Extensions du diagramme d'activité pour contrôler l'accès au SI

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

1. *Université d'Oran Es-sénia*
   31000 Oran, Algérie
   Salim.Chehida@imag.fr
2. *Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France*
   CNRS, LIG, F-38000 Grenoble, France
   Akram.Idani@imag.fr
3. *Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France*
   CNRS, LIG, F-38000 Grenoble, France
   Yves.Ledru@imag.fr
4. *Université d'Oran Es-sénia*
   31000 Oran, Algérie
   kamel_rahmouni@yahoo.fr

RÉSUMÉ. *L'ouverture des organisations et de leurs Systèmes d'Information (SI) pose le problème de leur sécurité. La définition d'une politique de contrôle d'accès est l'une des principales étapes dans la conception 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. 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.*

ABSTRACT. *The evolution of organisations 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.*

MOTS-CLÉS : *RBAC, Workflow, Processus métier, SecureUML, Diagramme d'activité d'UML 2*
KEYWORDS: *RBAC, Workflow, Business process, SecureUML, UML2 Activity Diagram*

1. Introduction

Avec la croissance fulgurante que connaissent 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), Role-Based Access Control (RBAC) (Ferraiolo *et al.*, 2003) est le modèle de contrôle d'accès le plus répandu dans les systèmes informatiques. En RBAC, 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 sur les diagrammes d'UML. Ce profil étend le modèle RBAC en exprimant des contraintes contextuelles : les contraintes d'autorisation. Ces contraintes portent sur l'état spécifié par le diagramme de classes et conditionnent l'évaluation 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 vie de développement. Elle intégre 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 grandes 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'uti-

1. Flux de travail

lisation 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 et contraintes d'autorisation exprimées en SecureUML qui seront associées comme pré-conditions aux actions concrètes concernées.

Le présent travail vise l'expression d'une politique de contrôle d'accès 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 dans l'exécution des activités métiers.

– Localiser les actions critiques.

– Définir une représentation facile à intégrer et à transformer dans des plateformes logicielles.

Dans la deuxième section, nous présentons les travaux qui traitent du contrôle d'accès au niveau des workflows. La troisième section présente SecureUML, le point de départ de notre approche ainsi que l'exemple de planification de réunions, qui illustre cet article. La section 4 discute l'expression des permissions et contraintes d'autorisation sous forme de préconditions sur les activités d'un processus métier. La section 5 définit le métamodèle qui permet d'étendre le diagramme d'activité pour le contrôle d'accès. Enfin, la dernière section conclut notre travail et présente quelques perspectives.

2. Contrôle d'accès au niveau de Workflow

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"*. Ces règles sont nécessaires car elles conditionnent le bon fonctionnement du workflow. Dans le présent travail nous nous intéressons en particulier aux règles de sécurité issues de politiques de contrôle d'accès. 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. 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'activités d'UML. L'objectif en est de disposer d'un ensemble de modèles accessibles aux divers acteurs d'un développement de logiciels.

Trois langages graphiques sont largement utilisés pour la modélisation des workflows : BPMN (BPMN2, 2011), les Réseaux de Pétri (Murata, 1989) et le diagramme d'activité d'UML 2(UML2, 2011). (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 qui ont utilisé le diagramme d'activité d'UML 2 pour contrôler l'accès au niveau des workflows, (Jurjens, 2010) a proposé l'utilisation d'un package stéréotypé par « rbac » avec les trois tags protected, role, et right qui 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. (Strembeck, Mendling, 2011) ont défini un méta-modèle pour l'extension du diagramme d'activité vers le modèle RBAC.

3. 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. Elle 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 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, 2010) 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. 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 classe associative stéréotypée par «Permission» signifie que les utilisateurs assignés à «Role» sont autorisés, par la permission en relation, à accéder aux éléments de «Entity». Il est possible de soumettre cette permission à des conditions contextuelles, appelées conditions d'autorisation. Une permission SecureUML est définie par un ensemble d'attributs. Chaque attribut permet d'assigner une action à la permission et est défini par trois propriétés : des stéréotypes pour définir 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.

3.1. 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" planifient des réunions et les "participants" répondent aux invitations des initiateurs.

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.

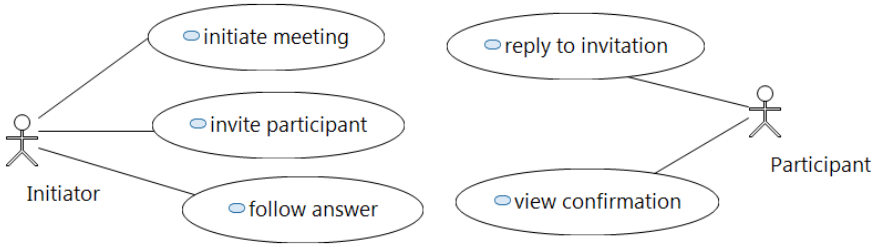

Figure 1. Le diagramme des cas d'utilisation du système d'organisation de réunions

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. Afin d'assurer la confidentialité et l'intégrité des réunions, le système d'organisation des réunions applique une politique de contrôle d'accès qui définit une seule cible de sécurité: la classe Meeting. Elle se limite aux données les plus sensibles qui sont constituées par les données des réunions telles que la date, l'heure et le lieu.

La figure 2 présente la spécification SecureUML de la politique de sécurité, elle comprend le diagramme de classes et ajoute 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.

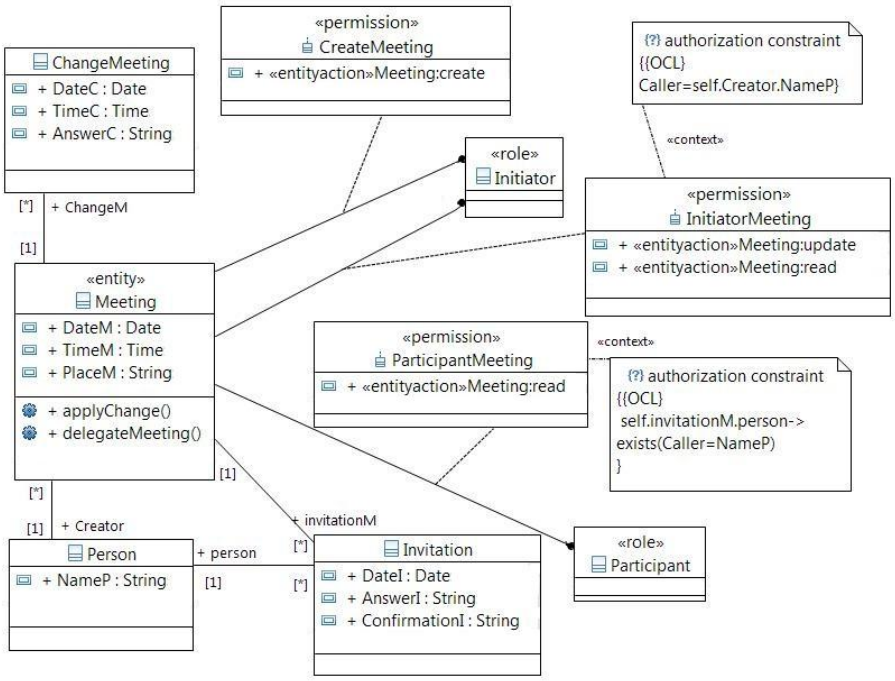

Figure 2. Le diagramme de classes SecureUML du système d'organisation de réunions

3.2. 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 :

– Modification de la classe Meeting (EntityAction 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 les opérations applyChange et delegateMeeting qui sont des opérations de modification.

– Creation de la classe Meeting (EntityAction Create) : permet l'appel au constructeur d'instances de la classe Meeting.

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

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 nécessaire pour la réalisation des opérations critiques autorisées.

4. Contrôle d'accès aux activités

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.

4.1. Activité abstraite

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. 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. Une activité abstraite permet de représenter un cas d'utilisation par un ensemble de tâches en coordination, représentées par des nœuds d'action, qui seront exécutées par les acteurs du cas d'utilisation. Une activité abstraite peut montrer plusieurs scénarios d'exécution. La figure 3 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.

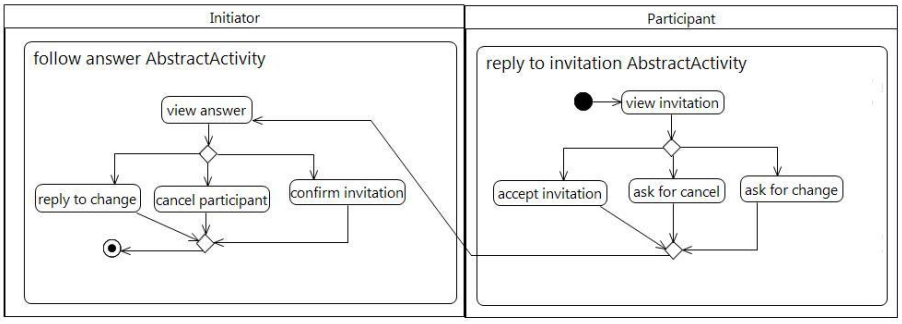

Figure 3. Activités abstraites des cas d'utilisation "reply to invitation" et "follow answer"

4.2. Activité concrète

Les activités concrètes sont construites à partir des activités abstraites. Les activités abstraites décrivent le système comme une boite noire sans détailler les objets qui le composent. Une fois qu'on dispose d'un diagramme de classes, ces diagrammes abstraits peuvent être raffinés en précisant les objets et les opérations qui les réalisent. Une activité concrète décrit 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, ... Dans la figure 4, chacune des tâches *view answer* et *reply to change* de l'activité abstraite *follow answer* (la partie gauche de la figure 3) est décomposée en une coordination d'actions concrètes. 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, C des entités (Meeting, Invitation et ChangeMeeting), dans la figure 4, sont indispensables pour l'exécution de l'activité *Follow answer*. Une activité concrète regroupe une famille de scénarios qui seront exécutés par un utilisateur représentant l'acteur du cas d'utilisation décrit par l'activité.

4.3. Pré-condition de contrôle d'accès

La séparation des préoccupations encourage à spécifier indépendamment les aspects fonctionnels et sécuritaires d'un système d'information. Cependant, leur interaction doit être prise en considération au niveau des diagrammes d'activités concrètes. Cette section explique comment exprimer une politique de sécurité spécifiée par SecureUML sur les activités concrètes fonctionnelles. Nous allons montrer comment dans la réalisation d'un processus métier, les permissions du diagramme SecureUML seront prises en compte.

Dans le diagramme d'activité de la figure 4, les permissions et leurs éventuelles contraintes d'autorisation sont exprimées par des préconditions. Ces préconditions sont soit associées à l'ensemble du diagramme, soit attachées à des actions concrètes.

La précondition associée à l'ensemble du diagramme, aussi appelée précondition d'activité, est stéréotypée par «PreCondition». Dans la figure 4, celle-ci impose que l'activité soit exécutée par un utilisateur associé au rôle "Initiator". Elle s'exprime en OCL par *Initiator.assignedUser -> includes (User)*. Cette précondition porte sur toutes les actions de l'activité. 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

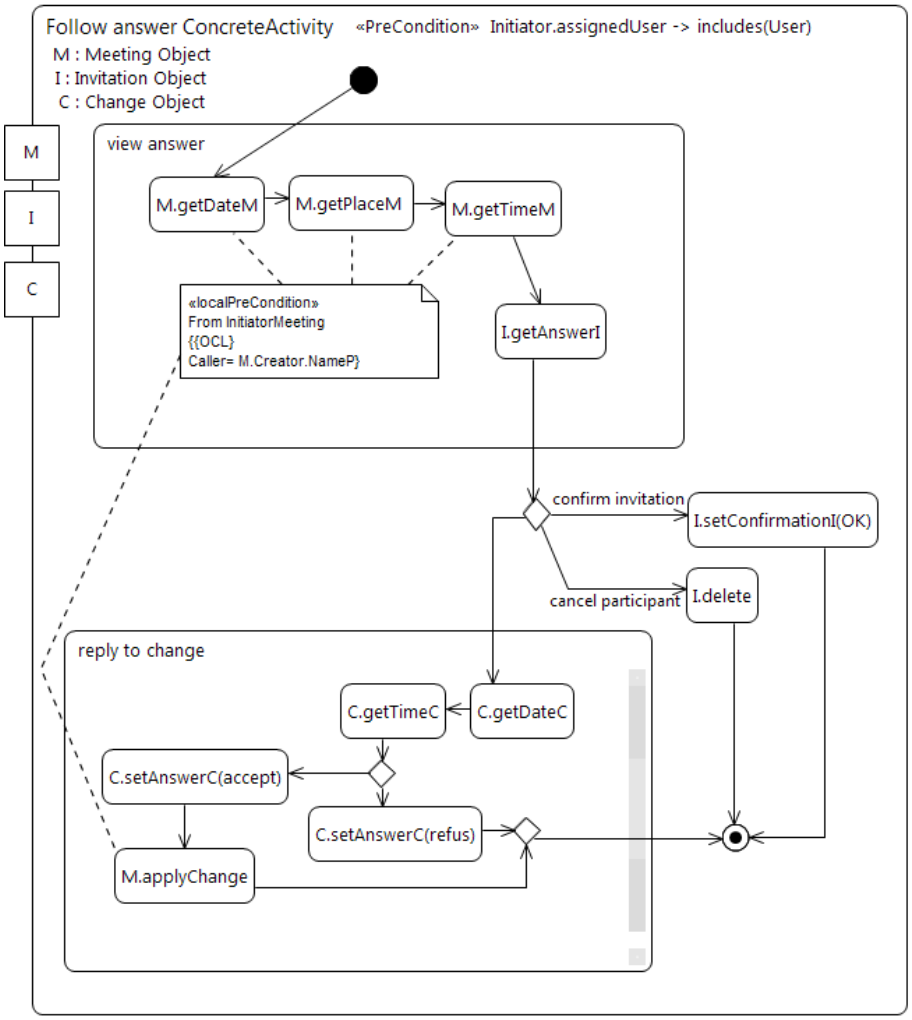

Figure 4. Contrôle d'accès à l'activité concrète " Follow answer"

(figure 1). Nous considérons donc les acteurs associés aux cas d'utilisation comme des rôles.

La précondition d'activité exprime le fait que toutes les actions doivent être exécutées par un utilisateur dans un rôle donné. Cependant, certaines actions 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 sont exprimées sous la forme de contraintes stéréotypées comme «localPreCondition» et reliées aux actions concrètes(UML2, 2011). Dans la figure 4, cette précondition locale 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 2.

Les préconditions d'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 4, on peut se contenter d'évaluer la précondition d'activité en entrée de l'activité, et la précondition locale avant l'exécution de *M.getDateM* si les conditions suivantes sont respectées:

1. L'utilisateur n'utilise pas d'autre rôle 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. Ce qui 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 vérifier les gardes plus souvent et d'encapsuler tout ou partie du diagramme dans une ou plusieurs transactions.

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

Notre travail vise à intégrer les concepts de SecureUML au niveau des diagrammes d'activité. 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.

5.1. Liens entre méta-modèles

Le diagramme de la figure 5 représente les concepts de SecureUML en lien avec les notions d'activité abstraite et d'activité concrète utiles dans le présent travail. 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. 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-

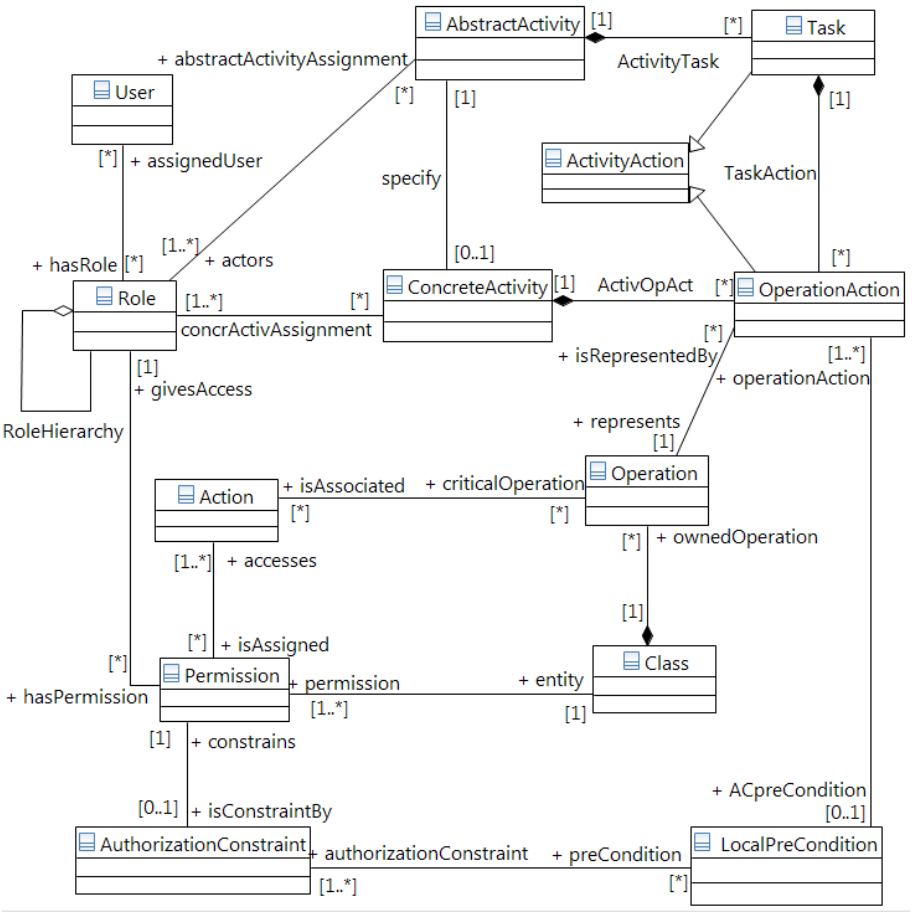

Figure 5. Métamodèle de contrôle d'accès aux activités

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, et les opérations non protégées (comme par exemple *I.setConfirmationI* et *I.delete* pour la classe Invitation). Aucune vérification n'est effectuée pour les opérations non protégées. En revanche, pour les opérations critiques, nous projetons les contraintes d'autorisation qui leur sont associées, sous forme de préconditions des actions OperationAction. De ce fait, l'appel d'une opéra-

tion fonctionnelle dans une activité concrète devient préconditionné par la satisfaction de contraintes d'autorisation.

5.2. Règles de cohérence

Dans cette sous-section nous spécifions au moyen d'expressions OCL les règles de cohérence que les instances de notre méta-modèle doivent respecter. Pour ce faire, nous définissons d'abord deux requêtes OCL, `isCritical()` et `criticalOps()` qui permettent respectivement d'indiquer si une opération donnée est critique, et de récupérer pour une activité concrète l'ensemble de ses opérations critiques. Par exemple, les actions associées à des opérations critiques de l'activité *Follow answer* de la figure 4 sont : *M.getDateM, M.getPlaceM, M.getTimeM et M.applyChange*. Les autres actions sont en relation avec des opérations non protégées.

Context Operation::isCritical():Boolean
Body: self.isAssociated → notEmpty()

Context ConcreteActivity::criticalOps():Set(Operation)
Body: self.ActivOpAct.represents → select(o:Operation | o.isCritical())

5.2.1. Conformité des rôles

L'invariant *RoleConformance* 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.

Context ConcreteActivity inv RoleConformance :
self.role → (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 la suite de nos travaux, nous étudierons d'une part la possibilité de relacher 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.

Dans notre démarche, les rôles associés à une activité concrète correspondent aux acteurs définis dans l'activité abstraite qu'elle spécifie. Nous établissons ainsi l'invariant suivant pour garantir un usage de rôles adéquat durant le processus de specification allant des activités abstraites aux activités concrètes.

Context ConcreteActivity inv RoleSpecification :
self.concrActivAssignment = self.specify.actors

5.2.2. Traçabilité des contraintes d'autorisation

Les contraintes d'autorisation définies au niveau des permissions sont projetées au niveau des activités concrètes sous forme de préconditions d'actions. Ces préconditions, ainsi issues des permissions *SecureUML*, permettent de contrôler l'exécution des actions d'activité concrète. Elles peuvent soit autoriser le déclenchement des actions d'opération critiques si les contraintes d'autorisation sont vérifiées ou bien bloquer l'exécution de l'activité dans le cas contraire. Le lien entre les méta-classes Precondition et AuthorizationConstraint permet d'avoir une traçabilité entre ces deux notions.

6. 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. A 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.

Enfin, nous avons proposé 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 grandes tailles 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.

La suite de ce travail sera principalement consacrée à la mise en oeuvre de cette approche et du méta-modèle associé dans des outils. La plateforme B4MSecure [2] (Ledru Y. et al, 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'activités abstrait et concret), de vérifier la conformité de ces diagrammes à notre méta-modèle, et ensuite de traduire 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.

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. *Inf. & Softw. 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 Trans. Softw. Eng. Methodol.*, 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.

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)

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

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

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.

Jurjens J. (2010). *Secure systems development with UML.* Berlin, Heidelberg, Springer-Verlag.

Kandala S., Sandhu R. (2002). Secure role-based workflow models. *The International Federation for Information Processing*, vol. 87, n° 2002, p. 45-58.

Ledru Y. et al. (2015). Validation of IS Security Policies featuring Authorisation Constraints. *International Journal of Information System Modeling and Design (IJISMD)*, vol. 6, n° 1.

Ma G., Wu K., Zhang T., Li W. (2011). A flexible policy-based access control model for workflow. *Computer Science and Automation Engineering*, vol. 2, n° 2011, p. 533 - 537.

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

Murata T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, vol. 77, n° 4, p. 541 - 580.

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

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

Strembeck M., Mendling J. (2011). Modeling process-related RBAC models with extended UML activity models. *Information and Software Technology*, vol. 53, n° 2011, p. 456-483.

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° 2007, 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)