces règles sont exprimées en langage OCL [22] et elles sont associées aux méta-classes du méta-modèle BAAC@UML (voir figure 19).

R1) ActivityMissingAssignedRole

Une activité concrète est associée à un ou plusieurs rôles. Les utilisateurs assignés à un de ces rôles sont autorisés à exécuter l'activité. L'invariant ActivityMissingAssignedRole ci-dessous permet de garantir que toute activité concrète est assignée à au moins un rôle autorisé à l'exécuter.

context ConcreteActivity inv ActivityMissingAssignedRole:
self.oclIsTypeOf (ConcreteActivity) implies self.role ->notEmpty()

R2) ActionMissingCalledOperation

Une action concrète doit faire appel à une opération de classe. L'invariant ci-dessous permet de garantir que toutes les actions concrètes font appel à des opérations.

context ConcreteAction inv ActionMissingCalledOperation:
self.oclIsTypeOf (ConcreteAction) implies self.operation->notEmpty ()

R3) LACPreConditionNotRelatedToElement

L'invariant ci-dessous garantit qu'une pré-condition locale doit être associée à unélément (elle ne doit pas être isolée).

Context LACPreCondition inv LACPreConditionNotRelatedToElement :
self.oclIsTypeOf (LACPreCondition) implies self.base_Constraint.constrainedElement ->notEmpty()

R4) LACPreConditionNotRelatedToConcreteAction

L'invariant ci-dessous garantit qu'une précondition locale doit être associée à une action concrète.

context LACPreCondition inv LACPreConditionNotRelatedToConcreteAction :
self.*base_Constraint.constrainedElement->notEmpty*() implies
self.*base_Constraint.constrainedElement->forAll*(*oclIsKindOf*(*OpaqueAction*))

R5) ConcreteActivityIsEmpty

L'invariant ci-dessous garantit qu'une activité concrète ne doit pas être vide. Elle doit au moins inclut un nœud d'activité.

Context ConcreteActivity inv ConcreteActivityIsEmpty :
self.oclIsTypeOf(ConcreteActivity) implies self.base_Activity.node ->notEmpty()

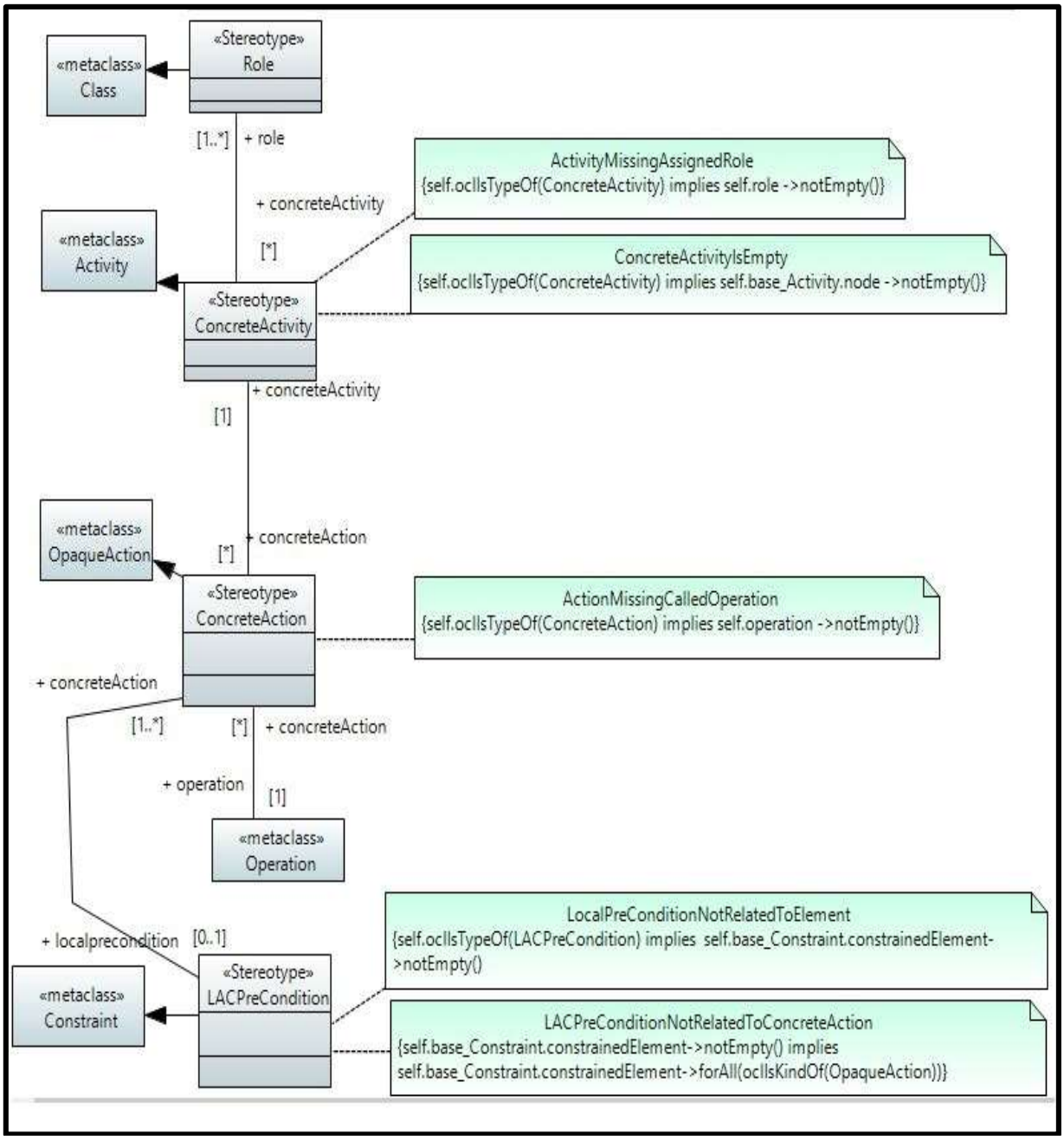


Figure 19 : méta-modèle BAAC@UML avec règles de cohérence associées

3.2 Règles de cohérence intra-modèle SecureUML

Dans cette partie, nous allons introduire des règles pour vérifier la cohérence des modèles SecureUML qui expriment la vue statique de politiques RBAC (voir figure 20).

R6) PermissionMissingRole

Une permission doit être associée à un rôle. L'invariant PermissionMissingRole ci-dessous permet de garantir que toutes les permissions sont associées à des rôles.

Context Role inv PermissionMissingRole:
self.oclIsTypeOf(Permission) implies self.role->notEmpty()

R7) PermissionMissingEntity

Une permission doit être associée à une classe protégée. L'invariant *PermissionMissingEntity* ci-dessous permet de garantir que toutes les permissions sont associées à des entités.

```
context Permission inv PermissionMissingEntity:
self.oclIsTypeOf(Permission) implies self.entity->notEmpty()
```

R8) PermissionMissingAction

L'invariant PermissionMissingAction ci-dessous permet de garantir que toute permission possède au moins une action.

```
Context Permission inv PermissionMissingAction:
self.oclIsTypeOf(Permission) implies self.secureaction->notEmpty()
```

R9) AuthorizationConstraintNotRealatedToElement :

Une contrainte d'autorisation doit être associée à un élément. Elle ne doit pas être isolée. Cette contrainte est vérifiée par l'invariant ci-dessous.

```
context AuthorizationConstraint inv AuthorizationConstraintNotRealated :
self.oclIsTypeOf(AuthorizationConstraint) implies
self.base_Constraint.constrainedElement->notEmpty()
```

R9) AuthorizationConstraintNotRealatedToPermission :

L'invariant ci-dessous assure qu'une contrainte d'autorisation doit être associée à une permission.

```
context AuthorizationConstraint inv AuthorizationConstraitNotRealatedToPermission :
self.base_Constraint.constrainedElement->notEmpty() implies
self.base_Constraint.constrainedElement->forAll(oclIsKindOf(AssociationClass))
```

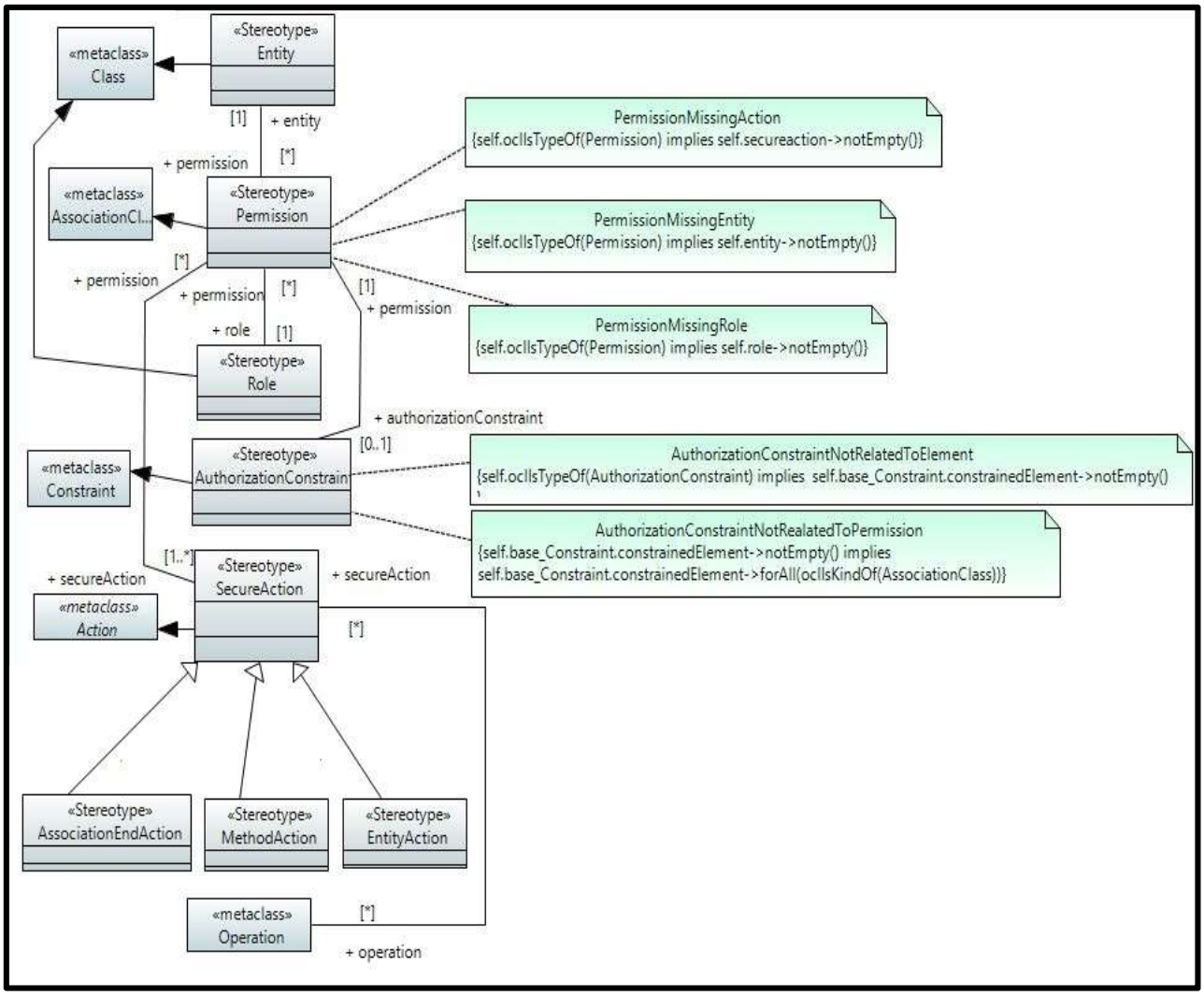


Figure 20 : Méta-modèle SecureUML avec règles de cohérence associées.

4. Mise en œuvre des règles dans un outil

Les méta-modèles BAAC@UML et SecureUML ainsi que les différentes règles de cohérence associées à leurs éléments (voir figures 19 et 20) sont mis en œuvre dans un outil. Cet outil sera utilisé pour détecter les incohérences des modèles SecureUML et BAAC@UML.

Pour la mise en œuvre des méta-modèles et des règles de cohérence, nous avons utilisé l'outil Papyrus [23]. Cet outil est disponible sous forme de plugin de la plateforme EMF d'eclipse. Il fournit un environnement intégré pour l'édition de modèles UML et SysML. Son objectif principal est de mettre en œuvre la spécification standard complète d'UML2. Papyrus fournit également un support étendu pour les profils UML. Il comprend donc toutes les fonctionnalités pour définir et appliquer des profils UML de manière très riche et efficace. Il permet de définir un métamodèle, d'en créer des diagrammes et de vérifier les contraintes OCL sur ces diagrammes. [24]

Dans notre approche, nous avons commencé par la création des diagrammes de profil qui spécifient les méta-modèles BAAC@UML et SecureUML. Les diagrammes de profil

définissent les stéréotypes représentant les concepts de deux profils et lie ces stéréotypes aux méta-classes UML correspondantes. Ils définissent également les liens de chaque stéréotype avec les autres stéréotypes ou méta-classes UML ainsi que les contraintes OCL associées aux stéréotypes.

Après la création de diagrammes de profil, nous avons les appliqué sur les diagrammes UML pour permettre création des modèles BAAC@UML et SecureUML et la vérification de leur cohérence.

5. Exemples de détection d'incohérences

Notre outil permet de signaler tout type d'incohérence des modèles SecureUML et BAAC@UML.

5.1 Test des modèles BAAC@UML

Afin de tester les règles de cohérence associées au méta-modèle BAAC@UML, nous allons utiliser l'exemple de *Meeting* présenté dans [25]. Ce système s'adresse à deux types d'utilisateurs : les « initiateurs » qui planifient des réunions et les « participants » qui répondent aux invitations des initiateurs. La figure 21 présente un modèle BAAC@UML décrivant l'activité « répondre à invitation ». Cette activité est réalisée par un utilisateur associé au rôle participant. Après la lecture de l'invitation, le participant choisit entre l'acceptation de l'invitation (I.setAnswer(accept)), le refus de l'invitation (I.setAnswer(refus)) ou la proposition d'un changement de réunion (C.create, C.setDate, C.setTime). La précondition de contrôle d'accès *MeetingRead* autorise les participants d'une réunion à lire ses informations.

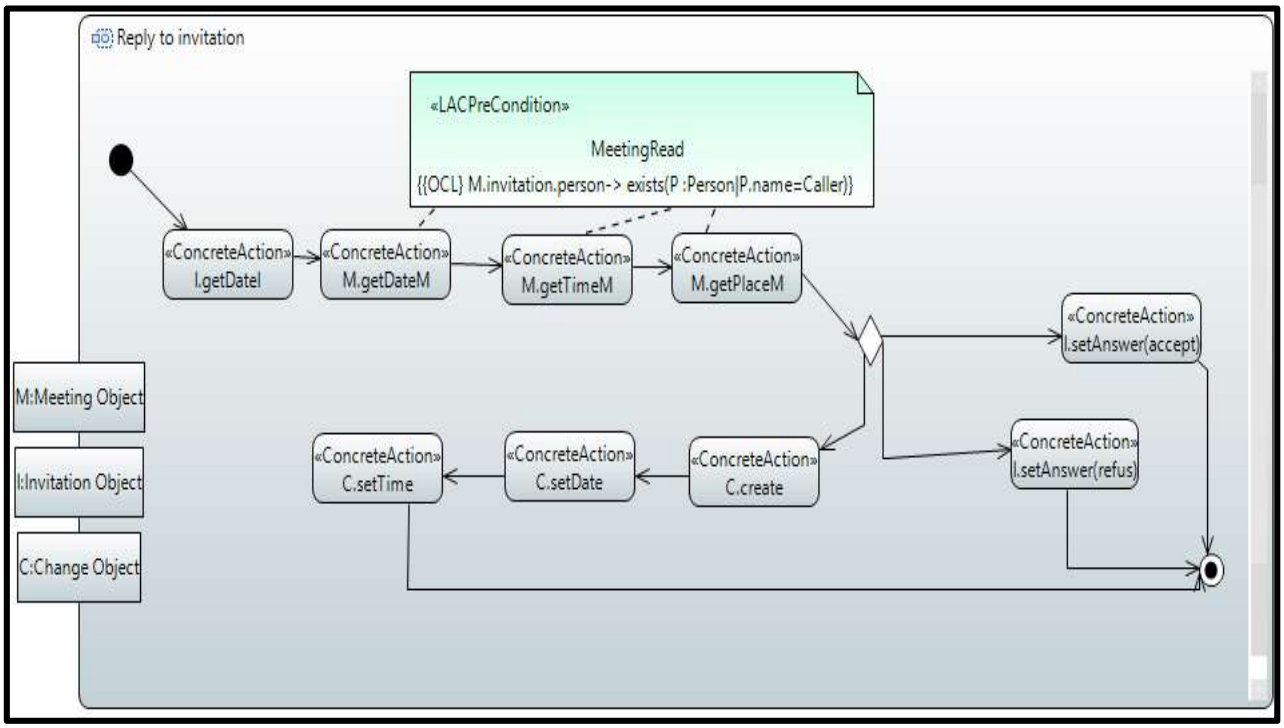


Figure 21 : Activité BAAC@UML répondre à invitation.

Dans la figure 22, nous avons crée des cas d'incohérences à partir du modèle de la figure 21 pour voir la réaction de notre outil. Après la validation du modèle BAAC@UML, la fenêtre *Model Validation* indique trois cas de violation :

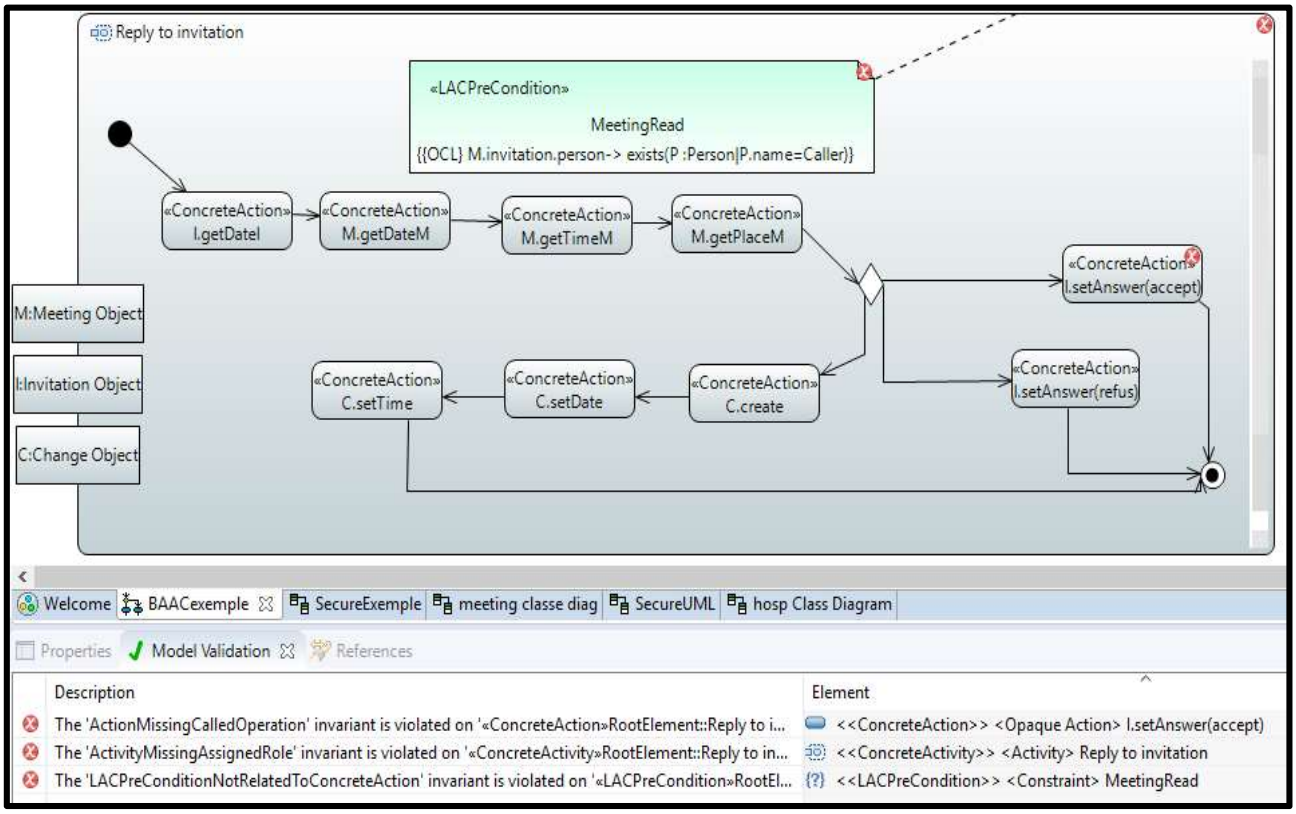


Figure 22 : Exemple d'incohérence dans un modèle BAAC@UML

- ✓ Le premier message correspond à l'invariant ActionMissingCalledOperation. Il indique que l'action concrète *I.setAnswer(accept)* ne fait pas appel à une opération de classe.

- ✓ Le deuxième message correspond à l'invariant ActivityMissingAssignedRole. Il indique que l'activité *répondre à invitation* n'est associée à un rôle.

- ✓ Le troisième message correspond à l'invariant LACPreConditionNotRelatedToConcreteAction .Il indique que la précondition *MeetingRead* n'est pas associée à une concrète action.

Après la détection des incohérences, des corrections ont été réalisées sur l'exemple de la figure 23 :

- ✓ Nous avons associé l'operation *setAnswerI(accept*) a l'action concrete *I.setAnswer(accept).*
- ✓ Nous avons affecté l'activité *répondre à invitation* au rôle participant.

- ✓ Nous avons associé la précondition *MeetingRead* aux actions de lecture des informations de réunions *(M.getDateM, M.getTimeM, M.getPlaceM)*.

5.2 Tests du modèles SecureUML

Afin de tester les règles de cohérence associées au méta-modèle SecureUML, nous allons utiliser l'exemple de système d'information médical proposé par [32]. Ce système gère les opérations liées à la prise en charge des patients en amont ou en aval d'une éventuelle hospitalisation. La figure 23 présente un exemple du modèle SecureUML qui contrôle l'accès aux classes ManagementAct et MedicalAdvice. Le modèle définit deux permissions : la première « TeamDoctorMA » spécifie que seul le rôle « TeamDoctor » peut créer ou modifier l'entité « ManagementAct ». La deuxième « ParmAdviceMa » spécifie que seul le rôle « PARM », qui représente opérateurs médicaux, peut créer ou modifier l'entité « MedicalAdvice ».La contrainte d'autorisation associée à la permission « MedicalAdvice » signifie qu'un conseil médical ne peut être modifié que par l'acteur pré-hospitalier qui l'a créé. Les actions de type «EntityAction» représentent des opérations abstraites qui donnent lieu à des appels d'opérations concrètes réalisables sur des classes fonctionnelles.

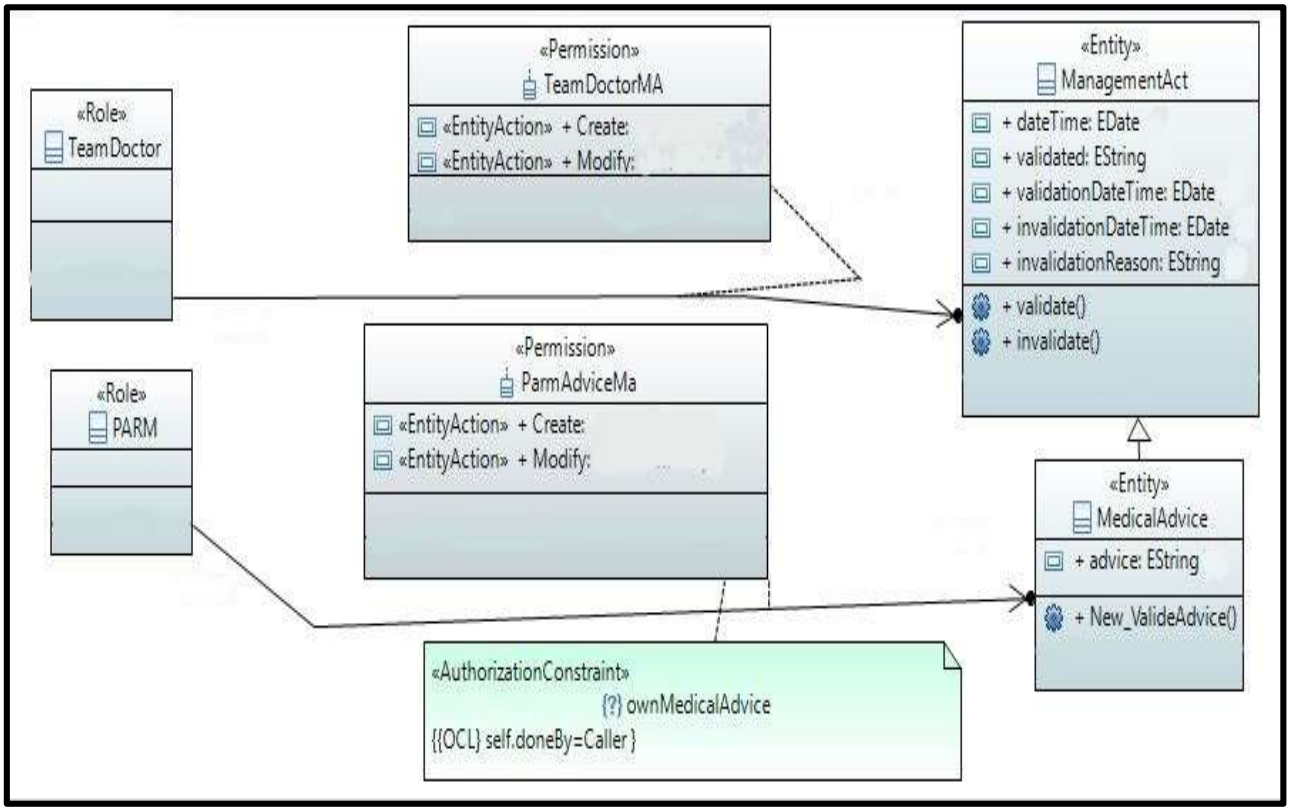


Figure23 : Le modèle SecureUML de contrôle d'accès au système d'information médical

Dans la figure 24, nous avons crée des cas d'incohérences dans le modèle SecureUML de la figure 23 pour voir la réaction de notre outil. Après la validation, la fenêtre *Model Validation* indique quatre cas de violation :

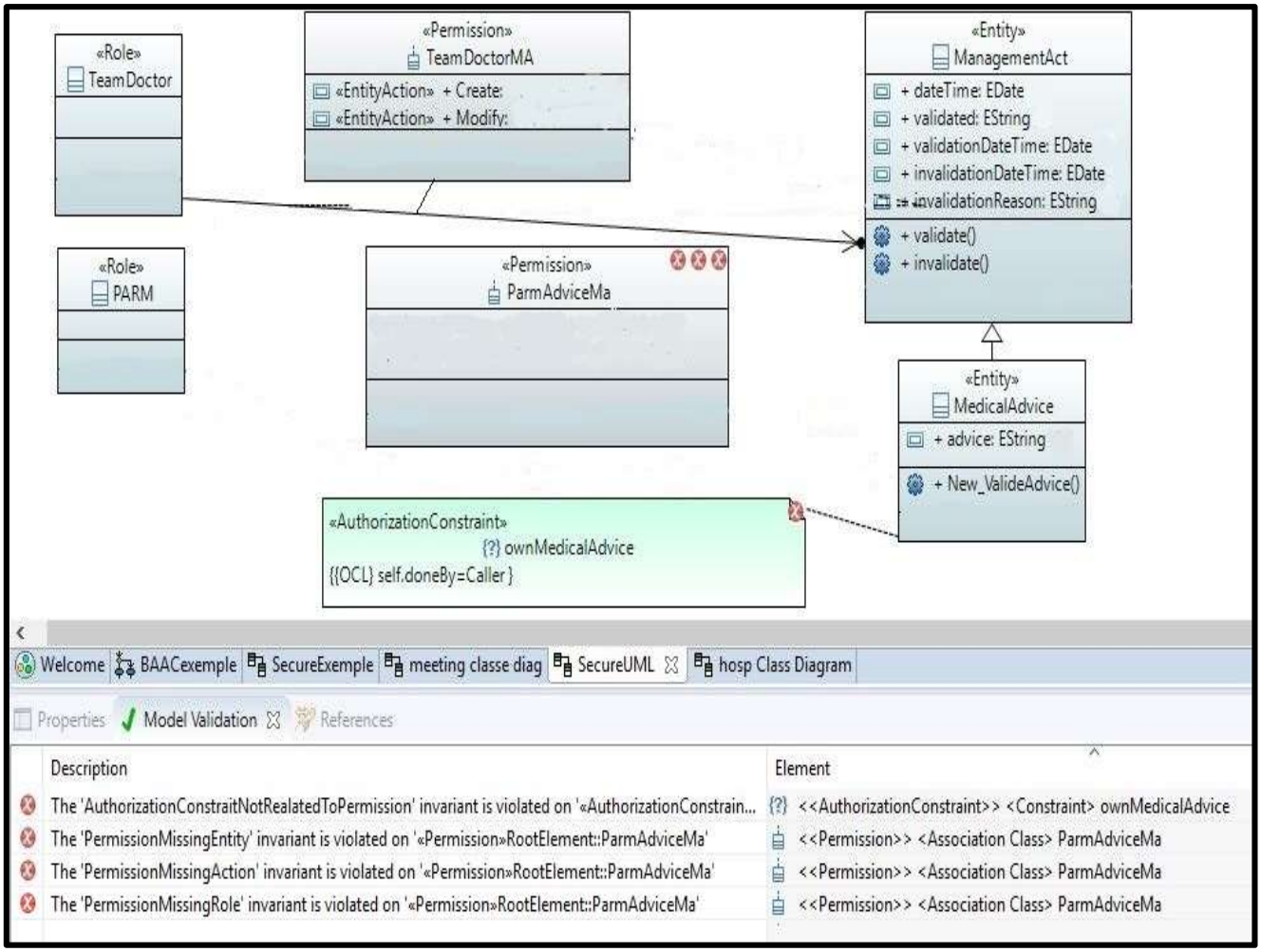


Figure 24 : Exemple d'incohérence dans un modèle SecureUML

- ✓ Le premier message correspond à l'invariant AuthorizationConstrainteNotRealatedToPermission. Il indique que la contrainte d'autorisation « ownMedicalAdvice » n'est pas associée à une permission.
- ✓ Le deuxième message correspond à l'invariant *PermissionMessingEntity*. Il indique que la permission « ParmAdviceMA » n'est pas associé à une entité.
- ✓ Le troisième message correspond à l'invariant PermissionMissingAction .Il indique que la permission « ParmAdviceMA » ne contient pas d'action.
- ✓ Le quatrième message correspond à l'invariant RoleMissingPermission. Il indique que la permission « ParmAdviceMA » n'est pas associée à un rôle.

Après la détection des incohérences, des corrections ont été réalisées sur l'exemple de la figure 24 :

- ✓ Nous avons associé la contrainte d'autorisation« ownMedicalAdvice » à la permission « ParmAdviceMA ».
- ✓ Nous avons ajouté l'action Create et Modify a la permission « ParmAdviceMA ».

6. conclusion

Dans ce chapitre, nous avons spécifié les méta-modèles BAAC@UML et SecureUML. Nous avons également proposé un ensemble de règles permettant d'assurer la cohérence des modèles SecureUML et BAAC@UML. Les méta-modèles et les règles de cohérence associées ont été et mises en œuvre dans un outil basé sur Eclipse qui montre toutes les incohérences des modèles de deux profils.

Conclusion générale

Le contrôle d'accès est utilisé pour objectif de maîtriser l'accès à l'information. Il permet d'autoriser l'accès des utilisateurs légitimes et d'empêcher les accès non-autorisés. Dans ce projet, nous avons étudié les travaux qui proposent des extensions de diagrammes UML pour tenir compte une politique de contrôle d'accès basée sur le modèle RBAC.

Dans le premier chapitre , nous avons présenté le modèle RBAC qui est le modèle le plus répendu pour controler l'accès aux systèmes. Le concept central du modèleRBAC est le rôle. En RBAC, l'accès à un objet est accordé à un utilisateuren fonction du rôle qui lui est associé. Les avantages de RBAC sont nombreux et surtout il établit une relation stable entre la sécurité des données, la structure d'une organisation et les tâches dans une organisation. [2]

Dans le deuxième chapitre, nous avons présenté les combinaisons entre le modèle RBAC et les diagrammes d'UML. Nous avons visé par notre étude les trois profils : SecureUML qui etend le diagramme de classes, UMLsec et BAAC@UML qui etendent le diagramme d'activités. Nous avons conclu notre études par une comparaison entre les trois profils.

Dans le dernier chapitre, nous avons proposé un outil basé sur Eclipse qui permet la représentation graphique des modèles BAAC@UML et SecureUML ainsi que la vérification de leur cohérence. Cet outil met en œuvre les méta-modèles SecureUML et BAAC@UML avec des règles de cohérence associées. Il montre tous les cas de non-conformité dans les modèles BAAC@UML et SecureUML.

Les references

[1] Bokefode Jayant, D. Ubale Swapnaja A, Apte Sulabha S, Modani Dattatray G. Analysis of DAC MAC RBAC Access Control based Models for Security, International Journal of Computer Applications. Volume 104 – No.5, October 2014.

[2] Raimundas Matulevičius and Marlon Dumas, A Comparison of SecureUML and UMLsec for Role-based Access Control, 2010

[3] American National Standard for Information Technology – Role Based Access Control ANSI INCITS 359-2004. American National Standards Institute, Inc.

[4] Pascal Andre, Transformation de modèles dans le cadre de l'OMG. Master Miage M2 - Option ISI. January 18, 2011

[5] DAVID F. FERRAIOLO National Institute of Standards and Technology, SERBAN GAVRILA VDG Incorporated and D. RICHARD KUHN and RAMASWAMY CHANDRAMOULI National Institute of Standards and Technology. Proposed NIST Standard for Role-Based Access Control. ACM Transactions on Information and System Security, Vol. 4, No. 3, August 2001 p 236

[6] RAVI SANDHU George: Mason University, GAIL-JOON AHN: University of North Carolina at Charlotte, Role-Based Authorization Constraints Specification. 2000

[7] Indrakshi Ray , Na Li , Robert France , Dae-Kyoo Kim Using :Colorado State University, Fort Collins, CO : UML to visualize role-based access control constraints :Conference Paper·January 2004

[8] ITSEC, Information Technology Security Evaluation Criteria, v 1.2, ISBN 92-826-3005-6, Office des publications officielles des Communautés Européennes, Luxembourg, 1991.

[9] J. Gabay et D. Gabay, UML2 Analyse et Conception, Groupe Dunod, ISBN : 978-2-10-053567-5,2008

[10] David Basin, Manuel Clavel, Jürgen Doser, Marina Egeac, Automated analysis of security-design models.Information Security Group, ETH Zurich, Switzerlandb IMDEA Software Institute, Madrid, Spain. Computer Science Department, Universidad Complutense de Madrid, Spain, 2009.

[11] Jurjens, J.: UMLsec: Exending UML for Secure Systems Development. In: Proceedings of the 5th International Conference on The Unified Modeling Language, LNCS. Springer-Verlag, 2002

[12] P. A. Muller et N. Gaertner, Modélisation objet avec UML, Groupe Eyrolles, ISBN : 2-212-11397-8, 2004.

[13] D. Basin, J. Doseret T. Lodderstedt, Model Driven Security: from UML Models to Access Control Infrastructures, Information Security Group, ETH Zurich Interactive Objects Software GmbH Freiburg, Vol 15, N°1, Janvier 2006.

[14] Ferraiolo D. F,Sandhu,R., Gavrila,S., Kuhn,D.R., Chandramouli,R ,RBAC: Role-based Access Control..proposed NIST Standard for role-based Acces Control.ACM transaction on information and system security (TISSEC), 4(3), 224—274,2001

[15] Salim Chehida, Akram Idani, Yves Ledru, Mustapha Kamel Rahmouni .Combining UML and B for the Spécification and Validation of RBAC Policies in Business Process Activities. IEEE RCIS, Grenoble, France 2016

[16] Salim CHEHIDA, Approche de spécification et validation formelles de politiques RBAC au niveau des processus métiers, S. CHEHIDA, AFADL2016, Besançon, France.

[17] Basin, D., Doser, J., Lodderstedt, T : Model Driven Security : from UML Models to Access Control Infrastructure. ACM Transactions on Software Engineering and Methodology (TOSEM), 15 (1), 39--91 (2006)

[18]Laurent Bloch et Christophe Wolfhugel, Sécurité informatique - Principes et méthode,Eyrolles, 2011

[19] James Rumbaugh, Ivar Jacobson,Grady Booch : The Unified Modeling Language Reference Manual, Second Edition ,2004

[20] Muller P.-A., Gaertner N. : Modélisation objet avec UML-2ème édition, Edition Eyrolles, ISBN : 2-212-09122-2, Mars 2000

[21] Jean Louis Sourrouille, Vérification de la cohérence de modèles UML Workshop Consistency Problems in UML-based Software Development, 2005.

[22] OCL2. Object Constraint Language (OCL) Version 2.3.1. Object Management Group.(http://www.omg.org/spec/OCL/2.3.1/PDF/), (2012).

[23] https://eclipse.org/papyrus/download.html

[24] https://wiki.eclipse.org/Papyrus_User_Guide

[25] Salim Chehida, Akram Idani, Yves Ledru ,Mustapha Kamel Rahmouni .Extension du diagramme d'activité pour la specification de politique RBAC , ISI Volume 21 –n°2/2016

[26] Pierre Bazex - Jean-Paul Bodeveix - Christophe Le Camus - Thierry Millan – Christian Percebois .Vérification de modèles UML fondée sur OCL .IRIT-UPS Université Paul Sabatier – 31062 Toulouse In {INFORSID, Nancy, 03/06/03-06/06/03} pp. 185-200, juin 2003

[27] Kuzniarz L., Staron M. Inconsistencies in student designs. In Workshop on Consistency Problems in UML-based software development II. 2003.

[28] Huzar Z., Kuzniarz L., Reggio G., Sourrouille J. L. Consistency Problems in UML Based Software Development. In N. J. Nunes, B. Selic, A. R. d. Silva, A. T. Alvarez (Eds.), UML Modeling Languages and Applications, p. 1-12. Springer Berlin Heidelberg. 2005.

[29] Allaki D., Dahchour M., En-nouaary A. A new taxonomy of inconsistencies in UML Models with their detection methods for better MDE. International Journal of Computer Science and Applications, vol. 12, no 1, p. 48 – 65.2015

[30] Gogolla M., Kuhlmann M., Hamann L. Consistency, Independence and Consequences in UML and OCL Models. In Tests and Proofs, vol. 5668, p. 90-104. Springer Berlin Heidelberg.2009.

[31] Chiorean D., Pa¸sca M., Cârcu A., Botiza C., Moldovan S. Ensuring UML Models Consistency Using the OCL Environment. Electronic Notes in Theoretical Computer Science, vol. 102, p. 99-110.2004.

[32] Yves Ledru, Akram Idani, and Jean-Luc Richier.Validation of a Security Policy by the Test of its Formal B Specification - a Case Study. IEEE/ACM 3rd FME Workshop on Formal Methods in Software Engineering 2015.