Extensions du diagramme d'activité pour la spécification de politiques RBAC

Salim Chehida [1], **Akram Idani** [2], **Yves Ledru** [3], **Mustapha Kamel Rahmouni** [4]

1. *Département Informatique, Faculté des Sciences Exactes et Appliquées, Université Oran1 Ahmed BenBella*
   BP 1524 EL Mnaouer Oran- Algérie
   Salim.Chehida@imag.fr
2. *Univ. Grenoble Alpes, LIG, 38000 Grenoble, France*
   CNRS, LIG, 38000 Grenoble, France
   Akram.Idani@imag.fr
3. *Univ. Grenoble Alpes, LIG, 38000 Grenoble, France*
   CNRS, LIG, 38000 Grenoble, France
   Yves.Ledru@imag.fr
4. *Département Informatique, Faculté des Sciences Exactes et Appliquées, Université Oran1 Ahmed BenBella*
   BP 1524 EL Mnaouer Oran- Algérie
   kamelrahmouni1946@gmail.com

RÉSUMÉ. *L'ouverture des organisations et de leurs systèmes d'information (SI) pose le problème de leur sécurité. Ainsi, la définition de politiques de contrôle d'accès constitue une étape cruciale dans le développement d'un SI. Ce travail propose une approche pour la spécification d'une politique de sécurité, basée sur le modèle RBAC, au niveau des workflows d'un processus métier. Cette approche consiste à propager les permissions définies sur un diagramme de classes modélisé avec le profil SecureUML, vers des contraintes associées aux activités décrivant un processus métier. Les diagrammes d'activité sont définis à deux niveaux : un niveau abstrait qui ne détaille pas ces permissions et un niveau concret où des contraintes sont associées à certaines actions ou à l'ensemble du diagramme. Nous avons défini un métamodèle dans le but de spécifier la sémantique de ces diagrammes d'activité ainsi que la sémantique de leurs liens avec les modèles SecureUML. Cet article présente une version étendue de (Chehida* et al.*, 2015), qui propose un ensemble de règles permettant d'assurer la cohérence entre les modèles d'activité concrète et les modèles SecureUML. Nous avons étendu également ce travail par la mise en œuvre de nos règles dans un outil qui montre toutes les contradictions des deux modèles.*

Ingénierie des systèmes d'information – n° 2/2016, 11-37

ABSTRACT. *The evolution of organizations and their information systems towards more openness raises the challenge of their security. The definition of an access control policy is a major activity in the design of an Information System. This paper proposes an approach for the specification of security policies, based on the RBAC model, at the workflow level. This approach propagates permissions defined on a class diagram, using the SecureUML profile, towards constraints linked to the business process activities. Activity diagrams are defined at two levels : an abstract level which does not detail these permissions and a concrete level where constraints are associated to specific actions or to the whole diagram. A metamodel was been defined in order to specify the semantics of these activity diagrams and the semantics of their links with SecureUML models. This paper presents an extended version of (Chehida* et al.*, 2015), which proposes a set of rules to ensure consistency between the concrete activity models and SecureUML models, and the implementation of these rules in a tool that reports all contradictions between both models.*

MOTS-CLÉS : *RBAC, Workflow, processus métier, SecureUML, UML2, diagramme d'activité, cohérence.*

KEYWORDS: *RBAC, Workflow, business process, SecureUML, UML2, activity diagram, consistency.*

DOI:10.3166/ISI.21.2.11-37

1. Introduction

Avec la croissance fulgurante que connait le monde des télécommunications et l'ouverture des Systèmes d'Information (SI), la spécification de ces SI ne peut plus se contenter de la seule modélisation fonctionnelle mais doit prendre en compte les besoins de sécurité. La définition des cas d'utilisation et des activités associées doit s'enrichir en intégrant le contrôle d'accès dans la description de ces processus. Adopté comme une norme ANSI / INCITS (ANSI, 2004), le modèle RBAC (Role-Based Access Control) (Ferraiolo *et al.*, 2003) est le modèle le plus répandu pour contrôler l'accès aux SI. Dans ce modèle, l'accès à un objet est accordé à un utilisateur en fonction du rôle qui lui est associé. Cependant, si RBAC est bien adapté à l'expression d'une politique de contrôle d'accès dans une vue statique comme le diagramme de classes, il est plus difficile de formaliser une politique de contrôle d'accès dans une vue dynamique du système (Basin *et al.*, 2006). Ce travail s'intéresse à cette problématique ; nous proposons une nouvelle approche qui permet de représenter une politique de sécurité, basée sur le modèle RBAC, dans une vue dynamique en exprimant les règles de contrôle d'accès sur les workflows [1] d'un SI.

SecureUML est un profil UML qui permet de spécifier une politique de contrôle d'accès RBAC au travers des diagrammes d'UML. Ce profil permet d'exprimer des contraintes contextuelles, appelées aussi contraintes d'autorisation. Ces contraintes portent sur l'état spécifié par le diagramme de classes et conditionnent l'évaluation

1. Flux de travail

des permissions. Nous proposons de compléter cette vue statique par des diagrammes d'activité d'UML 2 (UML2, 2011), qui sont l'un des modèles préconisés pour la modélisation des workflows. Nous étendons ces diagrammes pour qu'ils représentent le déroulement d'un processus métier en tenant compte des différentes permissions et contraintes d'autorisation d'une spécification SecureUML.

Notre approche identifie les exigences de sécurité dès les premières étapes du cycle de développement de logiciels. Elle intègre trois vues de modélisation. La première vue est fonctionnelle ; elle est représentée par le diagramme de cas d'utilisation qui montre des acteurs interagissant avec les fonctions d'un système. La deuxième est statique ou structurelle et décrit les données d'un système sous forme de classes et d'associations, et les permissions qui leur sont associées. Elle est décrite en SecureUML. La troisième est dynamique et permet d'établir un pont entre les deux premières visions. Pour construire ce pont, nous avons utilisé le diagramme d'activité à deux niveaux différents. Le premier est abstrait et permet de décrire les cas d'utilisation par une coordination d'actions de haut niveau exécutées par les acteurs du système. Le deuxième est concret et consiste à exprimer les actions abstraites par des actions de bas niveau qui représentent les opérations des classes. Ce diagramme concret tient compte des permissions SecureUML et leurs éventuelles contraintes d'autorisation qui seront associées comme préconditions locales aux actions concrètes concernées. Le contrôle d'accès à une activité concrète est aussi réalisé en affectant l'activité à un ou plusieurs rôles. Seuls les utilisateurs assignés à ces rôles peuvent exécuter les actions concrètes de l'activité. La sémantique de nos extensions de diagrammes d'activité et de leurs liens avec les modèles SecureUML est spécifiée au moyen d'un métamodèle et de contraintes associées.

Le présent travail vise l'expression d'une politique de contrôle d'accès basée sur le modèle RBAC au niveau des activités d'un processus métier et poursuit les objectifs suivants :

- Propager une politique d'accès statique au niveau des activités.
- Vérifier les permissions intrinsèques aux activités métiers.
- Identifier les actions critiques.
- Définir une représentation facile à intégrer dans des plateformes logicielles.

Notre profil permet de spécifier une politique RBAC dans une vue dynamique afin de compléter la spécification statique SecureUML de cette même politique. Cependant, il est nécessaire d'éviter les incohérences entre les deux modèles car ces incohérences peuvent être une source de failles dans les systèmes (Torre *et al.*, 2014). Par exemple, un rôle doit avoir les mêmes droits d'accès dans les vues statique et dynamique. Ce travail propose l'élaboration d'un ensemble de règles au moyen du langage OCL (OCL2, 2012) pour assurer la cohérence entre les diagrammes d'activité concrète et les modèles SecureUML. Ces règles assurent, d'une part, que la spécification SecureUML ne bloque pas la réalisation d'une activité par un utilisateur légitime, et d'autre part, que chaque activité concrète respecte les interdictions de la spécification statique

SecureUML. Nous considérons la cohérence inter-modèles qui vérifie la conformité des rôles et des contraintes contextuelles dans les différentes vues.

Notre métamodèle et les différentes règles de cohérence associées à ses éléments sont mis en œuvre dans un outil. Cet outil sera utilisé pour évaluer les différentes règles de cohérence sur les modèles et corriger toutes les contradictions entre les modèles SecureUML et nos modèles d'activité concrète.

Dans la deuxième section, nous présentons SecureUML, le point de départ de notre approche, ainsi que l'exemple de planification de réunions, qui illustre cet article. La section 3 discute l'expression des permissions et contraintes d'autorisation sous forme de préconditions sur les activités d'un processus métier. La section 4 définit le métamodèle qui permet d'étendre le diagramme d'activité avec les concepts de SecureUML. Elle présente aussi un ensemble de règles qui assurent la cohérence entre les diagrammes de deux vues. Dans la section 5, ces règles seront mises en œuvre dans un outil et vérifiées au moyen d'un ensemble de cas de test. Ainsi, cette section discute les erreurs de cohérence détectées dans les modèles d'activité concrète et les modèles SecureUML à l'aide de cet outil. La section 6 compare notre approche avec les travaux similaires qui traitent le contrôle d'accès au niveau des workflows d'un processus métier. Enfin, la dernière section conclut notre travail et présente quelques perspectives.

Cet article est une version étendue de (Chehida *et al.*, 2015), qui définit des règles de cohérence entre les vues statiques et dynamiques de contrôle d'accès et donne des éléments de leur validation.

2. SecureUML

UML est une notation graphique standard de l'OMG (Object Management Group) utilisée pour l'analyse des besoins et la conception d'un système. Le concept de « profil » permet l'extension des diagrammes d'UML pour spécifier un aspect particulier d'un système. 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 AuthUML (Alghathbar, 2012) qui propose des extensions du diagramme de cas d'utilisation et UMLsec (Jurjens, 2004) qui présente un profil pour étendre le diagramme d'activité.

Dans notre approche, nous utilisons le profil SecureUML (Basin *et al.*, 2006) (Basin *et al.*, 2009) qui permet de représenter une politique de sécurité basée sur le modèle RBAC dans une vue statique (voir figures 3 et 4). Ce diagramme utilise des permissions, représentées par des classes associatives, pour exprimer les règles de contrôle d'accès. Une permission est liée à une classe stéréotypée par «Role » qui représente les utilisateurs affectés au rôle, et une autre classe stéréotypée par «Entity» qui représente la classe cible de la permission. La permission spécifie ainsi les droits d'accès des utilisateurs à l'entité qu'elle protège. Elle est définie par un ensemble d'attributs indiquant les types d'actions autorisées (lecture, modification, etc).

Ces attributs sont définis par trois propriétés : des stéréotypes pour préciser le type de la ressource à protéger, le nom de l'attribut qui détermine la ressource protégée, et le type de l'attribut pour spécifier l'action autorisée par la permission. Il est possible de soumettre cette permission à des conditions contextuelles, appelées contraintes d'autorisation.

2.1. Exemple illustratif : organisation de réunions

Afin d'illustrer notre travail, nous allons utiliser l'exemple de l'organisation de réunions, proposé initialement par (Feather *et al.*, 1997). Ce système s'adresse à deux types d'utilisateurs : les « initiateurs » qui planifient des réunions et les « participants » qui répondent aux invitations des initiateurs.

2.1.1. Besoins fonctionnels

Dans un premier temps, nous représentons les différents besoins fonctionnels du système en utilisant le diagramme de cas d'utilisation de la figure 1. Ce dernier exprime les différentes façons dont les acteurs peuvent utiliser le système. L'acteur *Initiator* peut créer des réunions, inviter des participants et suivre leurs réponses. L'acteur *Participant* répond aux invitations et suit les confirmations des réponses.

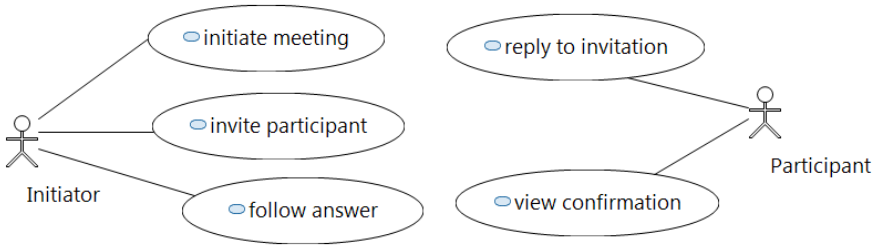

Figure 1. Le diagramme des cas d'utilisation du système d'organisation de réunions

Le système d'information permet d'enregistrer les données des personnes (participants et initiateurs), des invitations, des réunions et des propositions de changement, ainsi que les liens entre ces données. La figure 2 montre le diagramme de classes du système.

2.1.2. Besoins de sécurité

Afin de protéger les données de l'application, le système d'organisation de réunions applique une politique de contrôle d'accès, basée sur le modèle RBAC, qui répond à deux exigences de sécurité :

– La confidentialité d'une réunion, en assurant que seuls l'initiateur et les participants sont au courant de la réunion.

– L'intégrité d'une réunion, en assurant que seul l'initiateur peut modifier une réunion.

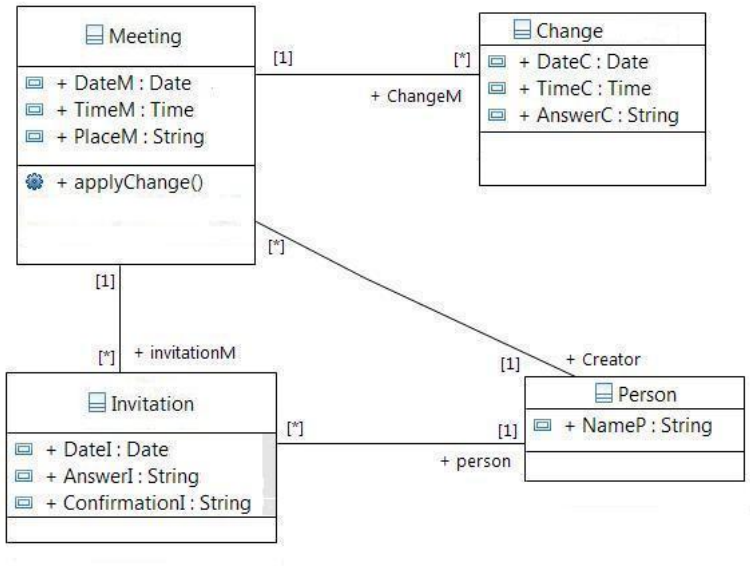

Figure 2. Le diagramme de classes du système d'organisation de réunions

Pour satisfaire ces exigences, la politique RBAC définit deux cibles de sécurité : les classes *Meeting* et *Invitation*. Elle se limite aux données les plus sensibles qui sont constituées par les données des réunions et des invitations. La politique de sécurité autorise un initiateur à créer des réunions et à lire et modifier leurs informations. Elle autorise aussi les participants d'une réunion à lire ses informations. Concernant les invitations, la modification de leurs informations est autorisée aux initiateurs créateurs des réunions concernées par ces invitations alors que la lecture est réservée aux participants invités.

2.1.3. Modèles SecureUML du système d'organisation de réunions

La figure 3 présente la spécification SecureUML de la politique de contrôle d'accès à la classe *Meeting*, elle comprend trois permissions. La première *CreateMeeting* spécifie que seul un initiateur peut créer des réunions. La deuxième *InitiatorMeeting* exprime qu'une réunion ne peut être lue ou modifiée que par un initiateur, pour autant qu'il soit le créateur de cette réunion. Cette restriction est exprimée par la contrainte d'autorisation associée à la permission. La dernière permission *ParticipantMeeting* exprime que les participants peuvent également lire les informations des réunions, pour autant qu'ils fassent partie des personnes invitées à la réunion. Les deux permissions *InitiatorMeeting* et *ParticipantMeeting* sont attachées à une contrainte d'autorisation exprimée en OCL (OCL2, 2012). Nous utilisons le mot clé *Caller* du type *String* dans les expressions OCL pour faire référence au nom de l'utilisateur. Souvent ces

contraintes font le lien entre les informations issues du modèle de sécurité (comme l'utilisateur et ses rôles) et l'état du modèle fonctionnel.

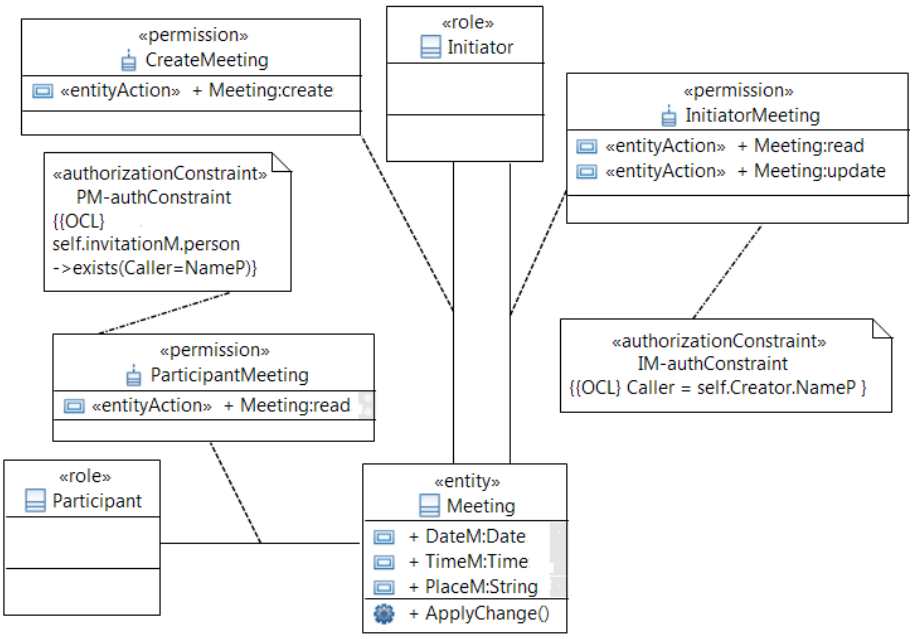

Figure 3. Le modèle SecureUML de contrôle d'accès à la classe Meeting

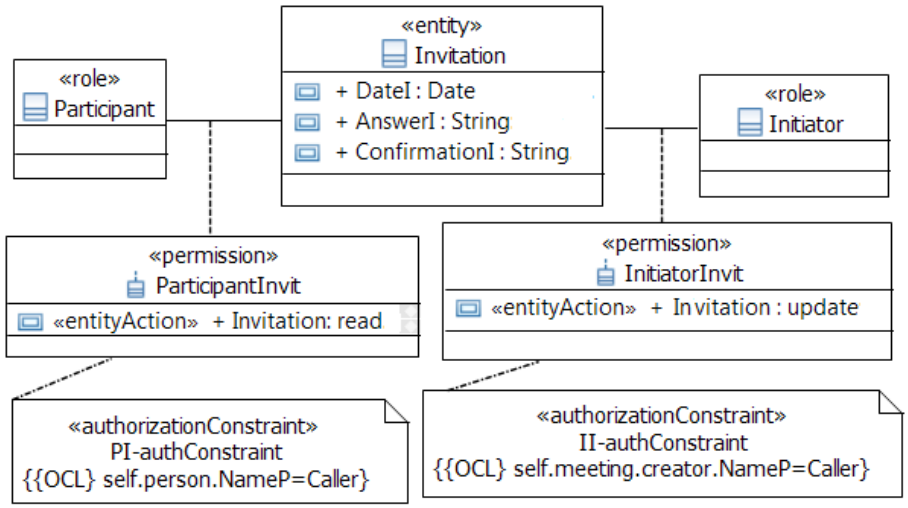

Figure 4. Le modèle SecureUML de contrôle d'accès à la classe Invitation

Le modèle SecureUML de la figure 4 exprime le contrôle d'accès à la classe *Invitation*. Cette classe est protégée par deux permissions. La première *InitiatorInvit*

autorise un utilisateur assigné au rôle *Initiator* à modifier les informations d'une invitation pour autant qu'il soit le créateur de la réunion concernée par cette invitation. La deuxième *ParticipantInvit* exprime que seul un utilisateur assigné au rôle *Participant* peut lire les invitations. La contrainte d'autorisation rattachée à cette permission restreint la lecture à la seule personne invitée.

2.2. Contrôle d'accès aux opérations

Les actions de type «EntityAction» représentent des opérations abstraites qui donnent lieu à des appels d'opérations concrètes réalisables sur des classes fonctionnelles. Dans le cadre de notre exemple, les actions autorisées au travers des permissions se traduisent comme suit :

1) Les permissions associées à la classe *Meeting* (figure 3) :

– Modification de la classe Meeting («entityAction» Meeting:update) : permet d'appeler les setters d'attributs (DateM, PlaceM et TimeM), ainsi que les setters des extrémités d'associations : Creator, invitationM, et ChangeM ; et permet d'invoquer l'opération *applyChange()* qui est une opération de modification.

– Creation de la classe Meeting («entityAction» Meeting:create) : permet l'appel au constructeur d'instances de la classe Meeting.

– Lecture de la classe Meeting («entityAction» Meeting:read) : permet l'appel à toutes les opérations de lecture d'attributs et d'extrémités d'association de la classe Meeting.

2) Les permissions associées à la classe *Invitation* (figure 4) :

– Modification de la classe Invitation («entityAction» Invitation:update) : fait appel aux setters d'attributs (DateI, AnswerI et ConfirmationI), ainsi qu'aux setters des extrémités d'associations : person et meeting.

– Lecture de la classe Invitation («entityAction» Invitation:read) : fait appel à toutes les opérations de lecture d'attributs et d'extrémités d'association de la classe *Invitation*.

Nous considérons comme « critique » toute opération d'une classe protégée qui correspond à une action d'une permission. La contrainte d'autorisation associée à la permission exprime une condition obligatoire pour la réalisation des opérations critiques.

3. Contrôle d'accès aux activités

Contrôler l'accès au niveau des workflows, selon (Kandala, Sandhu, 2002) consiste à assigner aux utilisateurs, conformément aux règles de l'organisation, des permissions pour effectuer certaines tâches au sein de l'organisation en fonction de leurs

qualifications et responsabilités. Cette section montre comment exprimer les permissions sous forme de préconditions associées aux opérations critiques et comment les propager sur les activités d'un processus métier. Cela se fait en trois phases : la première consiste à décrire chaque cas d'utilisation par une activité abstraite, la deuxième permet de spécifier une activité abstraite par une activité concrète et la dernière complète ce diagramme d'activité pour contrôler l'accès à ses actions.

3.1. Activité abstraite

Dans (WFMC, 1999), un workflow est défini comme étant *l'automatisation totale ou partielle d'un processus d'entreprise, au cours duquel on échange d'un participant à un autre, des documents, des informations ou des tâches pour action, et ce selon un ensemble de règles procédurales*. Le diagramme d'activité d'UML 2 (UML2, 2011) fournit un langage de modélisation des workflows qui est utilisé pour décrire le déroulement d'un cas d'utilisation. Plusieurs travaux comme (Russell *et al.*, 2006) et (Sarshar, Loos, 2007) ont traité l'adéquation des diagrammes d'activité d'UML2 pour la modélisation des processus métiers. Ce modèle est composé des nœuds d'activité, des nœuds d'action, des nœuds d'objet et des nœuds de contrôle. Ces nœuds sont reliés par deux types d'arcs pour représenter les flux de contrôle et de données. L'élément principal d'un diagramme d'activité est l'activité. Son comportement est défini par une coordination d'actions.

Plusieurs auteurs, comme (Roques, 2006), ont spécifié un cas d'utilisation par un ensemble de séquences d'actions qui représentent des tâches exécutées par des acteurs et qui produisent un résultat observable. Les tâches sont des opérations abstraites qui permettent d'une part de bien visualiser le comportement d'un cas d'utilisation et d'autre part de communiquer facilement et précisément avec les acteurs du SI. Elles représentent des actions ou des réactions de la part d'un acteur du système pour réaliser un comportement encapsulé dans un cas d'utilisation. Les actions d'un diagramme d'activité peuvent être utilisées pour modéliser les tâches d'un processus métier (Strembeck, Mendling, 2011).

Une activité abstraite permet l'abstraction de la manière dont collaborent des individus pour réaliser un cas d'utilisation. Elle spécifie un cas d'utilisation par un ensemble de tâches en coordination, représentées par des nœuds d'action, pouvant être exécutées par les acteurs du cas d'utilisation. Les tâches sont l'unité de description de déroulement de l'activité abstraite. Une activité abstraite peut montrer plusieurs scénarios d'exécution. Chaque scénario représente une succession particulière de tâches, s'exécutant du début à la fin de l'activité abstraite. La figure 5 montre un diagramme d'activité qui décrit les deux cas d'utilisation *reply to invitation* et *follow answer* de la figure 1 par deux activités abstraites. Les activités sont placées dans des partitions séparées qui précisent les acteurs responsables de ces tâches.

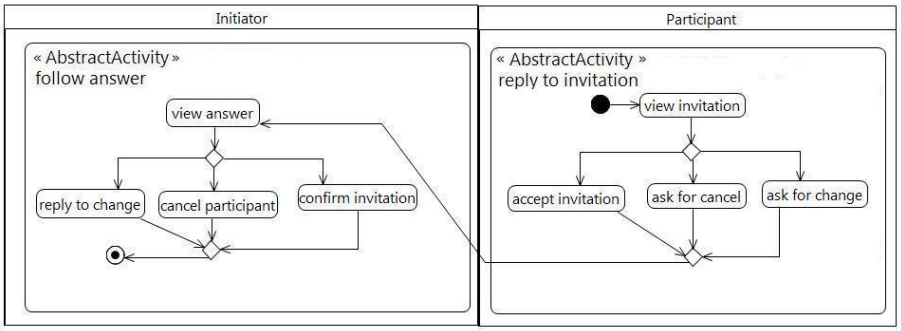

Figure 5. Activités abstraites spécifiant les cas d'utilisation « reply to invitation » et « follow answer »

3.2. Activité concrète

Les activités concrètes sont construites à partir des activités abstraites. Les activités abstraites décrivent le système comme une boîte noire sans détailler les objets qui le composent. Une fois qu'on dispose d'un diagramme de classes, ces diagrammes abstraits peuvent être précisés en définissant les objets et les opérations qui les réalisent. Une activité concrète raffine une activité abstraite en spécifiant le comportement de ses différentes tâches par une coordination d'un ensemble d'actions concrètes qui font référence à des opérations sur les objets des classes. L'exécution de chaque tâche fait appel à une ou plusieurs opérations concrètes. Une opération peut être, par exemple, une affection de valeurs à des attributs, un accès à la valeur d'une propriété structurelle (attribut ou terminaison d'association), la création d'un nouvel objet ou lien, etc. Chacune des tâches *view answer* et *reply to change* de l'activité abstraite *follow answer* (la partie gauche de la figure 5) est décomposée en une coordination d'actions concrètes dans la figure 6. Les tâches *confirm invitation* et *cancel participant* font chacune appel à une seule action concrète.

Les actions concrètes appellent une opération d'une instance de la classe. Ces instances sont définies comme des paramètres de l'activité concrète. Ceux-ci fournissent des entrées nécessaires à l'exécution des actions. Les instances M, I et C des entités (Meeting, Invitation et Change), dans la figure 6, sont indispensables pour l'exécution de l'activité *Follow answer*. Une activité concrète regroupe une famille de scénarios pouvant être exécutés par un utilisateur représentant l'acteur du cas d'utilisation décrit par l'activité. Chaque scénario d'une activité concrète représente une succession particulière d'actions concrètes s'exécutant du début à la fin de l'activité.

3.3. Précondition de contrôle d'accès

La séparation des préoccupations encourage à spécifier indépendamment les aspects fonctionnels et sécuritaires d'un système d'information. Cependant, leur inter-

action doit être prise en considération au niveau des diagrammes d'activité concrète. Cette section présente notre approche de spécification d'une politique RBAC au niveau des activités métiers concrètes. Cela se fait en deux niveaux : le premier consiste à affecter l'activité concrète à un ensemble de rôles et le deuxième permet d'associer certaines actions concrètes à des préconditions contextuelles. Dans le deuxième niveau, nous allons montrer comment dans la réalisation d'un processus métier, les permissions du diagramme SecureUML seront prises en compte. Les permissions et leurs éventuelles contraintes d'autorisation sont exprimées par des préconditions dans le diagramme d'activité de la figure 6.

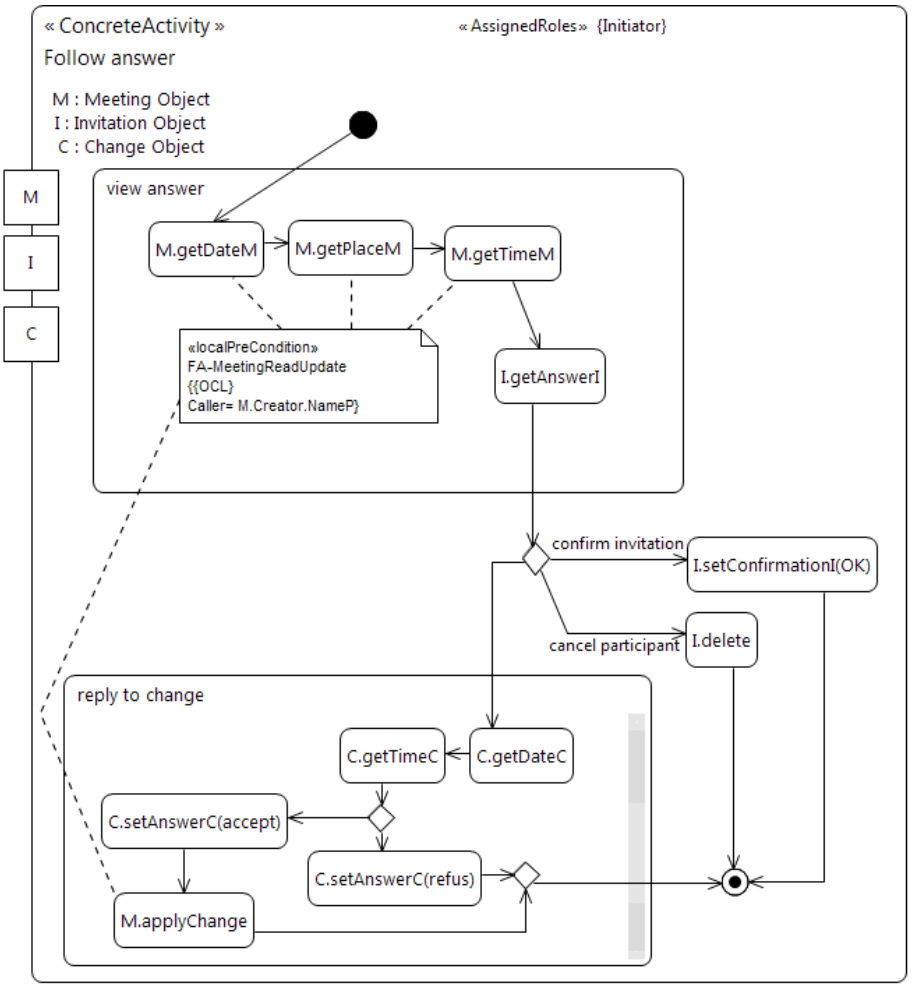

Figure 6. Contrôle d'accès à l'activité concrète Follow answer

3.3.1. Affectation des rôles aux activités

Une activité concrète est affectée à un ou plusieurs rôles. Seuls les utilisateurs assignés à ces rôles peuvent exécuter les tâches et les actions concrètes de l'activité. L'affectation des rôles aux activités est formalisée au moyen d'une précondition d'activité stéréotypée par «AssignedRoles». La précondition d'activité exprime le fait que toutes les actions doivent être exécutées par le même utilisateur dans un rôle donné. Dans la figure 6, celle-ci impose que l'activité soit exécutée par un utilisateur associé au rôle *Initiator*. Ici, elle traduit le fait que le cas d'utilisation *follow answer* est exécuté par l'acteur *Initiator* dans le diagramme des cas d'utilisation (figure 1). Nous considérons donc les acteurs associés aux cas d'utilisation comme des rôles.

3.3.2. Précondition locale

Certaines actions de l'activité concrète impliquent des opérations critiques dont les permissions sont associées à des contraintes d'autorisation. Il faut tenir compte de ces contraintes d'autorisation dans la définition de l'activité. Pour ce faire, des conditions supplémentaires, aussi appelées préconditions locales, sont exprimées sous la forme de contraintes stéréotypées comme «localPreCondition» et reliées aux actions concrètes. Ce sont des contraintes contextuelles exprimées en OCL comme précondition d'action et qui permettent l'usage de ces actions en fonction d'informations issues du diagramme de classes. Elles permettent d'empêcher certains utilisateurs assignés aux rôles de l'activité concrète d'exécuter les actions concrètes protégées. Elles font référence aux contraintes d'autorisation qui portent sur ces actions dans le diagramme SecureUML.

Dans la figure 6, la précondition locale *FA-MeetingReadUpdate* est associée aux quatre actions *M.getDateM*, *M.getPlaceM*, *M.getTimeM* et *M.applyChange*. Elle garantit que l'utilisateur qui exécute ces actions est le créateur de la réunion : *Caller* = *M.Creator.NameP*, ce qui correspond à la contrainte d'autorisation de la permission *InitiatorMeeting* dans la figure 3.

L'assignation des rôles à une activité et les préconditions locales définissent dans quelles conditions les actions concrètes doivent être réalisées. On peut également les voir comme des gardes qui sont évaluées lors de l'exécution du diagramme et qui garantissent que la politique de contrôle d'accès est bien respectée. Pour la figure 6, on peut se contenter de vérifier que l'utilisateur a un des rôles prévus en entrée de l'activité, et d'évaluer la précondition locale avant l'exécution de *M.getDateM* si les conditions suivantes sont respectées :

1. L'utilisateur n'utilise pas d'autres rôles que celui ou ceux prescrits par la précondition d'activité lors de l'exécution du processus.

2. L'utilisateur ne perd pas le droit d'utiliser ces rôles pendant l'exécution de l'activité.

3. La précondition locale reste vraie pendant l'exécution des quatre opérations concernées. Ceci signifie que les objets et associations concernées ne sont pas modifiés par les actions du diagramme d'activité ou par des activités extérieures qui seraient

menées en parallèles de ce diagramme.

Si ces conditions ne sont pas garanties, il est nécessaire de limiter les rôles de l'utilisateur, de vérifier les gardes avant chaque opération critique et de garantir l'exécution atomique de tout ou partie du diagramme.

4. Extension du méta-modèle des diagrammes d'activité

Notre travail vise à faire le lien entre les concepts de SecureUML et les diagrammes d'activité d'UML. Pour ce faire, nous définissons la sémantique des liens entre ces modèles au moyen de leurs méta-modèles respectifs. L'explicitation de ces liens permet, outre la définition de leur sémantique, de définir les contraintes qui garantissent la cohérence entre ces modèles.

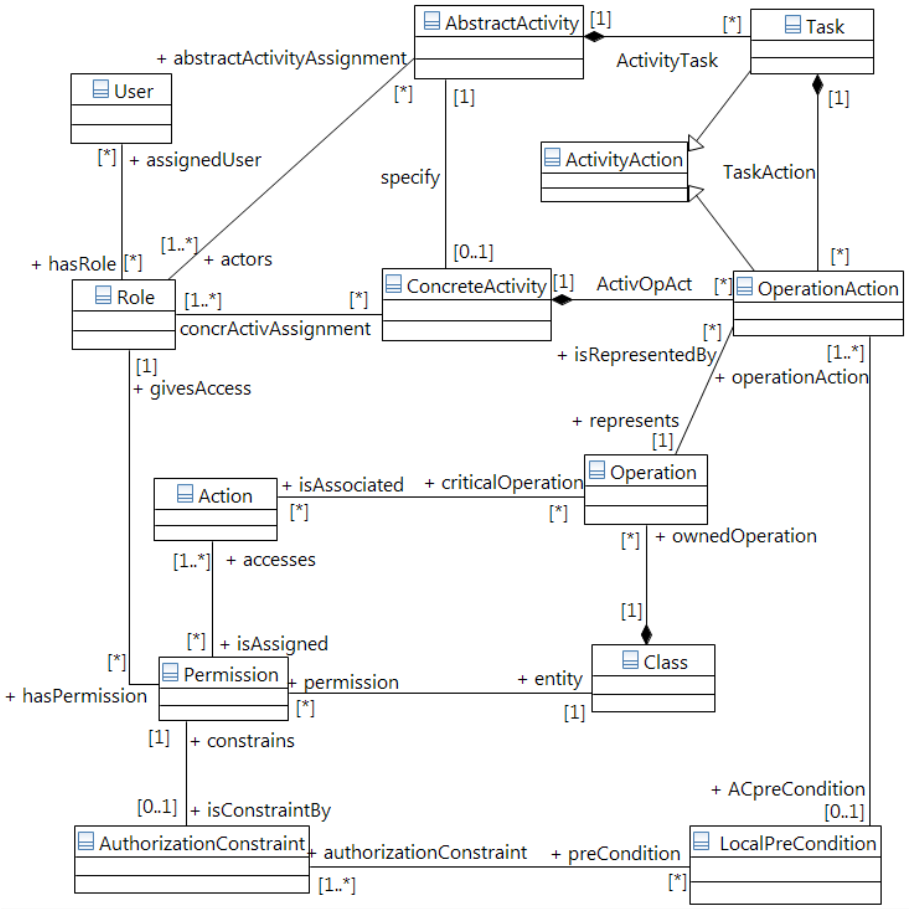

Figure 7. Métamodèle de contrôle d'accès aux activités

4.1. Liens entre méta-modèles

Le diagramme de la figure 7 représente les concepts de SecureUML en lien avec les notions d'activité abstraite et d'activité concrète. Les concepts de SecureUML sont : *User, Role, Permission, Action,* et *AuthorizationConstraint*. Les concepts inhérents aux activités sont : *AbstractActivity, ConcreteActivity, Task, OperationAction* et *LocalPreCondition*.

Une activité concrète est enclenchée par un ou plusieurs rôles et est nécessairement issue d'une activité abstraite. Dans notre démarche, les rôles associés à une activité concrète correspondent aux acteurs définis dans l'activité abstraite qu'elle spécifie. Nous établissons l'invariant suivant pour garantir un usage de rôles adéquat durant le processus de spécification allant des activités abstraites aux activités concrètes.

Context ConcreteActivity inv RoleSpecification :
self.concrActivAssignment → asSet() =self.specify.actors→asSet()

L'activité abstraite est réalisée par un ensemble de tâches (méta-classe Task) et l'activité concrète est réalisée par un ensemble d'actions (méta-classe OperationAction) qui correspondent à des appels d'opérations fonctionnelles. Nous considérons que ces actions sont contenues dans les tâches réalisant l'activité abstraite. Ce lien de composition permet de garantir une certaine traçabilité entre les activités abstraites et concrètes tout en étant en relation avec les opérations fonctionnelles.

Les actions d'une activité concrète (OperationAction) font appel à deux types d'opérations : les opérations critiques auxquelles une ou plusieurs permissions sont associées dans le modèle SecureUML (comme par exemple *getDateM()* de la classe *Meeting*) et les opérations non critiques (comme par exemple *getTimeC()* de la classe *Change*). Pour les opérations critiques, nous projetons les éventuelles contraintes d'autorisation qui leur sont associées, sous forme de préconditions des actions concrètes (OperationAction). De ce fait, l'appel d'une opération fonctionnelle dans une activité concrète devient préconditionné par la satisfaction de contraintes d'autorisation.

4.2. Règles de cohérence

Allaki et al (Allaki *et al.*, 2015) 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. D'autres approches comme (Gogolla *et al.*, 2009) et (Chiorean *et al.*, 2004) proposent l'utilisation du langage OCL. Il s'agit ici d'exprimer les règles de cohérence directement en OCL sur le méta-modèle d'UML. Notre choix dans ce travail porte sur cette approche. Elle est basée sur un langage de contraintes riche, intégré dans UML et supporté par plusieurs outils. Cette sous-section spécifie un ensemble de règles exprimées en OCL et

qui permettent de vérifier la cohérence entre nos modèles d'activité et les modèles SecureUML.

4.2.1. Quelle cohérence ?

(Elaasar, Briand, 2004) présente une classification de différents types de cohérence dans les modèles UML. Il considère selon la classification de (Kuzniarz, Staron, 2003) et de (Huzar *et al.*, 2005), que la cohérence est soit intra-modèles (appelée aussi horizontale), i.e. une propriété d'un modèle affirmant sa conformité syntaxique et sémantique, ou soit inter-modèles (également appelée verticale), entre deux modèles en relation mais de types différents.

Dans ce travail, nous considérons la cohérence verticale entre le modèle d'activités et le modèle SecureUML. Les mêmes éléments comme par exemple des rôles ou des contraintes contextuelles peuvent apparaître dans les deux modèles. Cela peut introduire des contradictions. Par exemple, la même opération fonctionnelle peut être bloquée par le modèle SecureUML et autorisée par les modèles d'activités. Pour éviter ces contradictions, nous spécifions au moyen d'expressions OCL les règles de cohérence que les instances de notre méta-modèle doivent respecter.

– La cohérence des rôles assure que tous les rôles associés à une activité concrète ont, dans le diagramme SecureUML, les permissions d'exécuter toutes les actions de l'activité.

– La cohérence des contraintes contextuelles assure que les préconditions locales de contrôle d'accès associées à une action d'une activité concrète font référence aux contraintes d'autorisation adéquates dans les modèles SecureUML.

À la section 4.2.2, nous définissons un ensemble de requêtes qui seront utilisées pour exprimer les règles de cohérence. Ensuite, nous introduisons les règles de cohérence dans les sections 4.2.3 et 4.2.4.

4.2.2. Requêtes OCL

isCritical() indique si une opération donnée est critique. Une opération critique est une opération fonctionnelle associée à des permissions SecureUML.

Context Operation::isCritical():Boolean
body: $self.isAssociated \rightarrow notEmpty()$

criticalOps() permet de récupérer l'ensemble des opérations critiques pour une activité concrète. Par exemple, les opérations critiques appelées par les actions de l'activité *Follow answer* de la figure 6 sont : *getDateM(), getPlaceM(), getTimeM(), applyChange(), getAnswerI() et setConfirmationI()*. Les autres opérations ne sont pas associées à des permissions SecureUML.

Context ConcreteActivity::criticalOps():Set(Operation)
body: $self.ActivOpAct.represents \rightarrow select(o:Operation \mid o.isCritical())$

isGuardedAction() vérifie si une action d'activité concrète est associée à une précondition locale de contrôle d'accès. Par exemple : les actions gardées de l'activité *Follow answer* de la figure 6 sont : *M.getDateM, M.getPlaceM, M.getTimeM et M.applyChange*.

Context OperationAction::isGuardedAction():Boolean
body: not self.ACpreCondition→ isEmpty()

OAPermission() permet de récupérer les permissions SecureUML associées à une action d'activité concrète. Ce sont les permissions associées à l'opération critique appelée par cette action et qui sont affectées aux rôles de l'activité concrète incluant l'action.

Chacune des quatre actions *M.getDateM, M.getPlaceM, M.getTimeM et M.applyChange* , par exemple, est associée à la permission *InitiatorMeeting*.

Context OperationAction::OAPermission():Set(Permission)
body: self.ActivOpAct.concrActivAssignment.hasPermission → asSet()
→ intersection(self.represents.isAssociated.isAssigned → asSet())

4.2.3. Conformité des rôles

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 *ActivityMissingPermissionToLegitimateUser* ci-dessous permet de garantir que tous les rôles autorisés à enclencher une activité concrète disposent de permissions dans le modèle SecureUML leur donnant le droit de réaliser toutes les opérations critiques de l'activité. Dans la section 5.2, nous présentons un exemple de violation de cet invariant. Le rôle *Initiator* affecté à l'activité concrète *Follow answer* de la figure 6 n'est pas autorisé à exécuter toutes les opérations critiques associées aux permissions dans le modèle SecureUML de la figure 4. Par exemple, ce rôle n'a pas la permission d'exécuter l'opération *getAnswerI()*.

Context ConcreteActivity inv ActivityMissingPermissionToLegitimateUser :
self.concrActivAssignment.hasPermission → notEmpty() implies
self.concrActivAssignment
→ forAll(r : Role | r.hasPermission.accesses.criticalOperation
→ includesAll(self.criticalOps()))

Il faut noter que cet invariant impose que le plus faible rôle associé à l'activité ait les droits sur toutes les permissions. Dans nos travaux futurs, nous étudierons la possibilité de relâcher cette contrainte en s'assurant que l'utilisateur ait des droits pour chaque opération, mais en lui permettant d'utiliser des rôles différents et complémentaires. Il serait également intéressant d'exprimer explicitement le changement de rôle d'un utilisateur lors de l'exécution d'une activité concrète.

Cependant, l'invariant précédent a été défini dans le contexte d'une activité concrète. Si il n'est pas vérifié, il identifiera l'activité erronée mais pas l'opération qui pose problème. Dès lors, nous introduisons une nouvelle contrainte, dans le contexte d'une action d'activité concrète qui nous permettra d'identifier les actions non permises au rôle. L'invariant *OperationActionMissingPermissionToLegitimateUser* permet d'assurer la conformité de rôles au niveau de chaque action d'activité concrète. Il permet d'assurer que l'ensemble des rôles autorisés à exécuter une action d'activité concrète est inclus dans l'ensemble des rôles autorisés par des permissions SecureUML à exécuter l'opération critique appelée par cette action. La section 5.2 montre un cas de violation de cet invariant. L'action *I.getAnswerI* de l'activité concrète *Follow answer* dans la figure 6 ne dispose pas d'une permission dans le modèle SecureUML de la figure 4 pour être exécutée par un utilisateur assigné au rôle *Initiator*.

Context OperationAction inv OperationActionMissingPermissionToLegitimateUser:
self.represents.isCritical() implies
self.represents.isAssociated.isAssigned.givesAccess
→ includesAll(self.ActivOpAct.concrActivAssignment)

4.2.4. Conformité des contraintes d'autorisation

Une action gardée d'une activité concrète est associée à une ou plusieurs permissions SecureUML. Il faut s'assurer que les contraintes d'autorisation associées à ces permissions sont reliées à la précondition de contrôle d'accès locale gardant cette action d'activité. Cela permet aux rôles associés à l'activité concrète d'exécuter cette action. Nous avons défini trois invariants qui permettent de souligner progressivement tous les types d'incohérences au niveau des contraintes contextuelles.

L'invariant *MissingPermissionAuthorizationConstraint* ci-dessous permet de garantir que toute action gardée d'une activité concrète est associée à au moins une permission rattachée à une contrainte d'autorisation et associée à un des rôles de l'activité. Par exemple, à la figure 6, si l'action *C.getDateC* était rattachée à la précondition locale *FA-MeetingReadUpdate*, cet invariant serait violé, car l'action n'est associée à aucune permission et donc à aucune contrainte d'autorisation.

Context OperationAction inv MissingPermissionAuthorizationConstraint:
self.isGuardedAction() implies self.OAPermission()
→ exists(P : Permission | P.isConstraintBy → notEmpty())

L'invariant *NotConformanceOfAuthorizationConstraint* ci-dessous permet de garantir que la précondition de contrôle d'accès locale gardant une action d'activité concrète fait référence à toutes les contraintes d'autorisation des permissions associées à cette action, et à aucune autre contrainte. Par exemple, cette contrainte serait violée si l'action *I.setConfirmationI(OK)* de la figure 6 était rattachée à *FA-MeetingReadUpdate*, qui ne correspond pas à la contrainte d'autorisation *II-authConstraint* de la figure 4.

Context OperationAction inv NotConformanceOfAuthorizationConstraint :
self.isGuardedAction() implies self.OAPermission().isConstraintBy → asSet()
=self.ACpreCondition.authorizationConstraint→asSet()

La contrainte suivante s'applique aux actions non gardées. L'invariant *MissingOperationActionLACPC* assure que les contraintes d'autorisation associées aux permissions sont bien propagées sur les préconditions locales rattachées aux actions d'activités concrètes concernées. Dans ce cas, toute action d'activité concrète non gardée ne doit pas être associée à des permissions rattachées à une contrainte d'autorisation. Un exemple de violation de cet invariant est présenté dans la section 5.2. Dans la figure 6, l'action *I.setConfirmationI(OK)* de l'activité concrète *Follow answer* n'est pas gardée par une précondition locale. Cependant, cette action est associée à la permission *InitiatorInvit* dans le modèle SecureUML de la figure 4 qui est rattachée à la contrainte d'autorisation *II-authConstraint*.

Context OperationAction inv MissingOperationActionLACPC:
not self.isGuardedAction() and self.represents.isCritical()
implies self.OAPermission().isConstraintBy→isEmpty()

5. Validation des règles de cohérence

Pour mettre en oeuvre ces règles de cohérence, nous avons utilisé l'outil Ecore d'Eclipse [2]. Cet outil permet de définir un métamodèle, d'en créer des instances et de vérifier les contraintes OCL sur ces instances. Ces contraintes sont exprimées dans le langage OCLinEcore [3]. Nous avons spécifié le métamodèle de la figure 7 ainsi que les différentes opérations et invariants OCL présentés précédemment. La spécification du métamodèle en Ecore permet ensuite de créer des instances à partir de modèles SecureUML et de modèles d'activité concrète. L'évaluation des différentes règles de cohérence est réalisée à partir de ces instances.

Notre outil permet de signaler tout type d'incohérence entre les instances correspondant aux modèles SecureUML et à nos modèles d'activité concrète.

Pour valider nos contraintes OCL, nous avons défini une centaine d'instances du méta-modèle, sur lesquelles nous avons évalué les contraintes. Suivant les conventions de (Legeard *et al.*, 2002), ces instances constituent des cas de test positifs et négatifs.

5.1. Test positif

Les tests positifs portent sur des instances cohérentes. Ils sont destinés à se terminer par un succès. Aucune de ces instances ne viole les propriétés invariantes spécifiées par les règles de cohérence.

2. http://www.eclipse.org/ecoretools/download.html
3. http://help.eclipse.org/luna/index.jsp?topic=\%2Forg.eclipse.ocl.doc\%2Fhelp\%2FOCLinEcore.html

Par exemple, le modèle SecureUML de la figure 3 (sans prendre en compte les permissions de la figure 4) et le modèle d'activité concrète *Follow answer* de la figure 6 présentent un exemple de cas de test positif.

5.2. Test négatif

Les tests négatifs permettent de voir la réaction de notre outil face à des instances incohérentes. Ils doivent se terminer par un message d'erreur qui signale les invariants violés. Pour chaque règle de cohérence, nous avons défini un ensemble de cas de test négatifs afin de tester si notre outil réagit vis-à-vis de toutes les propriétés invariantes. Cette validation a été complétée par des cas de test dont les instances ne sont pas conformes au métamodèle.

Le modèle SecureUML de la figure 4 et le modèle d'activité concrète *Follow answer* de la figure 6 présentent un cas d'incohérence. Après la création et la validation des instances correspondantes à ces modèles, les messages de la figure 8 sont affichés. Ils montrent trois cas de violations des invariants représentant les règles de cohérence.

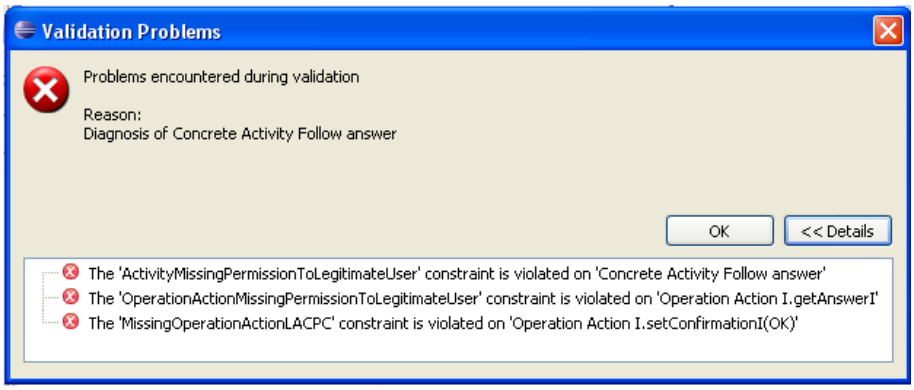

Figure 8. Exemple du test négatif (figure 4 et figure 6)

Le premier message correspond à l'invariant *ActivityMissingPermissionToLegitimateUser*. Comme nous l'avons vu dans la section 4.2.3, il indique que l'activité concrète *Follow answer* inclut au moins une action qui fait appel à une opération non autorisée par les permissions SecureUML de la figure 4 pour le rôle *Initiator*.

Le deuxième concerne l'invariant *OperationActionMissingPermissionToLegitimateUser* qui précise la contrainte précédente. Il identifie l'action non autorisée de l'activité *Follow answer* : il s'agit de *I.getAnswerI*. Comme nous l'avons vu dans la section 4.2.3, la permission *ParticipantInvit* de la figure 4 ne donne le droit qu'aux utilisateurs assignés au rôle *Participant* de lire les données de la classe *Invitation*. Cependant dans le modèle d'activité *Follow answer* de la figure 6, un utilisateur assigné au rôle *Initiator* doit lire la réponse de l'invitation (AnswerI). Dans ce cas l'opération *getAnswerI* est bloquée par la spécification statique SecureUML et autorisée par la spécification dynamique exprimée par notre profil.

Le dernier message correspond à l'invariant *MissingOperationActionLACPC* qui n'est pas respectée par l'action *I.setConfirmationI(OK)*. En effet, comme nous l'avons vu dans la section 4.2.4, la permission *InitiatorInvit* de la figure 4 est associée à une contrainte d'autorisation qui n'autorise que l'initiateur créateur de la réunion concernée par une invitation à modifier ses informations. Cependant dans le modèle de l'activité concrète *Follow answer* de la figure 6, aucune garde n'est associée à l'action *I.setConfirmationI(OK)*.

5.3. Bilan de la validation

La validation des règles OCL présentées dans la section 4 s'est basée d'une part sur la relecture attentive de ces contraintes et d'autre part sur leur expérimentation sur notre étude de cas.

Chacun des cinq diagrammes d'activité décrivant les cinq cas d'utilisation de la figure 1 a constitué un test positif. Ensuite nous avons introduit systématiquement des mutations de ces diagrammes pour falsifier chacun des cinq invariants. Pour chaque diagramme d'activité nous avons créé dix tests négatifs. Au total cinquante tests négatifs et cinq tests positifs on été réalisés. Leur caractère systématique nous donne confiance dans la correction des cinq invariants.

5.4. Mise au point de l'étude de cas

Les tests nous ont également permis de mettre au point l'exemple d'organisation de réunions. La figure 9 présente le modèle SecureUML de contrôle d'accès à la classe *Invitation* (figure 4) après correction. Deux actions ont été ajoutées à la permission *InitiatorInvit : «entityAction» Invitation:read* et *«entityAction» Invitation:delete* qui autorisent respectivement le rôle *Initiator* à lire ou à supprimer les invitations qu'il a créées. L'action *«MethodAction» setAnswerI():execute* ajoutée à la permission *ParticipantInvit* permet au rôle *Participant* de modifier la réponse de son invitation.

D'autres corrections sur les diagrammes d'activité concrète ont été réalisées. Pour l'activité *Follow answer*, les trois actions : *I.getAnswerI, I.setConfirmationI(OK) et I.delete* sont rattachées à la précondition locale de contrôle d'accès définie par l'expression OCL *I.meeting.creator.NameP=Caller* .

Notre étude de cas suggère une démarche méthodologique qui débute par la spécification de la vue statique d'une politique RBAC, exprimée au moyen de modèles SecureUML. Ensuite, des diagrammes d'activité concrète sont réalisés sans tenir compte de la politique de contrôle d'accès. La prise en compte de cette politique se fait alors en deux étapes : la première consiste à repérer les opérations critiques appelées par les actions du diagramme d'activité concrète. Comme nous l'avons vu dans la section 2.2, les opérations critiques sont celles correspondant aux actions des permissions SecureUML. La deuxième étape consiste à exprimer les éventuelles contraintes d'autorisation des permissions qui leur sont associées comme préconditions des actions d'activités concrètes appelant ces opérations critiques.

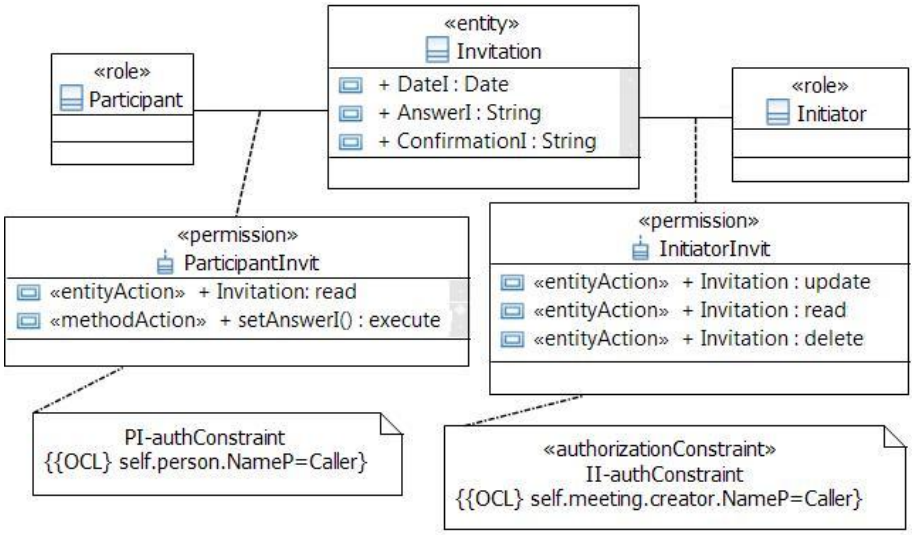

Figure 9. Correction du modèle SecureUML de contrôle d'accès à la classe Invitation

6. Travaux connexes

La prise en compte des besoins de sécurité dans les différentes phases de processus de développement est une préoccupation majeure. Plusieurs approches ont été proposées : nous citons, par exemple, les approches orientées but (comme KAOS étendu (Lamsweerde, 2007) et SecureTropos (Mouratidis, Giorgini, 2007)) et les approches basées sur les ontologies (comme (Souag *et al.*, 2015)). Ces approches s'intéressent aux exigences de sécurité et se situent en grande partie au niveau de l'activité d'analyse des besoins. Dans notre travail, nous ciblons les activités de conception et de mise en œuvre avec un niveau de détail plus important. En effet, notre objectif est de couvrir aussi bien une modélisation structurelle que comportementale d'une politique de contrôle d'accès.

Notre travail est basé sur des extensions d'UML, comme SecureUML (Basin *et al.*, 2009) et UMLsec (Jurjens, 2004). Cette approche définit des profils qui étendent les diagrammes d'UML pour spécifier une politique de sécurité répondant aux différentes exigences de sécurité. Nous proposons une extension de diagramme d'activité d'UML 2 pour exprimer une politique RBAC. Notre approche permet, d'une part, la spécification d'une politique de sécurité, basée sur le modèle RBAC, au niveau des activités d'un processus métier, et d'autre part, la vérification de la cohérence entre cette dernière et une spécification statique de politique RBAC exprimée au moyen de modèles SecureUML.

Plusieurs travaux comme (Ahn *et al.*, 2000), (Bertino *et al.*, 1999) et (Wainer *et al.*, 2003), ont proposé des solutions basées sur le modèle RBAC. Une partie de ces travaux ont étendu le modèle RBAC pour contrôler l'accès au niveau des workflows.

Ces extensions incluent par exemple : la spécification des politiques d'autorisation dynamiques (Ma *et al.*, 2011) et la délégation dynamique de tâches (Gaaloul, 2010) et (Wainer *et al.*, 2007). Dans notre travail nous ne cherchons pas à étendre RBAC, mais plutôt à combiner des formalismes approuvés et bien connus tels que SecureUML pour la conception de politiques RBAC et les diagrammes d'activité d'UML. L'objectif est de disposer d'un ensemble de modèles accessibles aux divers acteurs d'un développement de logiciels.

Plusieurs langages graphiques sont utilisés pour la modélisation des workflows d'un processus métier. Cependant BPMN (BPMN2, 2011) et UML (UML2, 2011) sont considérés comme les principales normes (Rodríguez *et al.*, 2007). (Geambasu, 2012) compare BPMN et le diagramme d'activité d'UML2 et conclut qu'ils sont équivalents pour représenter des processus de façon compréhensible et pour décrire des processus métiers. Notre choix dans cet article porte sur les diagrammes d'activité d'UML2 car ils incluent un support riche pour représenter les workflows : les actions, les flux de contrôle et de données ainsi que le choix, le parallélisme, les séquences et les événements.

Parmi les travaux de contrôle d'accès au niveau des workflows qui sont basés sur le diagramme d'activité d'UML 2 : (Jurjens, 2004) propose une extension du diagramme d'activité d'UML pour la spécification de politique RBAC. Il utilise un package stéréotypé par « rbac » avec trois tags : *protected*, *role*, et *right*. Ces tags définissent respectivement l'ensemble des actions protégées d'un diagramme d'activité, les utilisateurs assignés aux rôles exécutant l'activité et l'affectation des actions protégées aux rôles qui peuvent les utiliser. Par contre, cette extension ne permet pas de définir des contraintes d'autorisation. De plus, nous n'avons pas connaissance de l'existence d'un outil pour vérifier la cohérence entre les diagrammes. En outre, cette extension n'est pas définie en utilisant un méta-modèle. (Strembeck, Mendling, 2011) présente une approche basée sur un méta-modèle pour la modélisation des processus métiers comprenant une spécification RBAC basée sur les rôles, les hiérarchies de rôles, et la séparation des tâches. Une extension du modèle d'activité d'UML2 a été proposée pour illustrer l'approche. Cependant, cette extension ne prend pas en charge les contraintes contextuelles d'autorisation.

Notre travail propose une extension des diagrammes d'activité d'UML 2 pour garder l'accès des utilisateurs affectés à des rôles aux activités d'un processus métier en utilisant des préconditions de contrôle d'accès. Notre profil est basé sur un métamodèle qui intègre les éléments du métamodèle du diagramme d'activité et du métamodèle de SecureUML. Il permet, outre la spécification d'une politique RBAC, la prise en compte des contraintes contextuelles d'autorisation qui permettent de formaliser des politiques de contrôle d'accès dépendant des aspects dynamiques du système, tels que l'évolution des valeurs d'attribut. Notre approche considère ainsi la vue statique d'une politique RBAC dans le contrôle d'accès aux activités.

Plusieurs approches ont été proposées pour étendre BPMN en vue de traiter le contrôle d'accès au niveau des workflows d'un processus métier. Parmi ces approches, (Rodríguez *et al.*, 2007) présente une extension de BPMN qui supporte plusieurs as-

pects de sécurité (comme la non-répudiation, la détection des risques d'attaque, l'intégrité, ..). (Wolter, Schaad, 2007) définit une extension de BPMN pour la modélisation des contraintes d'autorisation basées sur les tâches. (Brucker *et al.*, 2012) propose SecureBPMN pour la spécification de politiques RBAC au niveau de processus métier. Les deux derniers travaux de Wolter et Brucker permettent l'intégration des contraintes TBSoD (*Task-based Separation of Duty*) et BoD (*Binding of Duty*). Ces contraintes considèrent l'ordre et l'historique d'une tâche dans une instance particulière de processus pour décider si un certain sujet ou rôle est autorisé à effectuer une certaine tâche (Strembeck, Mendling, 2011) (Botha, Eloff, 2001).

Dans le cadre de notre profil, un utilisateur peut effectuer plusieurs tâches de différentes activités qui devraient être effectuées par des utilisateurs distincts, par exemple pour éviter un conflit d'intérêt. Les travaux de Wolter et Brucker peuvent compléter nos modèles en spécifiant les contraintes TBSoD et BoD au niveau des activités. Les contraintes TBSoD empêchent un sujet d'effectuer deux activités ou deux tâches d'une activité en conflit d'intérêt. Les contraintes BoD définissent un ensemble d'activités ou de tâches qui doivent être exécutées par le même sujet. Les modèles concrets que nous proposons peuvent aussi compléter les travaux de Wolter et Brucker en vue de représenter le détail des tâches et des activités en actions concrètes avec leurs éventuelles préconditions de contrôle d'accès construites à partir des contraintes d'autorisation et de leurs relations avec les opérations fonctionnelles.

Peu de travaux ont abordé le problème de la cohérence entre les profils de contrôle d'accès. (Matulevicius, Dumas, 2011) présente des règles de transformation entre SecureUML et UMLsec. (Montrieux *et al.*, 2011) propose deux nouveaux profils pour la spécification de politique RBAC sur les diagrammes de classes et d'activité, et également un outil qui vérifie la cohérence entre eux. Dans notre travail, nous avons proposé des règles qui assurent la cohérence entre le profil que nous proposons pour spécifier le contrôle d'accès aux activités et SecureUML qui est un profil bien adapté pour spécifier une politique RBAC statique.

7. Conclusion et perspectives

Le contrôle d'accès est devenu un souci majeur dans le développement des SI. Plusieurs domaines de l'activité économique et sociale sont maintenant sujets à des lois et des normes très strictes au niveau de la sécurité des données (Milhau, 2011). Cet article propose une approche de spécification d'une politique d'autorisation RBAC au niveau du Workflow des activités d'un processus métier. Ceci est réalisé en utilisant des règles de contrôle d'accès, exprimées dans une vue statique via le profil SecureUML, pour contrôler l'accès des utilisateurs aux actions des activités.

La séparation entre les préoccupations fonctionnelles et sécuritaires est au coeur de notre approche. Cette séparation des préoccupations vise à maîtriser la complexité du système, et peut également se montrer utile quand il s'agit de réaliser la sécurisation d'un système d'information pré-existant. De plus, elle facilite l'évolution de la politique de contrôle d'accès. Dans notre approche, les aspects fonctionnels sont

exprimés dans les cas d'utilisation, le diagramme de classes et les diagrammes d'activité abstraits, et leur raffinement dans l'enchaînement des opérations concrètes. Les aspects sécuritaires sont exprimés statiquement, sous forme de permissions associées à des rôles, pour chaque classe protégée du diagramme de classes. À cet effet, nous nous basons sur le profil SecureUML. Les aspects sécuritaires sont ensuite exprimés dynamiquement par des préconditions associées localement ou globalement aux activités concrètes. En outre, les contraintes d'autorisation permettent des interactions entre les aspects fonctionnels et sécuritaires. Elles peuvent être spécifiées dans la vue statique et prendre la forme de gardes dans la vue dynamique.

Ce travail propose un métamodèle, associé à des contraintes OCL, qui précise les relations et les contraintes entre la vue statique de la politique de contrôle d'accès, et son expression dynamique dans les activités concrètes. Ce métamodèle doit encore évoluer pour permettre plus de flexibilité dans l'utilisation des rôles. Il sera alors possible de définir un outillage associé à un environnement UML2, qui vérifie la conformité de nos diagrammes par rapport à ce métamodèle. Nous avons, en outre, appliqué notre démarche sur le système d'organisation de réunions, et développé ainsi tous les cas d'utilisation de la figure 1. Les règles de contrôle d'accès ont ainsi été exprimées sur les différentes activités du système. Cette étude de cas nous a permis de valider par la pratique le présent travail et montrer l'intérêt d'exprimer des règles de contrôle d'accès au niveau des processus métiers. Des études de cas de plus grande taille seront étudiées dans de futurs travaux.

La démarche que nous proposons construit graduellement une politique de contrôle d'accès et identifie les gardes qui la mettront en oeuvre dynamiquement dans le déroulement des activités concrètes. Cette représentation est proche des scénarios d'exécution et facilite l'implémentation de la politique de sécurité dans des plateformes logicielles.

L'ensemble des règles de cohérence définies dans ce travail permet ainsi d'éviter toute contradiction entre cette représentation et une représentation statique de politique RBAC exprimée en SecureUML. Ces règles sont supportées par un outil. Elles sont illustrées par l'étude de cas du système d'organisation de réunions où elles ont permis de corriger les erreurs de cohérence détectées dans les modèles de contrôle d'accès des vues statique et dynamique.

La suite de ce travail sera principalement consacrée à la mise en oeuvre de cette approche et du méta-modèle associé dans des outils de validation. La plateforme B4MSecure [4] (Idani, Ledru, 2015), qui supporte une variante de SecureUML et permet la traduction de ces modèles vers des spécifications formelles exprimées en B (Abrial, 1996), est un candidat naturel pour cette mise en oeuvre. Cette plateforme supporte aujourd'hui la vue statique (diagramme de classes et permissions). Il s'agirait d'y intégrer les vues dynamiques (diagrammes d'activité abstraits et concrets), de vérifier la conformité de ces diagrammes à notre méta-modèle, et ensuite de traduire

4. http://b4msecure.forge.imag.fr/

ces vues dynamiques en B. La traduction de ces vues dynamiques fournira une base intéressante pour l'animation des spécifications B et la validation de l'ensemble des modèles.

Bibliographie

Abrial J.-R. (1996). *The B-book: assigning programs to meanings*. Cambridge University Press.

Ahn G., Sandhu R., Kang M., Park J. (2000). Injecting RBAC to secure a web-based workflow system. In *the 5th ACM Workshop on Role-Based Access Control*, p. 1-10. New York, NY, USA, Morgan Kaufmann Publisher.

Alghathbar K. (2012). Representing access control policies in use cases. *International Arab Journal of Information Technology*, vol. 9, n° 3.

Allaki D., Dahchour M., En-nouaary A. (2015). A new taxonomy of inconsistencies in UML models with their detection methods for better MDE. *International Journal of Computer Science and Applications*, vol. 12, n° 1, p. 48 – 65.

ANSI. (2004). Role based access control. *American national standard for information technology*, vol. 359, n° 2004, p. 1-47.

Basin D. A., Clavel M., Doser J., Egea M. (2009). Automated analysis of security-design models. *Information and Software Technology*, vol. 51, n° 5, p. 815-831.

Basin D. A., Doser J., Lodderstedt T. (2006). Model driven security: From UML models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology*, vol. 15, n° 1, p. 39-91.

Bertino E., Ferrari E., Atluri V. (1999). The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, vol. 2, n° 1, p. 65-104.

Botha R., Eloff J. (2001). Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, vol. 40, n° 3, p. 666–682.

BPMN2. (2011). *Business Process Modeling Notation (BPMN) Version 2.0*. Object Management Group. (http://www.omg.org/spec/BPMN/2.0/formal-11-01-03.pdf)

Brucker A. D., Hang I., Lückemeyer G., Ruparel R. (2012). SecureBPMN: Modeling and Enforcing Access Control Requirements in Business Processes. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, p. 123-126. New York, NY, USA, ACM.

Chehida S., Idani A., Ledru Y., Rahmouni M. (2015). Extensions du diagramme d'activité pour contrôler l'accès au SI. In *INFormatique des ORganisations et Systèmes d'Information et de Décision*, p. 151-165. Biarritz, france.

Chiorean D., Paşca M., Cârcu A., Botiza C., Moldovan S. (2004). Ensuring UML Models Consistency Using the OCL Environment. *Electronic Notes in Theoretical Computer Science*, vol. 102, p. 99-110.

Elaasar M., Briand L. (2004). *An overview of UML consistency management*. Rapport technique n° SCE-04-18. Carleton.

Feather M. S., Fickas S., Finkelstein A., Lamsweerde A. van. (1997). Requirements and Specification Exemplars. *Automated Software Engineering*, vol. 4, n° 4, p. 419-438.

Ferraiolo D., Kuhn D., Chandramouli R. (2003). *Role-Based Access Control*. Artech House.

Gaaloul K. (2010). *Une approche sécurisée pour la délégation dynamique de tâches dans les systèmes de gestion de workflow.* Thèse de doctorat. Nancy , France.

Geambasu C. (2012). BPMN vs. UML activity diagram for business process modeling. *Accounting and Management Information Systems*, vol. 11, n° 4, p. 637–651.

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

Huzar Z., Kuzniarz L., Reggio G., Sourrouille J. L. (2005). 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.

Idani A., Ledru Y. (2015). B for Modeling Secure Information Systems - the B4MSecure platform . In *The 17th International Conference on Formal Engineering Methods*, vol. 4907. Springer.

Jurjens J. (2004). *Secure systems development with UML.* Berlin, Heidelberg, Springer-Verlag.

Kandala S., Sandhu R. (2002). Secure Role-Based Workflow Models. In *Database and Application Security XV*, vol. 87, p. 45-58. Springer.

Kuzniarz L., Staron M. (2003). Inconsistencies in student designs. In *Workshop on Consistency Problems in UML-based software development II.*

Lamsweerde A. van. (2007). Engineering requirements for system reliability and security. *IOS Press*, vol. 9, p. 196-238.

Legeard B., Peureux F., Utting M. (2002). Automated boundary testing from Z and B. In *Fme'02, formal methods europe*, vol. 2391. Springer.

Ma G., Wu K., Zhang T., Li W. (2011). A flexible policy-based access control model for Workflow Management Systems. In *International IEEE Conference on Computer Science and Automation Engineering (CSAE)* , vol. 2, p. 533-537.

Matulevicius R., Dumas M. (2011). *Towards Model Transformation between SecureUML and UMLsec for Role-based Access Control* (vol. 224). IOS Press Ebooks.

Milhau J. (2011). *Un processus formel d'intégration de politiques de contrôle d'accès dans les systèmes d'information.* Thèse de doctorat. Paris, France. (page 2)

Montrieux L., Wermelinger M., Yu Y. (2011). Tool Support for UML-Based Specification and Verification of Role-Based Access Control Properties. In *8th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering.*

Mouratidis H., Giorgini P. (2007). Secure tropos: a security-oriented extension of the tropos methodology. *International Journal of Software Engineering and Knowledge Engineering*, vol. 17, n° 2, p. 285–309.

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

Rodríguez A., Fernández-Medina E., Piattini M. (2007). A BPMN Extension for the Modeling of Security Requirements in Business Processes. *IEICE - Transactions on Information and Systems*, vol. E90-D, n° 4, p. 745-752.

Roques P. (2006). *UML 2 par la Pratique*. Paris, Eyrolles.

Russell N., Aalst W. van der, Hofstede A. ter, Wohed P. (2006). On the suitability of UML 2.0 activity diagrams for business process modelling. In *3rd Asia-Pacific conference on Conceptual modelling*, p. 95-104.

Sarshar K., Loos V. (2007). Modeling the Resource Perspective of Business Processes by UML Activity Diagram and Object Petri Net. In *Enterprise Modeling and Computing with UML*, p. 204-215.

Souag A., Salinesi C., Mazo R., Comyn-Wattiau I. (2015). A security ontology for security requirements elicitation. In *Engineering secure software and systems - 7th international symposium, essos 2015, milan, italy, march 4-6, 2015*, p. 157–177.

Strembeck M., Mendling J. (2011). Modeling process-related RBAC models with extended UML activity models. *Information and Software Technology*, vol. 53, n° 5, p. 456-483. (Special Section on Best Papers from {XP2010})

Torre D., Labiche Y., Genero M. (2014). UML consistency rules: a systematic mapping study. In *18th International Conference on Evaluation and Assessment in Software Engineering*, p. 1-10. New York, NY, USA, ACM.

UML2. (2011). *Unified modeling language: Superstructure(version 2.4)*. Object Management Group. (http://www.omg.org/spec/UML/2.4/Superstructure/ptc-10-11-14.pdf)

Wainer J., Barthelmess P., Kumar A. (2003). W-RBAC.a workflow security model incorporating controlled overriding of constraints. *International Journal of Cooperative Information Systems*, vol. 12, n° 4, p. 455-486.

Wainer J., Kumar A., Barthelmess P. (2007). DW-RBAC: A formal security model of delegation and revocation in workflow systems. *Information Systems*, vol. 32, n° 3, p. 365-384.

WFMC. (1999). *Workflow management coalition Terminology and glossary*. Workflow Management Coalition. (http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf)

Wolter C., Schaad A. (2007). Modeling of Task-Based Authorization Constraints in BPMN. In *Business Process Management*, p. 64-79. Springer Berlin Heidelberg.