AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur : ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite de ce travail expose à des poursuites pénales.

Contact : portail-publi@ut-capitole.fr

LIENS

Code la Propriété Intellectuelle – Articles L. 122-4 et L. 335-1 à L. 335-10
Loi n°92-597 du 1er juillet 1992, publiée au *Journal Officiel* du 2 juillet 1992
http://www.cfcopies.com/V2/leg/leg-droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

THÈSE

En vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :
Université Toulouse 1 Capitole

Discipline ou spécialité :
Informatique

Présentée et soutenue par
Aymen KAMOUN
le : 17 novembre 2014

Titre :
Adaptation d'architectures logicielles de contrôle d'accès dans les environnements collaboratifs ubiquitaires

---

École doctorale :
Mathématiques Informatique Télécommunications (MITT)

Unité de recherche :
LAAS-CNRS

Directeurs de thèse :
Saïd Tazi, Maître de conférences, UT1, LAAS-CNRS
Khalil Drira, Directeur de recherche, LAAS-CNRS

Rapporteurs :
Mourad Chabane OUSSALAH, Professeur, LINA, Université de Nantes
Ons Chirine GHEDIRA GUEGAN, Professeur, IAE Lyon, Université Lyon 3

Autres membres du jury :
Jean-Christophe Lapayre, Professeur, DISC, Université de Franche-Comté
Bernard Coulette, Professeur, IRIT, Université de Toulouse 2

À Ma chère femme, Mariem

Remerciements

Je voudrais tout d'abord exprimer mes plus profonds remerciements à mes directeurs de thèse, Saïd Tazi et Khalil Drira, pour le grand soutien qu'ils m'ont apporté. Leur investissement dans mes recherches et leurs remarques ont été toujours une source de motivation et d'apprentissage.

Je voudrais aussi exprimer ma reconnaissance à Ons Chirine GHEDIRA GUEGAN et à Mourad Chabane OUSSALAH, qui ont accepté de rapporter mon manuscrit, ce qu'ils ont fait avec soin et intérêt. Je remercie également Jean-Christophe Lapayre et Bernard Coulette, qui ont été examinateurs de mon travail.

J'adresse ma plus grande gratitude à tous ceux qui ont partagé avec moi leur temps dans des sessions de travail. Spécialement Mahdi Ben Alaya pour ses efforts avec le service de déploiement, Ismail Bouassida pour son soutien depuis mon projet de fin d'étude et aussi Codé Diop, Emna Mezghanni, Mohamed Zouari et Ghada Gharbi .

Je voudrais également remercier spécialement Mohamed Amine Hannachi, Mahdi Ben Alaya et Mohamed Siala qui ont été des collègues tout au long de ma thèse. Je me souviendrai de nos matchs de Baby foot, de Ping Pong, des discussions sur ces matchs et surtout des soirées au diwan. Je remercie spécialement Hnach d'avoir été toujours disponible pour faire le tour du LAAS pour boire de l'eau qui est devenu virtuel depuis deux ans. Je n'oublierai jamais notre séjour à l'Andorre avec Mehdi Zaier qui reste toujours inoubliable.

Mes remerciements vont aussi à tous les doctorants et post-doc de l'équipe SARA avec lesquels j'ai pu partager des moments de détente, notamment les matchs de foot.

Je voudrai exprimer ma profonde reconnaissance à tous les membres de ma famille pour leur soutien qui a été inébranlable tout au long de ces trois ans. Mes parents, Abdelmajid et Basma, ma soeur, Rahma et mes frères, Nabil et Ahmed, ont été à côté de moi jour après jour avec leur soutien et leur amour. Je remercie également mes grands parents, mes oncles, mes tantes et ma belle-famille.

Je conclurai en remerciant de tout cœur ma femme, je ne saurais trouver assez de mots, ni les mots justes, pour la remercier pour tout ce qu'elle a fait pour moi. Mariem, tu m'as toujours soutenu, tu m'as supporté, tu m'as motivé, tu m'as fait avancer jusqu'à la fin de ce chemin. Tu t'es même mariée avec moi dans le début de ma troisième année de thèse ! Tu me rends, toujours, heureux. Cette thèse, ainsi que tout ce que je fais, est pour toi.

Table des matières

Liste des figures

Liste des tableaux

1 Introduction

1.1 Contexte

De nos jours, les technologies doivent être invisibles en les intégrant dans notre vie quotidienne. Cette idée fut proposée par Marc Weiser [Wei91] sous le nom de l'*informatique ubiquitaire (Ubiquitous Computing)* au début des années 1990. Cette vision révolutionnaire a inspiré de nombreux travaux dans différents domaines qui visent à changer les manières d'interaction entre les utilisateurs humains et leurs machines. Ces travaux ont été l'origine d'une évolution des terminaux mobiles et des technologies de communication qui ouvre la voie au développement d'applications innovantes. L'objectif de ces applications est d'offrir une grande variété de services à travers lesquels un grand nombre d'information et de ressources sont disponibles pour les utilisateurs. En informatique ubiquitaire, les machines se caractérisent par une haute mobilité et peuvent souvent fonctionner dans des environnements et réseaux différents. Ces machines sont hétérogènes en termes de moyens de communication et de capacités de traitement. Elles ne communiquent pas seulement avec les utilisateurs, mais aussi entre elles afin de répondre aux attentes des utilisateurs. Les interactions entre les machines visent à réduire les interventions des utilisateurs tout en satisfaisant leurs besoins à travers des services qui utilisent des informations spécifiques qu'on appelle *contexte*.

Un environnement ubiquitaire désigne souvent l'ensemble des services et machines en interactions. Ce type d'environnement vise à simplifier la vie quotidienne des utilisateurs en leur offrant des moyens qui leurs facilitent d'achever leurs tâches personnelles et professionnelles. Les services offerts par les environnements ubiquitaires peuvent être exploités pour assister la collaboration entre des utilisateurs. Dans ce contexte, les environnements de travail collaboratif permettent aux utilisateurs de collaborer dans différents groupes de travail en utilisant une ou plusieurs machines. Les utilisateurs peuvent ainsi se connecter de plusieurs emplacements afin de travailler sur des ressources partagées [DGLA99]. En outre, ils peuvent avoir plusieurs rôles durant les processus de collaboration. Par exemple, un utilisateur peut jouer un premier rôle de coordinateur dans un groupe et un rôle de concepteur dans un autre groupe. En fonction de leurs rôles, ces utilisateurs peuvent être impliqués dans plusieurs activités collaboratives. On distingue deux types de collaboration : (1) la collaboration formelle où le processus de collaboration est prédéfini en avance, et (2) la collaboration spontanée où les utilisateurs établissent des collaborations ad-hoc en utilisant différents outils tels que les outils de messagerie instantanée et les outils de conférence multimédia.

La gestion des ressources partagées est un des besoins fondamentaux dans les environnements collaboratifs. En fait, certaines de ces ressources peuvent ne pas être publiques et sont accessibles seulement par un ensemble particulier d'utilisateurs. Le contrôle d'accès aux ressources devient ainsi indispensable. Il permet de protéger les ressources et d'interdire les accès malveillants à travers des politiques de contrôle d'accès. Depuis les années 1970, de nombreux modèles ont été proposés pour l'expression de ces politiques dans les environnements statiques mono utilisateur. Dès l'émergence de l'informatique ubiquitaire dans les années 1990, de nouvelles exigences des modèles et des mécanismes de contrôle d'accès ont été étudiées afin de s'adapter aux caractéristiques des environnements ubiquitaires.

Le contrôle d'accès est plus complexe dans les environnements de travail collaboratif ubiquitaires pour deux raisons principales. D'une part, la variété des rôles et des tâches des utilisateurs peut causer des changements imprévisibles au niveau de la structure de la collaboration. En effet, les utilisateurs peuvent joindre plusieurs groupes ou changer de groupe au cours de la collaboration. Nous citons l'exemple des réunions par audio et vidéo conférence où des utilisateurs appartenant à différentes organisations définissent et partagent des ressources. Si, par exemple, au cours du processus de collaboration, les développeurs d'une organisation *o1* ont besoin de collaborer et de partager des ressources avec les développeurs d'une organisation *o2*, alors quelles sont les actions à faire pour assurer un accès transparent et sécurisé aux ressources ? Ces utilisateurs peuvent ainsi joindre d'autres groupes et partager d'autres ressources. La fréquence des changements au niveau de la structure de la collaboration dépend des besoins des utilisateurs et du processus de collaboration. L'aspect dynamique des environnements ubiquitaires pose des problèmes de sécurité et de contrôle d'accès aux ressources. D'autre part, l'augmentation du nombre d'utilisateurs, de ressources et de politiques de contrôle d'accès pose des problèmes de passage à l'échelle dans les grands réseaux collaboratifs. Les entités de contrôle dans ces environnements doivent être décentralisées et réparties dans les différents nœuds terminaux qui constituent le réseau [KS07]. Mais comment cette décentralisation peut être effectuée dans un environnement dynamique où la structure de collaboration peut évoluer dans le temps et le nombre d'utilisateurs n'est pas connu en avance ? Il est intéressant d'avoir des systèmes qui interprètent la structure de la collaboration et les actions des utilisateurs afin de mettre en place un mécanisme de contrôle d'accès décentralisé tout en tenant compte de l'aspect dynamique de l'environnement.

1.2 Contributions

Dans nos travaux de recherche, nous nous intéressons au contrôle d'accès dans les environnements collaboratifs ubiquitaires. Le but est de concevoir de nouveaux mécanismes de contrôle d'accès aux ressources qui sont adaptés à ce type d'environnements et qui assurent la bonne conduite des activités collaboratives. Les environnements ubiquitaires sont des environnements dynamiques où le contexte change fréquemment. Pour cela, il est nécessaire que les systèmes collaboratifs puissent s'adapter en fonction des changements du contexte. Notre objectif dans ce travail de thèse est de pouvoir reconfigurer dynamiquement, d'une part les architectures logicielles de

contrôle d'accès et d'autre part celle de collaboration afin d'assurer l'auto-adaptation et l'auto-protection des systèmes collaboratifs. Les mécanismes d'adaptation que nous proposons prennent en compte des besoins des applications.

Nous listons ci-dessous nos principales contributions :

- Architecture décentralisée de contrôle d'accès : nous proposons une méthode de décentralisation des entités de contrôle d'accès tout en assurant un échange standard des requêtes d'accès et des réponses d'autorisation. Dans cette architecture, le contrôle peut être effectué au niveau des nœuds terminaux qui constituent le réseau. Nous montrons l'effet de la décentralisation sur les performances du mécanisme de contrôle d'accès.
- Modèle générique de contrôle d'accès : nous avons défini un modèle générique de contrôle d'accès, appelé *Generic Access Control Ontology* (GACO), qui permet principalement d'exprimer un ensemble de connaissances et d'effectuer du raisonnement. Une instance de ce modèle représente les situations de collaboration et les besoins de contrôle d'accès à tout instant durant l'activité collaborative.
- Framework de contrôle d'accès : nous avons développé un framework de contrôle d'accès qui utilise le modèle générique de contrôle d'accès et des techniques de transformation de graphes. Ce framework permet l'adaptation de l'architecture logicielle de contrôle d'accès en fonction des exigences de l'application et des utilisateurs.
- Processus et techniques de déploiement des composants : nous proposons un processus et des techniques de déploiement des composants qui assurent le déploiement et la reconfiguration physique des composants logiciels selon des plans de déploiement.

Afin d'illustrer l'utilisation du framework de contrôle d'accès et de valider son fonctionnement nous proposons un exemple d'application développée dans le cadre du projet ANR Galaxy [1]. Cette application nous sert également pour valider le processus de déploiement des composants.

1.3 Organisation du mémoire

Ce mémoire est organisé comme suit :

Le chapitre 2 présente l'état de l'art. Il introduit les concepts utilisés dans les domaines auxquels nous nous intéressons dans nos travaux : les environnements collaboratifs ubiquitaires, le contrôle d'accès et la représentation des connaissances. Ensuite, nous analysons les travaux existants qui traitent des problématiques similaires à celles que nous traitons et nous situons nos contributions par rapport à ces travaux.

Dans le chapitre 3, nous présentons l'architecture de décentralisation des composants de contrôle d'accès qui se base sur les éléments du standard *eXtensible Access Control Markup Language* (XACML). Ce standard est très répandu et largement utilisé pour la spécification des politiques de contrôle d'accès et leur application afin de sécuriser l'accès aux ressources. Ensuite, nous introduisons les éléments principaux du langage *Web Ontology Lanaguage* (OWL) que nous avons utilisé pour la représentation des connaissances

1. La description du projet est disponible sur http://www.irit.fr/GALAXY/

et le raisonnement. Nous présentons également notre modèle de contrôle d'accès, les règles qui y sont associées et comment nous pouvons l'étendre pour exprimer les situations de collaboration dans des domaines spécifiques. La dernière partie de ce chapitre est consacrée à la présentation de notre framework de contrôle d'accès. Nous présentons l'approche d'adaptation et nous détaillons les différents modèles produits lors du processus d'adaptation.

Le Chapitre 4 présente notre apport pour le déploiement des composants. En premier lieu, nous présentons le cadre dans lequel nous avons besoin du déploiement. Notamment, nous nous en servirons pour le déploiement des composants de contrôle d'accès et des composants de communication. En second lieu, nous détaillons le processus de déploiement, ainsi que les techniques et les algorithmes qui assurent un déploiement cohérent et qui correspond aux exigences de l'application.

Dans le Chapitre 5, nous présentons un exemple d'application, développée dans le cadre du projet ANR Galaxy, qui nous sert pour évaluer le framework de contrôle d'accès et le processus de déploiement des composants. Ensuite, nous présentons une évaluation de performance du processus d'adaptation en mesurant le temps d'exécution nécessaire pour que l'application s'adapte aux changements de contexte. Nous évaluons également les performances des composants de contrôle d'accès en mesurant le temps de génération des décisions d'autorisation d'accès et nous mettons l'accent sur l'effet de décentralisation du contrôle d'accès sur les performances du système.

État de l'art

2

Sommaire

Dans ce chapitre, nous présentons les domaines de recherche que nous avons étudiés et les notions de base dont nous nous servirons tout au long de ce manuscrit. Ce chapitre nous permettra de bien nous positionner par rapport à divers aspects faisant intervenir différents domaines. Dans la première partie de ce chapitre, nous présentons les concepts liés à la collaboration et à l'informatique ubiquitaire. Nous nous focalisons, d'une part, sur les caractéristiques d'interaction entre les utilisateurs et les machines, et d'autre part sur les problèmes critiques adressés dans l'informatique ubiquitaire tels que l'aspect dynamique et le passage à l'échelle. Dans la deuxième

partie de ce chapitre, nous étudions les différents modèles de contrôle d'accès ainsi que les exigences que doivent satisfaire ces modèles et les mécanismes de contrôle d'accès. Ensuite, nous étudions les techniques de représentation sémantique des connaissances que nous utiliserons dans notre travail.

Après l'étude de ces domaines, nous analysons les travaux existants qui traitent les problématiques de contrôle d'accès dans les environnements collaboratifs ubiquitaires et nous positionnons nos contributions par rapport à ces travaux.

2.1 Environnements collaboratifs ubiquitaires

2.1.1 Informatique ubiquitaire

Le concept d'informatique ubiquitaire a été introduit par Marc Weiser en 1991 [Wei91]. Il imaginait un monde peuplé d'objets informatiques et numériques qui interagissent d'une manière autonome afin d'offrir des services. L'objectif de l'informatique ubiquitaire est de permettre aux utilisateurs d'accéder à ces services offerts par les objets communicants d'une manière transparente. Les utilisateurs doivent être capables d'accéder aux services et d'utiliser les technologies de l'information et de communication n'importe où, à tout moment et à travers divers dispositifs. L'informatique ubiquitaire peut être considérée comme une tendance vers l'informatisation et la miniaturisation des dispositifs électroniques favorisant l'accès aux ressources et fournissant des services de communication et d'interaction.

Selon Weiser, les ordinateurs doivent être invisibles en les intégrant dans notre environnement. Il déclarait : "*Les technologies les plus profondes sont celles qui disparaissent. Elles se fondent dans la vie de tous les jours jusqu'à ce qu'on ne peut plus les percevoir*". Cette vision permet aux utilisateurs de se concentrer surtout sur leurs tâches et leurs objectifs plutôt que sur la configuration de leurs machines.

En 2001, Satyanarayanan [Sato1] a fait une étude sur les thématiques de recherche qui contribuent à l'émergence de l'informatique ubiquitaire. Il explique que l'évolution des systèmes ubiquitaires se base principalement sur les systèmes distribués et les systèmes mobiles. Dans son article *Pervasive Computing : Vision and Challenges* [Sato1], il pose des questions liées à la sécurité telles que : "Comment on peut exprimer des contraintes de sécurité dans un environnement ubiquitaire ?" et "Quels sont les techniques d'authentification les plus adaptées à un environnement ubiquitaire ?". Dans le même article, Satyanarayanan identifie des axes de recherche liés à l'informatique ubiquitaire comme les espaces intelligents, l'invisibilité et le passage à l'échelle localisé.

Le passage à l'échelle est un problème critique dans les environnements ubiquitaires. En effet, le nombre d'interaction est important dans ce type d'environnements, ce qui fait que l'utilisation des ressources telles que la bande passante et l'énergie peut devenir critique. La présence de plusieurs utilisateurs peut compliquer encore la situation. Dans le cas des systèmes classiques, les études ne tiennent pas compte de la localisation physique des entités ; par exemple un serveur ou un serveur de fichier doit servir le plus grand nombre possible de clients sans tenir compte de leurs positions géographiques. Dans le cas des systèmes ubiquitaires, la situation est différente. En effet, les interactions se font entre des entités proches et donc le nombre d'interaction diminue lorsqu'un utilisateur s'éloigne d'un environnement ubiquitaire. Cette caractéristique peut être exploitée pour améliorer le passage à l'échelle dans les environnements ubiquitaires. Cependant, dans des environnements complexes où plusieurs utilisateurs interagissent et partagent un grand nombre de ressources, le passage à l'échelle reste toujours un défit à relever. Les systèmes doivent tenir compte, dès leur conception, des possibilités d'évolution à une échelle plus grande du nombre d'utilisateurs, du nombres des machines, etc.

Ce nouveau paradigme de l'informatique ubiquitaire a atteint une place prépondérante dans les dernières années ; il représente un nouveau type d'interaction entre les utilisateurs et leurs dispositifs. De nos jours, les technologies de communication sans fil telles que les réseaux locaux sans fil et Bluetooth assurent une meilleure interaction entre les dispositifs sans même avoir besoin d'une infrastructure centrale (les réseaux ad-hoc). En plus, les utilisateurs disposent de différents types d'unités mobiles telles que les ordinateurs portables, les assistants numériques personnels (PDA) ou les téléphones intelligents (Smartphone). Le changement de type d'interaction ne se limite pas uniquement aux changements d'interface homme-machine ; et l'utilisateur peut interagir avec plusieurs dispositifs simultanément, ce qui oblige les nouveaux systèmes informatiques à répondre à ces nouveaux besoins.

2.1.2 Notion de collaboration

La notion de collaboration a été initialement étudiée dans le domaine du travail collaboratif assisté par ordinateur (*Computer Supported Cooperative Work* - CSCW) qui a été introduit dans les années 1980 et raffiné dans les années 1990 [CS99]. Le terme collaboration assistée par ordinateur (*Computer Supported Collaboration* - CSC) est parfois utilisé pour remplacer le terme CSCW. Plusieurs définitions de ce terme ont été proposées. Grudin [Gru94] a proposé une définition comme suit :

> *« C'est étudier comment les gens peuvent utiliser la technologie, en plus du matériel et des logiciels, pour travailler ensemble dans des environnements partagés »*

Les termes communication, coordination et collaboration sont très utilisés dans CSCW. La communication représente l'échange d'information entre des utilisateurs. Cet échange peut être assuré à travers l'utilisation de la messagerie ou le courrier électronique. La coordination est l'action de coordonner les activités des membres d'un groupe donné. La collaboration est l'action de travailler ensemble. Deux ou plusieurs utilisateurs ayant des compétences complémentaires peuvent collaborer ensemble et partage la compréhension d'un processus, d'un produit ou d'un évènement [Sch90].

Un des défis des systèmes collaboratifs est de bénéficier des technologies et des opportunités offertes par les environnements ubiquitaires [HL09]. En effet, les diverses informations relatives aux utilisateurs et aux ressources qu'ils partagent peuvent être exploitées afin de proposer des services de collaboration et de sécurité adaptés aux besoins du système.

Structuration et organisation de la collaboration

Dans les environnements de travail collaboratif, les utilisateurs qui partagent des intérêts communs sont généralement organisés dans des groupes. Un groupe peut être défini en se basant sur plusieurs critères tels que la position géographique des utilisateurs, les ressources qu'ils partagent, etc. La notion de session est essentielle pour l'organisation et la structuration de la collaboration. Une session regroupe un ensemble d'utilisateurs qui se connectent de divers emplacements afin de travailler ensemble sur des données partagées [DGLA99].

Une session peut être synchrone ou asynchrone [Vilo6]. Dans une session synchrone, les utilisateurs sont présents en même temps et interagissent en temps réel. Les réunions par audio et vidéo conférence sont des exemples d'une session synchrone. Dans une session asynchrone, les membres du groupe n'interagissent pas en temps réel. Ils communiquent via des médiums asynchrones tels que le courrier électronique. Une session peut être implicite ou explicite. Dans le cas des sessions explicites, la définition des sessions se fait lors de la phase de conception du système. Tandis que les sessions implicites s'établissent en fonction des besoins des utilisateurs ou des exigences de l'application. Le système peut établir une session quand il détecte que certains utilisateurs ont intérêt à collaborer ensemble. Dans notre travail, nous considérons que chaque session regroupe un ensemble d'utilisateurs qui ont des rôles spécifiques définis par le concepteur de l'application. Nous nous sommes basés sur les rôles afin de faciliter l'administration et la gestion des sessions.

Plusieurs critères ont été définis pour la gestion des sessions [Vilo6]. Les deux critères qui nous semblent importants dans notre travail sont :

- La gestion des rôles des utilisateurs : ce critère définit les rôles manipulés par le système.
- La politique de droits d'accès : ce critère précise comment sont gérés et attribués les droits d'accès dans le système.

Les fonctionnalités de sécurité telles que l'authentification et les contrôle d'accès sont essentielles pendant la durée de vie d'une session afin de sécuriser l'accès aux ressources. Dans le cas où les sessions sont statiques (i.e., n'évoluent pas au cours du temps), les fonctionnalités de sécurité sont simples à mettre en place. Cependant, dans le cas des sessions dynamiques qui évoluent au cours du temps, la mise en place des fonctionnalités dynamiques de sécurité devient complexe. Dans notre travail, nous nous intéressons à la mise en place des mécanismes de contrôle d'accès afin de sécuriser l'accès aux ressources dans les sessions de collaboration qui évoluent dans le temps.

Collaboration dans les environnements d'ingénierie de logiciels

Ballesteros et al. [PDBo6] ont introduit le concept d'environnements de travail collaboratif (*Collaborative Working Environments* - CWE) dont l'objectif est de développer des technologies qui assurent l'interaction synchrone et asynchrone entre des individus qui collaborent ensemble pour atteindre des objectifs communs. Ces technologies permettent d'améliorer les processus métier dans les industries et dans le domaine d'ingénierie de logiciels. Nous considérons que les environnements de travail collaboratif sont des environnements particuliers de CSCW dans lesquels les entités qui collaborent appartiennent principalement à des organisations.

Dans le domaine de l'ingénierie de logiciels, une grande quantité d'informations doit être échangée entre plusieurs groupes de travail. Ces informations peuvent être simples telles qu'une spécification d'un logiciel ou encore plus complexes tels que des modèles sous la forme de diagrammes ou de codes. Afin d'automatiser l'évolution des processus de production des logiciels, un méta-modèle de conception de processus SPEM [1] (*Software Process*

1. http://www.omg.org/spec/SPEM/2.0/

Engineering Metamodel) a été introduit. Ce méta-modèle décrit les concepts qui permettent de décrire le processus de production de logiciels. Dans notre étude, nous nous intéressons au contrôle d'accès et à la collaboration durant le processus de production de logiciels. Dans la production de logiciels, la collaboration entre les membres de groupes est compliquée et elle n'est pas simple à gérer à cause de l'énorme quantité d'informations partagée et de la communication entre les différents types d'utilisateurs [MGvdHW10]. Généralement, les membres d'un groupe de production de logiciels collaborent quand ils détectent de nouveaux besoins ou/et quand ils ont des objectifs communs à un moment donné du processus de production. Par conséquent, la collaboration est fortement dépendante des besoins des groupes et de leurs situations. Une étude montre que plus que 69% des travaux collaboratifs dans le développement des logiciels sont ad-hoc et qu'ils ne sont pas planifiés antérieurement [RR00]. La collaboration ad-hoc est fortement sensible aux besoins des groupes et aux changements des situations des participants.

2.1.3 Notion de contexte dans l'informatique ubiquitaire

La notion de contexte attire l'attention de plusieurs chercheurs qui traitent des problématiques liées à l'informatique ubiquitaire.

Définitions

Plusieurs définitions du contexte ont été proposées depuis les années 1990 [Cha07]. Schilit et Theimer [ST94] sont les premiers qui ont défini le contexte. Selon eux, le contexte est représenté par la localisation de l'utilisateur et les identités et les états des personnes et des objets qui l'entourent. Brown et al. [BBC97] rajoutent à cette définition l'orientation de l'utilisateur, la saison et la température. Ryan et al. [RPM98] définissent le contexte en tant que la localisation, l'environnement, l'identité de l'utilisateur et ajoutent la notion de temps. Dey [Dey98] ajoute au contexte l'état émotionnel de l'utilisateur. Pascoe [Pas98] définit le contexte comme un ensemble d'états physiques et conceptuels d'une entité donnée. Dey et al. [ADB+99] définissent le contexte comme suit : « *Le contexte est toute information qui peut être utilisée pour caractériser la situation d'une entité. Une entité est une personne, un endroit ou un objet qui est considéré pertinent pour l'interaction entre l'utilisateur et l'application, y compris l'utilisateur et l'application* ». Winograd [Win01] affirme que cette définition couvre toutes les définitions précédentes du contexte. Cependant il considère qu'elle est trop générale. Il précise que le contexte est un ensemble structuré et partagé d'informations qui peut évoluer dans le temps. Selon lui, la considération d'une information comme contexte dépend de la manière dont elle est utilisée et non de ses propriétés inhérentes. Kulkarni et al. [KAT12] précisent que le contexte est toute situation, de l'environnement physique ou virtuel, qui peut être utilisée par une application pour effectuer une adaptation dynamique.

Nous considérons que les deux dernières définitions sont les plus utiles pour l'exploitation du contexte dans les environnements de travails collaboratif ubiquitaires. Pour cela, nous définissons le contexte comme suit :

« *Le contexte est toute information pouvant être utilisée pour caractériser la*

structure de la collaboration. Les informations de contexte évoluent dans le temps et peuvent être utilisées pour effectuer une adaptation dynamique »

Dans le cas où les sessions de collaboration sont utilisées pour la structuration de la collaboration, le contexte est toute information relative à la structure des sessions.

Sensibilité au contexte

La notion de sensibilité au contexte (*Context-Awareness*) représente la capacité du système à s'adapter aux changements du contexte afin de mieux répondre aux attentes des utilisateurs. L'objectif principal des applications sensibles au contexte est de réduire le plus possible les interactions entre les utilisateurs et les machines. Plusieurs définitions d'un système sensible au contexte ont été proposées [ST94] [RPM98] [BBC97]. Selon Dey et al. [ADB+99], un système est sensible au contexte s'il utilise celui-ci pour fournir à l'utilisateur des informations et des services pertinents. Cette pertinence dépend des tâches et des activités demandées à l'utilisateur.

Les environnements de travail collaboratif ont tendance à utiliser les informations contextuelles afin d'offrir des services et améliorer les fonctionnalités des applications [KAT12]. La conception d'applications de travail collaboratif sensibles au contexte est compliquée pour plusieurs raisons. Ces applications doivent supporter différents types d'interactions et de politiques de droits d'accès. En outre les utilisateurs de ce type d'environnements changent d'un processus de collaboration à un autre en fonction des besoins. Les processus de collaboration peuvent être représentés par les sessions de collaboration. Plusieurs systèmes ont proposé des mécanismes de sensibilité au contexte pour les applications CSCW [DCME01] [HI04]. Ces systèmes sont destinés à être utilisés dans des environnements bien particuliers et ne supportent pas des modifications pour répondre aux besoins des nouvelles technologies.

2.1.4 Adaptation

Un système est dit adaptatif s'il est capable de s'adapter aux changements. Ces changements peuvent avoir lieu dans le système lui-même ou dans une partie de l'environnement qui peut influer sur le comportement du système [ST09]. Ces systèmes sont souvent appelés auto-adaptatifs. Selon Laddaga [Lad00], la première raison qui justifie le besoin des systèmes adaptatifs est l'augmentation du coût de la gestion des systèmes logiciels. Plusieurs définitions de l'adaptation ont été proposées [OGT+99]. Selon McKinley et al. [MSKC04], l'adaptation est essentielle dans l'informatique ubiquitaire et ils signalent que les dispositifs mobiles doivent s'adapter aux contraintes des réseaux ad-hoc sans fil. Un système adaptatif doit s'adapter aux changements lors de son exécution [ST09].

Les systèmes adaptatifs sont fortement liés aux systèmes autonomiques [KC03] et il est difficile de distinguer entre les deux terminologies. En comparant ces deux systèmes, on remarque qu'il y a des points communs et quelques différences [ST09]. Les systèmes adaptatifs sont plus limités et font partie des systèmes autonomes. En effet, si on considère une architecture multi-niveaux composée des niveaux application, middleware, systèmes

d'exploitation, réseau et matériel [MSKC04], un système adaptatif peut couvrir les niveaux application et middleware, tandis que les systèmes autonomes peuvent couvrir aussi les niveaux les plus bas.

Une taxonomie des approches d'adaptation a été proposée par Salehie et Tahvildari [ST09] :

- Approche d'adaptation statique vs. dynamique : Dans les approches d'adaptation statiques, le processus d'adaptation est codé en dur et sa modification nécessite une recompilation du système ou de certains de ses composants. Tandis que dans les approches d'adaptation dynamiques, les politiques d'adaptation sont définies et gérées à l'extérieur du système, et par conséquent, elles peuvent être configurées au cours de l'exécution du système.
- Approche d'adaptation externe vs. interne : Les approches d'adaptation peuvent ainsi être classées dans deux catégories en fonction de la séparation entre le mécanisme d'adaptation et la logique de l'application. Dans les approches d'adaptation externes, le système adaptatif se compose d'un moteur d'adaptation qui implémente le processus d'adaptation du système adaptable. Tandis que dans les approches internes, le processus d'adaptation est intégré dans le code de l'application.

Des types d'adaptation ont été aussi définis tels que l'adaptation spécifique et générique, l'adaptation basée sur les modèles et l'adaptation sans modèles, l'adaptation réactive et proactive, etc.

Garlan et al. [GCH+04] introduit la notion d'adaptation architecturale qui consiste à changer la structure du système lors de son exécution afin de s'adapter aux changements. Selon Salehie et Tahvildari [ST09], l'adaptation d'une architecture peut nécessiter le redéploiement de certains composants du système ainsi que la façon dont ils sont connectés. Dans notre travail, nous nous focalisons sur l'adaptation architecturale et dynamique afin de répondre aux exigences de contrôle d'accès et de collaboration dans les environnements ubiquitaires.

Dans les systèmes de contrôle d'accès, les approches d'adaptation n'ont pas été bien étudiées. La seule approche d'adaptation qui a été utilisée est l'adaptation des politiques de contrôle d'accès. Samuel et al. [SGB08] proposent des mécanismes pour l'adaptation des politiques pendant les situations critiques dans des secteurs publiques tels que l'hospitalisation, la police, la protection civile, etc. Les politiques peuvent ainsi être adaptées en fonction des changements des informations contextuelles [TMKL06] ou en fonction des situations de conflit et d'incohérence [BB03].

2.1.5 Informatique autonomique

Le terme informatique autonomique (*Autonomic Computing*) a été introduit par Paul Horn en 2001 [Hor01]. Selon lui, l'émergence des nouvelles technologies rendent les systèmes informatiques de plus en plus complexes. L'administration de ces systèmes devient une problématique à cause du nombre d'outils et d'objets communicants. De plus, contrôler ces systèmes manuellement prend beaucoup de temps et peut être fastidieux pour les utilisateurs en fonction du niveau de complexité du système. Selon Horn, *l'automatisation est essentielle pour l'évolution des systèmes*. Le but de l'informa-

tique autonomique est d'offrir des moyens d'automatisation et d'autogestion (*self-management*) aux systèmes informatiques.

En 2003, Kephart et Chess [KC03] exposent leur vision de l'informatique autonomique dans leur article "*The vision of Autonomic Computing*". Ils définissent les quatre propriétés qui doivent être implémentées dans les systèmes informatiques pour qu'ils puissent être autogérés : "*self-configuration*", "*self-optimization*", "*self-healing*" et "*self-protection*".

1. Auto-configuration (*self-configuration*) : cette propriété représente la capacité du système à se reconfigurer dynamiquement en se basant sur des politiques de haut niveau qui spécifient le comportement du système suite aux changements. Cette propriété est fortement liée à la gestion dynamique de déploiement qui consiste à installer, mettre à jour, arrêter, et intégrer des entités logicielles afin de répondre à des besoins bien particuliers.
2. Auto-optimisation (*self-optimization*) : cette propriété représente la capacité du système à utiliser au mieux les ressources afin d'assurer un niveau de performance qui permet de satisfaire les besoins des utilisateurs. Le système doit être capable d'optimiser différents paramètres tels que le temps de réponse, l'utilisation des canaux de communication, la charge, etc. Par exemple, dans le cas d'un serveur central, l'augmentation du nombre de requêtes clients peut augmenter la charge et le temps de réponse.
3. Auto-réparation (*self-healing*) : Cette propriété est aussi appelée autodiagnostic [RL05]. Elle représente la capacité du système à réagir suite aux dysfonctionnements et à anticiper et préparer des actions pour éviter ces défaillances. Au cours des réparations, le système doit rester dans un état stable.
4. Auto-protection (*self-protection*) : cette propriété représente la capacité du système à protéger les données. Deux types de protection doivent être assurés : (1) réagir suite à des attaques malicieuses où accès interdits à une ressource et (2) anticiper les menaces en prenant les mesures de sécurité adéquates afin d'éviter les problèmes ou atténuer leurs effets.

2.2 Contrôle d'accès

Le domaine de sécurité est très vaste, et il y a plusieurs sujets qui peuvent être étudiés afin d'assurer la sécurité des systèmes. Notre travail de thèse traite le contrôle d'accès qui est essentiel dans le domaine de sécurité informatique. Le contrôle d'accès a été considéré comme un sujet de recherche et de développement pendant les trente dernières années et il est défini comme suit :

> « *Le contrôle d'accès est l'interdiction d'une utilisation non autorisée d'une ressource* » (ISO 7498-2 1989)

Le contrôle d'accès suit le mécanisme d'authentification. Il permet de contrôler l'interaction entre les utilisateurs déjà authentifiés avec les ressources. La vérification de l'identité de l'utilisateur qui utilise une ressource

est principalement la tâche des services d'autorisation [SS96]. Dans ce travail, nous ne traitons pas les mécanismes d'authentification et nous supposons qu'un utilisateur s'est déjà authentifié lors de son accès à une ressource.

Dans les systèmes collaboratifs multiutilisateurs, certaines ressources sont publiques ou accessibles par tous les utilisateurs ; d'autres sont privées aux utilisateurs qui les possèdent ; et d'autres sont accessibles par certains utilisateurs. Ces besoins sont généralement exprimés avec des politiques de contrôle d'accès. Chaque système informatique implémente des mécanismes de contrôle d'accès pour sécuriser l'accès à ses ressources. Par exemple, dans un système d'exploitation, les ressources peuvent être représentées par les fichiers, les répertoires et les machines. Dans un système de gestion de base de données, les ressources sont les tables et les vues. Le contrôle d'accès permet essentiellement de vérifier si un utilisateur a le droit d'accéder à une ressource ; mais d'autres contraintes peuvent être définies telles que comment, où et quand l'accès à la ressource est autorisé. Les politiques de contrôle d'accès définissent des règles qui spécifient qui a accès à une ressource et sous quelles conditions.

Les deux termes "modèle de contrôle d'accès" et "mécanisme/système de contrôle d'accès" sont très utilisés dans le domaine le contrôle d'accès. Ils sont définis dans [Cra05] comme suit : *Un mécanisme de contrôle d'accès est un composant d'un système informatique qui permet de contrôler l'accès aux ressources pour les utilisateurs authentifiés qui possèdent des droits d'accès fournis par le système. Un modèle de contrôle d'accès définit un ensemble de fonctions et de relations qui représentent les éléments d'un mécanisme de contrôle d'accès*. Il permet de spécifier la méthode de gestion d'accès aux ressources et comment les décisions d'autorisation sont prises.

Trusted Computer System Evaluation Criteria[2], connu aussi sous le nom de livre orange (*Orange Book*) définit deux types de contrôle d'accès : Le contrôle d'accès discrétionnaire (*Discretionary Access Control* ou DAC) et le contrôle d'accès obligatoire (*Mandatory Access Control* ou MAC). Le DAC contrôle l'accès à une ressource en se basant sur l'identité de l'utilisateur. Dans le DAC, chaque ressource doit avoir un propriétaire qui est capable de donner des droits d'accès aux autres utilisateurs pour cette ressource. Le DAC est largement utilisé dans les systèmes d'exploitation tels que UNIX et Windows. Le MAC limite le niveau de contrôle que possèdent les utilisateurs sur les ressources dont ils sont propriétaires. Il classe les ressources du système en catégories et les utilisateurs ou les processus doivent avoir accès à la catégorie appropriée afin de pouvoir interagir avec ces ressources. Contrairement au DAC, les politiques d'autorisation sont gérées par un administrateur et les autres utilisateurs du système ne peuvent pas les modifier.

2.2.1 Exigences des systèmes de contrôle d'accès

Pour répondre aux besoins des environnements dynamiques dont la structure de collaboration est complexe, les systèmes de contrôle d'accès doivent satisfaire un certain nombre d'exigences. Un ensemble d'exigences fondamentales pour les services d'autorisation ont été définies comme suit [KS07] [TAPH05] [BDEKK08] [Edw96] [JP96] [FB97] :

2. Un ensemble de critères énoncés par le département de la défense des États-Unis, disponible sur http://csrc.nist.gov/publications/history/dod85.pdf

1. **Décentralisation** Afin de réduire la complexité d'administration des politiques de sécurité, la décentralisation de la gestion de ces politiques a été adoptée dans plusieurs mécanismes de contrôle d'accès (sous-section 2.4.3). Dans les environnements où la structure de collaboration est complexe et change en fonction des besoins de l'application, il est désirable que l'administration et l'application des politiques de sécurité soit prise en charge par plusieurs entités dans le système. Cela permet la diminution de charge pour chaque entité qui peut être impliquée dans plusieurs activités.
2. **Passage à l'échelle** Les mécanismes de contrôle d'accès doivent être capables de gérer un grand nombre de politiques de sécurité, d'utilisateurs, d'applications et de ressources. Le coût d'évaluation et d'application de ces politiques ne doit pas augmenter en fonction du nombre d'éléments dans le système. En plus, il faut avoir la possibilité de définir des règles de sécurité pour chaque processus de collaboration. Cette exigence est fondamentale dans les environnements collaboratifs vu que le nombre de ressources et d'applications partagées dans les processus de collaboration est beaucoup plus important dans les environnements collaboratifs que dans les environnements mono utilisateur traditionnels. A notre connaissance, le passage à l'échelle n'a pas été bien étudié pour les mécanismes de contrôle d'accès.
3. **Support de plusieurs front-end** Le mécanisme de gestion des politiques de contrôle d'accès doit être indépendant du front-end utilisé. Cela veut dire qu'on doit assurer un découplage entre les politiques de contrôle d'accès et la méthode utilisée (interfaces graphiques, langage de programmation, etc.) pour la définition de ces politiques. En effet, ces politiques peuvent durer longtemps dans une organisation ou dans un réseau d'utilisateurs, alors que la manière qu'on utilise pour la spécification de ces politiques peut changer en fonction de l'évolution des interfaces graphiques utilisées. Cette exigence n'est pas spécifique au contrôle d'accès et elle est commune à tous les systèmes et surtout avec la grande variété des machines et des systèmes d'exploitation utilisés pour des tâches d'administration.
4. **Autonomie** Chaque processus de collaboration doit définir ses propres politiques de contrôle d'accès. En outre, l'accès à une ressource dans un processus de collaboration doit être indépendant de l'accès à cette même ressource dans un autre processus de collaboration.
5. **Dynamicité** Le système de contrôle d'accès doit être capable de tenir compte des évolutions possibles au niveau des politiques. Il faut qu'il soit possible de définir et de changer les politiques au cours du processus de collaboration. En effet, la structure et la configuration de la collaboration peut changer en fonction des besoins organisationnels ou de l'application, ce qui peut entraîner la mise à jour ou la redéfinition des politiques de sécurité.
6. **Cohérence** L'ensemble des politiques de sécurité doivent être cohérent afin d'assurer la fiabilité du système et éviter les accès malveillants.

2.2.2 Exigences des modèles de contrôle d'accès

Les modèles de contrôle d'accès permettent de spécifier la méthode de gestion d'accès aux ressources et comment les décisions d'autorisation sont prises. Plusieurs exigences ont été définies dans [JP96][TAPH05][BDEKK08] dont les principales sont :

1. **Applicabilité** Cette exigence présente la capacité de déployer le modèle de contrôle d'accès dans différentes infrastructures. Le modèle doit également assurer le niveau demandé de sécurité avec un coût acceptable. Le modèle doit être aussi assez générique afin de répondre à la grande variété des environnements de collaboration et aux différents modèles d'entreprise.
2. **Accès transparent** Le modèle de contrôle d'accès doit assurer un accès transparent aux ressources et interdire les accès qui ne sont pas autorisés sans interrompre le processus de collaboration. En plus, le processus de prise de décision d'autorisation doit être exécuté d'une manière transparente aux utilisateurs.
3. **Spécification et application des politiques** Le modèle de contrôle d'accès doit fournir des méthodes pour la spécification des politiques afin de sécuriser l'accès aux ressources dans les environnements collaboratifs. Ils doivent également fournir des méthodes qui permettent d'assurer la bonne spécification et évaluation de ces politiques.

En plus de ces exigences, plusieurs critères peuvent caractériser les modèles de contrôle d'accès [TAPH05]. Nous citons ci-dessous les principaux critères de ces modèles.

- **Facilité d'utilisation :** Ce critère indique le niveau de simplicité de la gestion des règles d'autorisation pour les utilisateurs finaux. Parfois les utilisateurs refusent d'utiliser un modèle de contrôle d'accès pour la simple raison qu'il n'est pas facile de spécifier et modifier les politiques. Les systèmes de contrôle d'accès sont souvent complexes, ce qui rend les tâches de gestion des autorisations un peu difficile pour les utilisateurs qui n'ont pas forcément une grande maîtrise du modèle utilisé.
- **Complexité :** Les fonctionnalités offertes par un modèle de contrôle d'accès ne doivent pas augmenter le degré de sa complexité. Un modèle très complexe rend son implémentation difficile et des problèmes divers peuvent apparaître lors de son exécution.
- **Support de la collaboration (groupes d'utilisateurs) :** Ce critère indique la possibilité d'utiliser et d'appliquer le modèle dans les environnements collaboratifs. Presque tous les modèles peuvent être appliqués dans un environnement mono-utilisateur, mais seulement certains d'entre eux peuvent être appliqués dans des environnements collaboratifs où des utilisateurs collaborent au sein des groupes pour achever ensemble des tâches particulières. Dans les environnements collaboratifs, les utilisateurs collaborent dans des groupes afin d'atteindre des objectifs communs. Ce critère indique la capacité du modèle de contrôle d'accès à spécifier des droits d'accès pour un groupe d'utilisateurs.
- **Granularité :** Dans certains environnements collaboratifs, les applications ont besoin d'un niveau de granularité bien particulier. En effet,

dans certaines situations, la spécification des droits d'accès pour un groupe d'utilisateurs ne suffit pas et on peut avoir besoin, par exemple, de donner des droits à un utilisateur pour accéder à une ressource juste au cours d'un intervalle de temps bien défini durant le processus de collaboration.

– **Utilisation des informations contextuelles :** L'utilisation des informations contextuelles lors de la spécification des droits d'accès permet de rajouter d'autres niveaux de granularité dans le modèle de contrôle d'accès. Dans certains environnements collaboratifs, les informations contextuelles des utilisateurs sont très utilisées et surtout lorsque ces informations sont essentielles pour la spécification des droits d'accès. Plusieurs informations contextuelles peuvent être utilisées telles que la position géographique, la machine utilisée, le rôle, etc.

2.2.3 Modèles de contrôle d'accès

Plusieurs modèles de contrôle d'accès ont été définis dans la littérature. Dans cette sous-section, nous présentons une synthèse de ces modèles.

Matrice de contrôle d'accès (Access Control Matrix)

En 1971, Lampson [Lam74] a commencé à définir les règles à suivre lors de la spécification des politiques de contrôle d'accès en définissant la matrice de contrôle d'accès. Cette matrice est un modèle conceptuel du contrôle d'accès discrétionnaire (DAC) qui spécifie les droits d'accès que possède chaque sujet pour chaque objet. L'objet représente l'entité ou la ressource à laquelle on peut accéder, tandis que le sujet représente l'entité qui accède à l'objet. Dans une matrice de contrôle d'accès, les lignes représentent les sujets et les colonnes représentent les objets. Un élément $A[i, j]$ d'une matrice A représente les droits d'accès que le sujet i a pour l'objet j. Pour chaque requête d'un sujet i pour effectuer une action a sur un objet j, la règle de contrôle d'accès vérifie si l'élément $A[i, j]$ de la matrice contient le droit a. Si c'est le cas, le sujet sera autorisé. Le Tableau 2.1 représente un exemple de matrice de contrôle d'accès. Dans cet exemple, les symboles R et W représentent respectivement les droits de lecture et d'écriture. Dans cette matrice, l'utilisateur $U1$ est le propriétaire du fichier $F1$ et il a les droits R et W de ce fichier. Néanmoins cet utilisateur ne peut pas accéder au fichier $F3$ et n'a que le droit R du fichier $F2$. L'utilisateur $U1$, étant le propriétaire du fichier $F1$, a donné le droit R pour l'utilisateur $U2$ et le droit W pour l'utilisateur $U3$.

En réalité, les systèmes de sécurité n'implémentent pas ce modèle en tant que matrice, mais ils l'implémentent en utilisant deux listes. Une première liste qui représente les lignes de la matrice, appelée liste de capacités C-List (*Capability list*), et une deuxième liste qui représente les colonnes de la matrice, appelée liste de contrôle d'accès ACL (*Access Control List*). Dans les ACLs, on présente pour chaque objet la liste des sujets qui peuvent y accéder et avec quels droits d'accès. Dans les C-Lists, on présente pour chaque sujet la liste des objets auxquels il peut accéder et avec quels droits d'accès. Ces deux modèles sont encore utilisés dans plusieurs systèmes pour la définition des droits d'accès notamment dans les systèmes d'exploitation (ex : UNIX).

objet / sujet	F1	F2	F3
U1	own R W	R	
U2	R	own R W	W
U3	W		own R W

Tab. 2.1 – *Exemple de matrice de contrôle d'accès*

Le problème dans les ACLs est qu'il est difficile de déterminer tous les droits d'accès qu'un sujet possède puisqu'il faut parcourir la liste de chaque objet et déterminer les droits de ce sujet. Pour les mêmes raisons, il est difficile de supprimer les droits d'un sujet s'il quitte le système. Par conséquent, il est difficile de changer ou de supprimer dynamiquement les droits d'un sujet. Ce modèle s'applique surtout dans les environnements où les changements des droits d'accès sont peu fréquents. Dans un environnement complexe avec un grand nombre d'utilisateurs et de ressources, les droits d'accès deviennent difficiles à gérer avec les ACLs ou les C-Lists.

Contrôle d'autorisations basé sur les tâches (Task-Based Authorization Controls TBAC)

Selon Thomas et Sadhu [TS98], les modèles qui se basent sur la matrice de contrôle d'accès sont très limités vu qu'ils ne tiennent pas compte d'autres informations contextuelles lors de l'évaluation des requêtes. Ils proposent d'étendre les modèles traditionnels avec des informations contextuelles basées sur les tâches des utilisateurs. Dans ce modèle, les permissions sont représentées par des *authorization-steps* qui dépendent de l'avancement des tâches des utilisateurs.

Ce modèle peut supporter la gestion dynamique des permissions dans les environnements collaboratifs. Néanmoins, il est limité à un type de contexte bien particulier qui est basé sur l'avancement des tâches des utilisateurs et il ne peut être appliqué que dans les systèmes collaboratifs où la spécification des droits d'accès doit dépendre de l'avancement des tâches.

Contrôle d'accès spatial (Spatial Access Control SAC)

En 1999, Bullock [Bul99] a proposé un modèle de contrôle d'accès pour les environnements collaboratifs virtuels, appelé *SPACE*. Ce modèle se base sur les mouvements et les navigations dans l'environnement pour la définition des politiques de sécurité. Ce modèle se compose de deux composants principaux : la limite (*boundary*) et le graphe d'accès. La limite permet de diviser l'environnement collaboratif en régions, alors que le graphe d'accès permet de spécifier des contraintes et des possibilités de mouvement dans l'environnement et de gérer l'accès.

Contrôle d'accès basé sur les rôles (Role-Based Access Control)

Le contrôle d'accès basé sur les rôles (*Role-Based Access Control* ou RBAC) [FCAG03] [FSG+01] est un modèle principal pour la gestion d'accès aux ressources. Il propose une nouvelle méthode pour l'attribution des droits d'accès aux individus. Il a été proposé comme une alternative du contrôle d'accès discrétionnaire et du contrôle d'accès obligatoire afin de réduire la complexité et le coût d'administration des politiques de contrôle [BGBJ05]. Le principe fondamental de RBAC est que la décision d'autorisation d'accès à une ressource est basée sur le rôle de l'utilisateur. Une fois que les rôles dans le système sont définis, des permissions sont attribuées à ses rôles. Ensuite, les utilisateurs auront des permissions en fonction de leurs rôles.

La Figure 2.1 décrit les relations entre les utilisateurs, les rôles et les permissions dans le modèle RBAC. Cette figure peut être décrite comme suit : un utilisateur peut avoir un ou plusieurs rôles ; et un rôle peut être attribué à un ou plusieurs utilisateurs. Une permission associe un ensemble de droits à un objet. Une permission peut être attribuée à un ou plusieurs rôles et un rôle peut avoir plusieurs permissions.

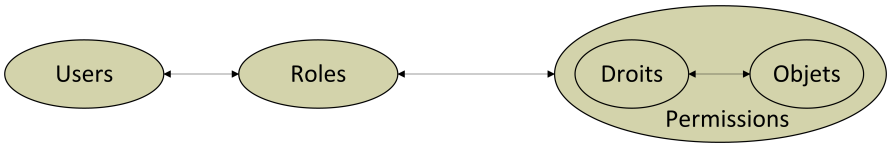

FIGURE 2.1 – *Élements du modèle RBAC*

Cette approche de contrôle a été bien étudiée dans les deux dernières décennies et plusieurs modèles RBAC ont été développés. En 1992, Feraiolo et Kuhn ont développé le premier modèle RBAC [FK92]. En 1995, ils l'ont bien raffiné en définissant tous les termes du modèle RBAC et ont validé son utilisation et sa faisabilité [FCK95]. En 1996, Sandhu et al. [SCFY96] proposent la famille la plus connue des modèles RBAC. En 2001, NIST a proposé un modèle standard pour RBAC. En 2004, ANSI (The American National Standard Institute) a proposé le standard ANSI RBAC [3] en se basant sur [SFK00] et [FSG+01]. Ce standard définit un modèle RBAC de référence et fournit les caractéristiques que tout système RBAC doit avoir. Il propose des définitions des termes suivants :

- Utilisateur : Un utilisateur est défini comme un être humain. Ce terme d'utilisateur peut être étendu pour représenter les machines, les agents autonomes intelligents et les réseaux.
- Rôle : Un rôle représente la fonction d'un utilisateur dans une organisation et il peut aussi représenter une responsabilité.
- Objet : Un objet représente chaque ressource dans le système telle qu'un fichier, une imprimante, une base de données, etc.
- Opération : Une opération est une commande exécutable d'un programme qui exécute des fonctions pour l'utilisateur après son invocation.
- Permission : Une permission est un accord pour exécuter une opération sur un ou plusieurs objets.

3. disponible sur http://csrc.nist.gov/rbac/rbac-std-ncits.pdf

Trois modèles de référence ont été définis dans le standard : *Core* RBAC, *Hierarchical* RBAC et le *Constrained* RBAC.

Le modèle *Core* RBAC définit les éléments et les relations essentiels pour la construction d'un système RBAC. Il définit les associations *user-role* et *permission-role* qui sont fondamentales dans chaque système RBAC. Le modèle *Core* RBAC introduit ainsi le concept d'activation de rôle. Les éléments et les relations du modèle *Core* RBAC sont décrits dans la Figure 2.2. Les éléments présentés dans la figure sont les suivants : les utilisateurs (USERS), les rôles (ROLES), les objets (OBS), les opérations (OPS), et les permissions (PRMS). En plus de ces éléments, le modèle Core RBAC définit le concept de session (SESSIONS) qui représente un mappage entre un utilisateur et l'activation d'un ensemble de ses rôles. Cela veut dire qu'un utilisateur peut établir une session durant laquelle il active un ensemble de rôles. Les deux relations UA et PA, décrites dans la figure, présentent respectivement deux relations plusieurs-à-plusieurs entre les utilisateurs et les rôles d'une part et entre les permissions et les rôles d'autre part. La relation *session_roles* permet de déterminer les rôles activés dans une session et la relation *User_sessions* permet de déterminer l'utilisateur associé à une session.

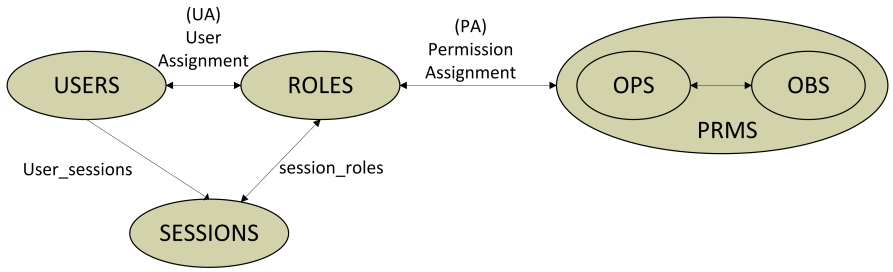

Figure 2.2 – Core *RBAC*

Le *Hierarchical* RBAC introduit des hiérarchies de rôles qui permettent de structurer les rôles afin de refléter les niveaux d'hiérarchie dans une organisation. Une hiérarchie de rôles définit des relations d'héritage entre les rôles. Dans le modèle *Hierarchical* RBAC, un rôle $r1$ hérite d'un rôle $r2$ si tous les privilèges de $r2$ sont aussi des privilèges de $r1$. Le standard définit deux types de hiérarchie de rôles : des hiérarchies de rôles générales (*General Role Hierarchies*) et des hiérarchies de rôles limitées (*Limited Role Hierarchies*). La hiérarchie de rôles générale est un ordre partiel arbitraire qui représente l'ensemble des rôles R. Cette hiérarchie supporte le concept d'héritage multiple qui offre la possibilité d'hériter des permissions de deux ou plusieurs rôles dans le système. Tandis que la hiérarchie des rôles limitée impose la définition des rôles sous forme d'un arbre plus simple où un rôle ne peut avoir qu'un seul descendant.

Le *Constrained* RBAC définit des contraintes lors de l'attribution des rôles aux utilisateurs et lors de l'activation des rôles. Ces contraintes, appelées contraintes de séparation de droit (*Separation of Duty Constraints*), sont utilisées pour empêcher les utilisateurs de dépasser un niveau raisonnable d'autorité et pour limiter les conflits potentiels que peuvent présenter les politiques dans un système de sécurité. Afin de modéliser ses contraintes, deux types de relations ont été définis : séparation de droits statique (*Static Separaion of Duty* - SSD) et séparation de droits dynamique (*Dynamic Separation of*

Duty - DSD). Les contraintes statiques (SSD) permettent d'imposer des restrictions sur l'attribution des rôles aux utilisateurs. Cela veut dire que si un rôle $r1$ est attribué à un utilisateur, on ne peut pas lui attribuer d'autres rôles qui peuvent présenter un conflit avec le rôle $r1$. Ces contraintes sont très utilisées dans plusieurs environnements. Par exemple, dans une université, les deux rôles "président de l'université" et "étudiant" ne peuvent pas être attribués à un seul utilisateur. En définissant ce type de contraintes, on réduit le nombre de permissions que peut avoir un utilisateur. Les autres contraintes, dites dynamiques (DSD), permettent d'éviter les conflits lors de l'activation des rôles. Elles permettent à un utilisateur d'activer un ou plusieurs rôles simultanément à condition que cette activation ne crée pas un conflit.

Contrôle d'accès basé sur les équipes (Team-Based Access Control TMAC)

La notion d'équipe permet de grouper les utilisateurs d'une organisation. Dans les modèles RBAC, un groupe est un ensemble d'utilisateurs qui jouent le même rôle. Néanmoins, dans certains environnements, des utilisateurs peuvent avoir plusieurs rôles dans un seul groupe. Le modèle TMAC, proposé par Thomas [Tho97], permet d'associer des droits d'accès à un groupe d'utilisateurs qui peuvent avoir des rôles différents et dont le but est d'atteindre un objectif particulier. TMAC définit deux aspects importants du contexte de collaboration : le contexte des utilisateurs qui permet d'identifier les utilisateurs qui jouent un rôle dans une équipe à un moment donné ; et le contexte des objets qui permet d'identifier les objets requis dans le processus de collaboration. En 2001, Georgiadis et al. [GMPT01] ont proposé le modèle C-TMAC qui rajoute d'autres informations contextuelles telles que la position géographique de l'utilisateur.

Contrôle d'accès sensible au contexte (Context-Aware Access Control Context-AW-AC)

Covington et al. [CLS+01] ont proposé un modèle de contrôle d'accès pour sécuriser les applications sensibles au contexte dans les environnements ubiquitaires. Ils proposent une extension du modèle RBAC avec la notion de rôles d'environnement. Ces rôles sont utilisés pour capturer le contexte des utilisateurs et de l'environnement et peuvent influer sur les décisions d'autorisation. Selon les auteurs, une décision d'autorisation dépend de l'état de l'environnement au moment de l'accès à un objet.

Contrôle d'accès basé sur les attributs (Attribute Based Access Control ABAC)

Le modèle ABAC [PFMP04] [YT05] est apparu avec le standard *eXtensible Access Control Markup Language* (XACML) [Mos05] afin d'assurer une meilleure flexibilité. Dans ce modèle, les propriétés des utilisateurs et des objets sont exprimées avec des attributs. Un attribut est une propriété exprimée sous forme d'une paire (nom,valeur) et associée à une entité dans le système. Une entité peut être un sujet, une action, une ressource ou un environnement. Le modèle ABAC permet de définir des politiques de contrôle d'accès en se basant sur ces attributs. Les deux modèles RBAC et ABAC peuvent être utilisés ensemble si on considère les rôles comme attributs des

utilisateurs. Néanmoins, le fait de considérer un rôle comme un attribut, fait perdre tous les avantages de RBAC parce qu'on ne peut plus considérer qu'un rôle représente un ensemble de permissions.

Comparaison entre RBAC et les autres modèles de contrôle d'accès

Une analyse comparative des modèles de contrôle d'accès a été réalisée dans [TAPH05]. Cette étude évalue les modèles de contrôle d'accès et présente leurs niveaux de satisfaction aux critères présentés dans la section 2.2.2 et à d'autres critères. Le Tableau 2.2 présente une évaluation des modèles de contrôle d'accès par rapport aux critères suivants : niveau de facilité d'utilisation, niveau de complexité, support de la collaboration, niveau de granularité et niveau d'utilisation des informations contextuelles. Dans ce qui suit, nous analysons ce tableau et comparons le modèle RBAC avec les autres modèles.

Critère	RBAC	Matrice	TBAC	SAC	TMAC	Context-AW-AC
Niveau de facilité d'utilisation	**élevé**	moyen	moyen	**bas**	**élevé**	**élevé**
Niveau de complexité	moyen	bas	moyen	bas	moyen	**élevé**
Support de la collaboration (groupes d'utilisateurs)	oui	**faible**	oui	oui	oui	oui
Granularité	faible	non	faible	non	**oui**	**oui**
Utilisation des informations contextuelles	faible	non	moyen	moyen	moyen	moyen

Tab. 2.2 – *Évaluation des modèles de contrôle d'accès*

Les politiques de contrôle d'accès peuvent être définies par les utilisateurs, les administrateurs ou par les activités de collaboration [TAK+04a]. Dans notre travail, les politiques de contrôle d'accès sont spécifiées par les utilisateurs qui gèrent des ressources dans les sessions de collaboration et qui ne sont pas forcément des experts de gestion de contrôle d'accès. C'est pour cette raison que nous considérons que les deux premiers critères du tableau, niveau de facilité d'utilisation et niveau de complexité, sont des critères fondamentaux que doivent satisfaire les modèles de contrôle d'accès dans les environnements de travail collaboratif. Supporter la manipulation et la spécification de droits d'accès pour des groupes d'utilisateurs représente aussi un critère important dans ce type d'environnement. Les deux autres critères, granularité et utilisation des informations contextuelles, sont aussi importants dans les environnements ubiquitaires mais moins importants que les autres critères dans les environnements de travail collaboratif ubiquitaires où ce sont les utilisateurs qui gèrent l'accès aux ressources. Dans ce travail nous n'utilisons pas les informations contextuelles pour la spécification des droits d'accès mais nous les utilisons pour déterminer les

mécanismes de contrôle d'accès qu'on doit mettre en place pour sécuriser l'accès aux ressources.

D'après le Tableau 2.2, nous remarquons que spécifier des droits d'accès pour des groupes d'utilisateurs (3ème critère dans le tableau) est très primitif et peu élaboré par les matrices de contrôle d'accès. Contrairement aux autres modèles, la matrice de contrôle est généralement utilisée dans les environnements mono-utilisateur dans lesquels la notion de groupe d'utilisateurs n'est pas fondamentale. Le modèle RBAC, par exemple, supporte la définition de droits d'accès pour un groupe d'utilisateurs qui jouent le même rôle dans le processus de collaboration. Par conséquent, le modèle de matrice de contrôle d'accès ne peut pas être utilisé dans les environnements collaboratifs. Nous remarquons aussi que le modèle SAC supporte la définition des droits d'accès pour des groupes d'utilisateurs. Néanmoins, son utilisation n'est pas facile dans les environnements collaboratifs.

D'après le Tableau 2.2, les modèles RBAC, TMAC et Context-AW-AC sont les plus faciles à utiliser dans les environnements collaboratifs. Par exemple, dans RBAC, le découplage entre l'attribution des rôles aux utilisateurs et l'attribution des permissions aux rôles simplifie énormément la gestion des autorisations. En effet, si un utilisateur doit changer de position dans une organisation ou dans n'importe quel système collaboratif, il suffit de lui attribuer un nouvel ensemble de rôles sans lui attribuer à nouveau un ensemble de permissions.

Contrairement aux autres modèles, les modèles TMAC et Context-AW-AC permettent de supporter une grande granularité dans les politiques de contrôle d'accès. Cependant, le modèle Context-AW-AC est complexe par rapport aux autres modèles, ce qui peut causer des problèmes d'implémentation. Dans le modèle ABAC, qui n'est pas présenté dans le tableau, la gestion des attributs peut devenir une activité complexe. En effet dans les grands réseaux où il y a un grand nombre de ressources, d'utilisateurs et d'applications, un grand nombre d'attributs et de politiques de contrôle d'accès doivent être gérés. En outre, l'augmentation des points de configuration des politiques rend la compréhension des politiques beaucoup plus difficile par apport aux autres modèles tels que RBAC et la matrice de contrôle d'accès [JKS12].

D'après ce qui précède, les modèles qu'on peut adopter pour le contrôle d'accès sont RBAC, TBAC et TMAC. Malgré que le modèle RBAC n'utilise pas les informations contextuelles lors de la prise des décisions d'autorisation et ne supporte pas une grande granularité, nous l'utiliserons comme modèle principal de contrôle d'accès pour les raisons suivantes :

- Il existe des langages standardisés pour l'implémentation du modèle RBAC tels que le langage XACML [Mos05] qui est un langage standard proposé par OASIS.
- RBAC est devenu un modèle prédominant pour le contrôle d'accès vu qu'il réduit le coût et la complexité d'administration des politiques de sécurité et qu'il répond aux besoins des organisations. En effet, il offre une meilleure flexibilité dans l'administration des politiques de sécurité par rapport aux approches traditionnelles qui utilisent les identifiants.
- RBAC permet aux systèmes de passer à l'échelle puisqu'il ne se base pas sur l'identité de l'utilisateur pour définir les droits d'accès.

2.2.4 Langage standard de contrôle d'accès : XACML

XACML (*eXtensible Access Control Markup Language*) est un langage basé sur XML dédié au contrôle d'accès. Il a été standardisé par OASIS en 2005 [Oas05]. Ce langage a été proposé pour l'implémentation du modèle ABAC et l'expression de ses politiques en se basant sur les attributs. XACML définit trois niveaux de politiques : ensemble de politiques (*PolicySet*), politique (*Policy*) et règle (*Rule*). La Figure 2.3 représente la structure des politiques dans XACML. Un *PolicySet* est une agrégation de plusieurs politiques (*Policy*) et éventuellement d'autres *PolicySet*. Une politique se compose d'un ensemble de règles de contrôle d'accès. Chaque règle est composée d'un effet (*Effect*) et d'une cible (*Target*). Un effet indique le résultat de l'évaluation d'une règle. Il peut avoir comme valeur *Permit* (autorisé) ou *Deny* (non autorisé). Une cible est un ensemble de propriétés associées aux sujets, aux ressources, aux actions et à l'environnement qui permettent d'identifier les règles et les politiques appropriées pour l'évaluation d'une requête donnée. Cette notion de cible de XACML a été introduite afin de facilité la recherche des règles qui doivent s'appliquer à une requête donnée. En plus de l'effet et de la cible, une règle peut contenir optionnellement une condition qui représente une expression booléenne qui permet de définir des restrictions plus complexes sur les attributs.

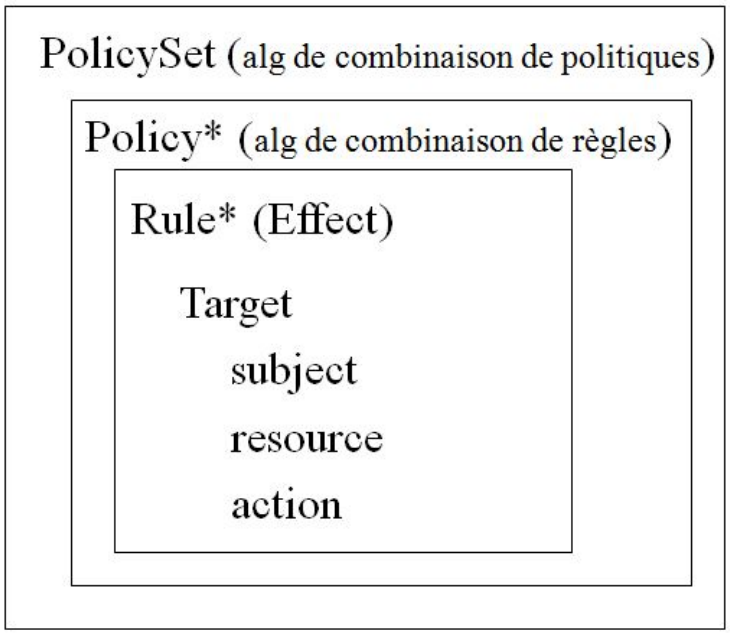

FIGURE 2.3 – *Schémas des politiques XACML*

Puisqu'un *PolicySet* peut contenir plusieurs politiques et qu'une politique peut contenir plusieurs règles, différentes décisions peuvent être prises dans un même *PolicySet* et dans une même politique. Afin de résoudre les incohérences des différentes décisions, XACML définit des algorithmes de combinaison. Les *PolicySet* utilisent des algorithmes de combinaison de politiques (*policy-combining algorithm*) afin de combiner les décisions des différentes politiques et générer une seule décision. Tandis que les politiques (*Policy*) utilisent des algorithmes de combinaison de règles (*rule-combining*

algorithm) afin de générer une seule décision à partir des décisions des différentes règles.

La décision retournée peut être :

- Permis (*Permit*) : Cette décision indique que l'accès est autorisé ;
- Interdit (*Deny*) : Cette décision indique que l'accès est refusé ;
- Indéterminé (*Indeterminate*) : Cette décision indique qu'il y a une erreur et/ou qu'un attribut manque ou est inconnu ;
- Non applicable (*NotApplicable*) : Cette décision indique qu'il n'y a aucune règle qui peut s'appliquer à la requête.

Quatre algorithmes de combinaison ont été définis dans la spécification 2.0 du standard XACML :

- *Permit-overrides* : s'il existe au moins une règle/politique applicable et évaluée avec un effet "Permit" alors la décision de combinaison est "Permit".
- *Deny-overrides* : s'il y a au moins une règle/politique applicable et évaluée avec un effet "Deny" alors la décision de combinaison est "Deny".
- *First-applicable* : la décision de combinaison est la décision de la première règle/politique applicable dans la liste. L'ordre des règles/politiques est important dans cet algorithme.
- *Only-one-applicable* : le résultat de combinaison assure qu'il existe une seule règle/politique applicable dans la liste. S'il n y a aucune règle/politique applicable alors le résultat de combinaison est "*NotApplicable*", si plusieurs règles/politiques sont applicables alors le résultat de combinaison est "*Indeterminate*" ; si une seule règle/politique est applicable alors la décision retournée est celle de cette règle/politique.

La spécification de la version 3.0 du standard ajoute une extension de ces algorithmes comme suit :

- *Ordered-permit-overrides* : cet algorithme utilise le même principe que "*Permit-overrides*", sauf que les règles/politiques sont évaluées dans l'ordre dans lequel elles sont définies.
- *Ordered-deny-overrides* : la même chose pour cet algorithme.
- *Deny-unless-permit* : Cet algorithme est destiné à être utilisé dans les cas où la décision "Permit" est prioritaire à la décision "*Deny*". L'utilisation de cet algorithme garantie d'avoir toujours une décision "*Permit*" ou une décision "*Deny*". Les décisions "*Indeterminate*" et "*NotApplicable*" ne sont jamais retournées.
- *Permit-unless-deny* : Le même principe pour cet algorithme qui est destiné à être utilisé dans les cas où la décision "*Deny*" est prioritaire à la décision "*Permit*".
- *Legacy Deny-overrides* : Cet algorithme est un peu plus complexe que les autres et il est utilisé dans les cas où la décision "*Deny*" a une priorité par rapport à la décision "*Permit*"
- *Legacy Permit-overrides* : Cet algorithme est destiné à être utilisé dans les cas où la décision "*Permit*" a une priorité par rapport à la décision "*Deny*".

Les principales entités de l'architecture XACML sont les suivants :

- Le point de décision de politiques (PDP - *Policy Decision Point*) : C'est l'entité qui détermine les politiques applicables à une requête et ren-

voie une décision d'autorisation. Ce terme a été défini par l'IETF et le DMTF dans le RFC3198[4] .

- Le point d'application de politiques (PEP - *Policy Enforcement Point*) : C'est l'entité qui intercepte la requête et l'envoie pour l'évaluer au niveau PDP. Elle contrôle l'accès en appliquant la décision retournée par le PDP. Cette entité est aussi définie dans le RFC3198. Elle peut être incorporée dans l'application ou placé comme intercepteur en dehors de l'application.
- Le point d'information de politiques (PIP - *Policy Information Point*) : C'est l'entité qui extrait des informations supplémentaires pour le PDP avant de prendre la décision.
- Le point d'administration de politiques (PAP - *Policy Administration Point*) : C'est l'entité du système qui définit les politiques et les rend disponibles au PDP.

La spécification 2.0 a définit une architecture de XACML sous forme d'un diagramme de flots de données. La Figure 2.4 présente une vue simplifiée de ce diagramme.

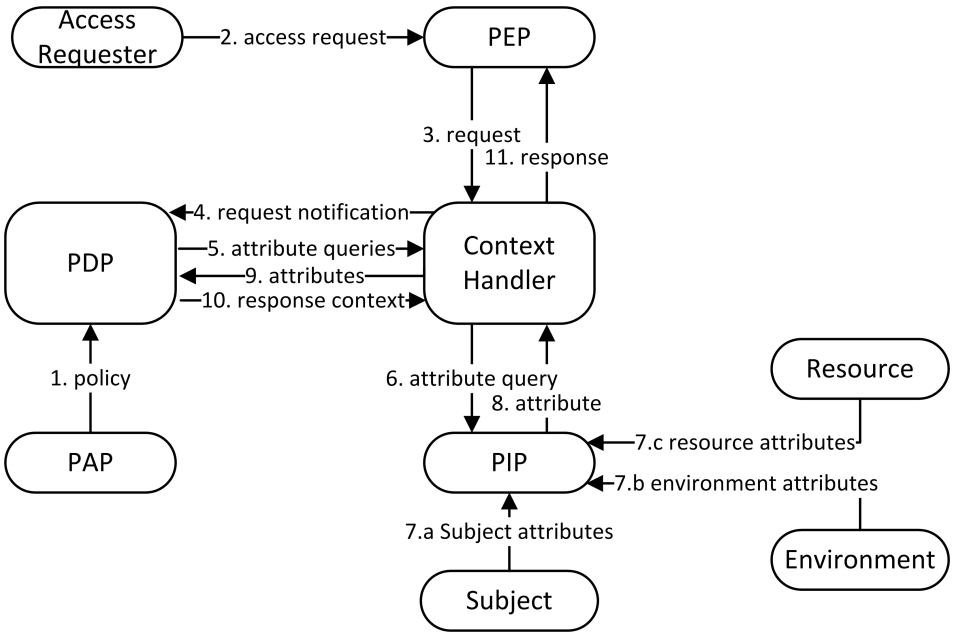

FIGURE 2.4 – *Diagramme de flots de données XACML [Oas05]*

Les différentes étapes du diagramme sont décrites comme suit :

1. Le PAP définit les politiques ou l'ensemble de politiques et les rend disponibles au PDP.
2. Le demandeur d'accès (*Access Requester*) envoie une requête d'accès au PEP.
3. Le PEP envoie la requête au format natif au gestionnaire de contexte (*Context Handler*).
4. Le gestionnaire de contexte traduit la requête au format standard et l'envoie au PDP.

4. http://www.ietf.org/rfc/rfc3198.txt

5. Le PDP peut demander au gestionnaire de contexte des attributs supplémentaires.
6. Le gestionnaire de contexte demande à son tour ces attributs au PIP.
7. Le PIP obtient les attributs demandés.
8. Le PIP retourne les attributs au gestionnaire de contexte.
9. Le gestionnaire de contexte envoie les attributs au PDP qui évalue la politique.
10. Le PDP retourne la décision au format standard au gestionnaire de contexte.
11. Le gestionnaire de contexte traduit la réponse au format natif utilisé par le PEP et l'envoie à ce dernier.

Comme nous l'avons mentionné, le standard XACML a été proposé comme un standard pour l'utilisation du modèle ABAC. Afin de répondre aux besoins des applications qui utilisent le modèle RBAC, OASIS a défini un profil pour RBAC [And05] qui répond aux exigences de la norme ANSI RBAC.

Ce profil propose des politiques (*Policy*) et des ensembles de politiques (*PolicySet*) spécifiques pour l'expression des politiques du modèle RBAC avec le langage XACML comme suit :

- *Permission* <*PolicySet*> ou PPS : définit les permissions associées à un rôle. Un PPS peut faire référence à un autre PPS et donc il hérite toutes les permissions définies dans le PPS référencé.
- *Role* <*PolicySet*> ou RPS : associe un rôle à un PPS. De ce fait, le rôle aura toutes les permissions définies dans le PPS. Si un rôle $r1$ est associé à un PPS $p1$ et un autre rôle $r2$ est associé à un PPS $p2$ et $p1$ fait référence à $p2$, alors le rôle $r1$ aura les permissions déjà définies dans $p1$ et hérite les permissions du rôle $r2$ qui sont définies dans $p2$.
- *Role Assignment* <*Policy*> ou <*PolicySet*> : spécifie quels rôles peuvent être attribués à quels sujets. L'utilisation de ces politiques est optionnelle.
- *HasPrivilegesOfRole* <*Policy*> : C'est une politique dans un PPS qui permet de savoir si un sujet a les permissions associées à un rôle donné.

Le modèle RBAC doit être implémenté en utilisant les PPSs et les RPSs. Pour chaque rôle, un RPS et un PPS doivent être définis. La hiérarchie des rôles est réalisée à travers le mécanisme de référencement entre les PPSs.

2.2.5 Contrôle d'accès et informatique ubiquitaire

Dans les environnements ubiquitaires, les ressources sont partagées par des utilisateurs qui utilisent des dispositifs hétérogènes. La variété des applications et des interactions des utilisateurs avec l'environnement introduit des problèmes de sécurité. En outre, l'hétérogénéité des dispositifs mobiles et des utilisateurs dans un tel environnement dynamique rend l'administration des politiques de sécurité beaucoup plus complexe dans les environnements ubiquitaires que dans les environnements traditionnels.

Dans les environnements ubiquitaires, les applications doivent répondre aux questions suivantes : Quelles sont les entités qui doivent interagir ? Quelles informations doivent être disponibles pour une entité donnée ?

Wang et al. [WZC08] identifient des problèmes de sécurité qui doivent être adressés dans les environnements ubiquitaires. Ils mettent l'accent sur les exigences qui doivent être satisfaites pour l'élaboration d'une architecture de contrôle d'accès flexible et sécurisée :

- Contrôle d'accès dans les environnements collaboratifs [CKK+04] : L'informatique ubiquitaire est souvent utilisée dans les environnements collaboratifs où plusieurs utilisateurs travaillent ensemble pour achever des tâches spécifiques. Un modèle de contrôle d'accès doit fournir une structure de collaboration sécurisée pour le partage des ressources.
- Administration décentralisée [WZCV03] [Din03] : L'administration des politiques de sécurité doit être décentralisée dans les différentes partitions de l'environnement ubiquitaire.
- Tenir compte des informations contextuelles (aspect dynamique) : Dans les systèmes traditionnels, les utilisateurs et les objets sont statiques et les changements du contexte sont peu fréquents. Cependant, dans les environnements ubiquitaires, les utilisateurs peuvent rejoindre et quitter un processus de collaboration d'une manière dynamique. En effet, un utilisateur peut rejoindre ou quitter les sessions de collaboration en fonction des besoins ; il peut également changer de session au cours du processus de collaboration. Dans ce type d'environnements, les applications sont dynamiques de nature i.e. les ressources peuvent être crées et modifiées, les entités qui utilisent les ressources peuvent changer et le type d'accès peut changer durant l'application.

2.3 Représentation des connaissances

Dans cette section, nous présentons des techniques de représentation de connaissances et nous citons les principaux travaux qui ont utilisé les ontologies pour l'implantation des systèmes de contrôle d'accès qui se basent sur le modèle RBAC.

2.3.1 Définitions

La représentation des connaissances est un champ d'étude de l'intelligence artificielle. Il se concentre principalement sur les méthodes de représentation de connaissances et le traitement de ces connaissances par des mécanismes de raisonnement d'une façon automatisé [BL04]. L'avantage principal de la représentation des connaissances est qu'elle permet de décrire le comportement des systèmes complexes en utilisant différents vocabulaires.

Une connaissance met en relation un expert, qui peut être un humain, et une formule logique qui peut être une proposition ou un prédicat. Par exemple, dans l'expression « *Philippe sait que Julie va aller à la fête* », le connaisseur est Philippe, alors que la proposition est « *Julie va aller à la fête* » [BL04]. La représentation des connaissances consiste à utiliser des symboles formels pour représenter un ensemble de formules logiques.

Le raisonnement est le traitement formel des symboles d'une représentation de connaissances afin de produire une représentation de nouvelles

connaissances [BL04]. Le raisonnement est un concept important ; il représente le moyen qui permet d'atteindre l'objectif essentiel de la représentation des connaissances.

L'ensemble des symboles d'une représentation de connaissance est souvent appelé base de connaissances (*Knowledge Base* - KB). Un système à base de connaissances a la capacité de capturer des changements dans l'environnement et de s'adapter en fonction de ces changements [BL04]. Ce qui fait que la représentation des connaissances et le raisonnement sont des moyens essentiels utilisés par les systèmes adaptatifs.

2.3.2 Les ontologies et la représentation des connaissances

Les ontologies sont parmi les modèles les plus connus dans le domaine de l'ingénierie des connaissances. Ils visent à établir une représentation sémantique des connaissances d'un certain domaine pour qu'elles puissent être traitées et manipulées par les machines. Selon Gruber [Gru93], « *une ontologie est une spécification explicite d'une conceptualisation* ». « *Une conceptualisation est une vue simplifiée et abstraite de l'environnement qu'on veut représenter* ». L'étape de conceptualisation consiste à identifier les connaissances spécifiques à un domaine donné. Elle est nécessaire avant la construction d'une ontologie.

Plusieurs classifications des ontologies ont été proposées [GP99] :

- Ontologie de haut niveau [Gua97] : Ce sont des ontologies générales qui représentent les catégories des choses dans le monde. Ces ontologies représentent les concepts de haute abstraction tels que les évènements, le temps, l'espace, etc.
- Ontologies génériques (appelées aussi méta-ontologies ou core ontologies) : Ces ontologies sont moins abstraites que les ontologies de haut niveau. Elles peuvent être réutilisées dans différents domaines.
- Ontologies de domaine [MKSK00] : Une ontologie de domaine est réutilisable dans un domaine spécifique. Ces ontologies utilisent des vocabulaires pour décrire des concepts relatifs à un domaine d'application bien particulier.

Perez [GP99] a identifié d'autres classes d'ontologies telles que les ontologies de tâches et les ontologies d'application.

2.3.3 Le langage d'ontologie web (Web Ontology Language - OWL)

Plusieurs langages ont été proposés pour la construction des ontologies. Les langages standardisés par le W3C sont Ressource Description Framework (RDF), RDF *Schema* (RDFS) et *Web Ontology Language* (OWL) [SWM04]. Dans la suite, nous nous focalisons sur le langage OWL qui se base sur RDF et qui utilise une syntaxe en XML. OWL permet l'expression des ontologies. Il permet principalement la représentation de concepts et de relations entre ces concepts. Il existe trois sous langages du langage OWL : *OWL-Lite*, *OWL-DL* (OWL-Description Logic) et *OWL-Full*. Chacun de ces sous langages est une extension du précédent.

- *OWL lite* est le sous langage le plus simple. Il permet la création des hiérarchies de classification et l'expression de contraintes simples.

- *OWL-DL* a été proposé afin de fournir plus d'expressivité tout en restant complet et décidable. Ce langage est appelé OWL-DL vu sa correspondance avec la logique de description. OWL est utilisé avec quelques restrictions. Par exemple, il ne supporte pas l'héritage multiple entre les concepts.
- *OWL-Full* assure une expressivité maximale et permet d'enrichir le vocabulaire prédéfini. Néanmoins la décidabilité n'est pas garantie avec ce sous langage.

Le choix entre ces trois sous langages dépend du niveau d'expressivité exigé par le développeur de l'ontologie. OWL-DL est le langage le plus utilisé car il se base sur la logique de description et vu qu'il permet d'effectuer du raisonnement. Le *Semantic Web Rule Language* (SWRL) [HPSB+04] a été proposé pour effectuer le raisonnement sur les ontologies OWL. C'est une extension d'OWL pour la définition des règles de raisonnement. SWRL permet l'inférence de nouvelles connaissances à partir des connaissances représentées dans une ontologie OWL.

2.3.4 Utilisation des ontologies pour le contrôle d'accès

Plusieurs travaux de recherche ont utilisé des techniques de représentation de connaissances pour le la contrôle d'accès basé sur les rôles [TMKL06] [DJYM05] [KHD08].

Cirio et al. [CCT07] étendent le modèle RBAC avec des attributs de contexte en utilisant les ontologies OWL. Ils proposent une ontologie de haut niveau qui peut être étendue par une ontologie de domaine pour l'expression de ces attributs. Heilili et al. [HCZ+06] proposent une approche basée sur les ontologies OWL qui permet d'étendre le modèle RBAC avec les autorisations négatives.L'approche proposée permet également de détecter les situations de conflits.

Finin et al. [FJK+08] décrivent deux approches différentes pour modéliser l'approche RBAC en utilisant les ontologies OWL. La première approche proposée permet de représenter la hiérarchie de rôles comme un ensemble de classes reliées par des relations d'héritage. Dans cette première approche, les contraintes de séparation de droit sont représentées en utilisant des classes disjointes. La deuxième approche proposée permet de représenter les rôles comme des attributs. Ferrini et al. [FB09] proposent un framework qui intègre les ontologies OWL et les politiques XACML. Ils étendent l'architecture XACML avec des relations sémantiques. Raje et al. [RDF+12] propose une implémentation du modèle RBAC en utilisant des ontologies OWL. Cette implémentation est utilisée dans un groupe de recherche dans le domaine de biologie.

Les ontologies sont largement utilisées dans les systèmes logiciels qui nécessitent un haut niveau d'autonomie et de flexibilité. L'utilisation des ontologies pour la gestion des droits d'accès permet d'assurer une description flexible des rôles, des permissions et des ressources du système. Elle permet également de profiter des techniques de raisonnement. Dans notre travail de thèse, nous utilisons le langage XACML au lieu des ontologies pour la spécification des droits d'accès. Ceci est justifié par le fait que XACML est un langage standard conçu spécifiquement pour la gestion des droits d'accès et il peut être utilisé dans différents domaines. De plus, l'utilisation des

ontologies pour la spécification des droits d'accès oblige toutes les organisations d'utiliser les éléments d'un même modèle partagé. Cependant, nous utilisons les ontologies pour capturer les situations de collaboration et déterminer les entités de contrôle d'accès qui doivent être mises en place à tout moment de la collaboration.

2.4 Synthèse des travaux existants et positionnement

Dans cette section, nous faisons une synthèse des travaux existants dans le domaine de contrôle d'accès dans les environnements collaboratifs ubiquitaires et nous discutons le niveau de satisfaction de ces travaux aux exigences des systèmes de contrôle d'accès. Nous commençons par présenter les mécanismes de contrôle d'accès qui se basent sur les informations contextuelles. Ensuite, nous présentons les mécanismes de contrôle d'accès qui étendent le modèle RBAC. Puis, nous présentons les mécanismes de contrôles d'accès décentralisés. Finalement, nous comparons ces travaux et nous positionnons notre contribution par rapport à l'existant.

2.4.1 Contrôle d'accès basé sur les informations contextuelles

Dans cette sous-section, nous analysons les principaux mécanismes et approches de contrôle d'accès pour les environnements collaboratifs dans lesquels les droits d'accès dépendent totalement des informations contextuelles.

Corradi et al. [CMT04] propose un framework, appelé UbiCOSM (*Ubiquitous Context-based Security Middleware*), qui considère les informations contextuelles lors de la spécification des politiques de sécurité et pendant l'exécution du processus de contrôle. UbiCOSM utilise un format basé sur RDF pour la représentation du contexte. Cependant la représentation du contexte est limitée aux relations en RDF et les auteurs n'utilisent pas des mécanismes d'inférence pour bien profiter de cette représentation. Lachmund et al. [LWG+06] proposent un framework qui permet d'assurer un contrôle d'accès basé sur les informations contextuelles. Ils définissent des mécanismes qui permettent d'assurer que les politiques de sécurité sont bien appliquées et qui permettent de générer des décisions d'autorisation en se basant sur une combinaison des informations contextuelles. Dans le framework proposé, les droits d'accès dépendent principalement des attributs de l'utilisateur qui demande une permission. Ce framework ne peut pas être utilisé dans des environnements dynamiques puisque les services de contrôle sont déployés dans des dispositifs mobiles qui constituent le réseau d'une manière statique. Groba et al. [GGS07] proposent une approche qui permet d'utiliser les informations contextuelles pour simplifier la gestion des mécanismes de contrôle d'accès pour les utilisateurs. Comme dans le travail précédent, les politiques ne considèrent que les informations contextuelles qui caractérisent le demandeur de la permission. Martin et Filho [FM09] considèrent, en plus des informations contextuelles du demandeur de la permission, le contexte des ressources et du possesseur de la ressource.

Wang et al. [WZC08] proposent un modèle de contrôle d'accès qui permet de protéger les services et les machines dans les environnements ubiquitaires. Ils considèrent le contexte des ressources durant le processus de

contrôle. Les auteurs considèrent les obligations et les conditions comme des facteurs importants lors de la génération des décisions d'autorisations. Dans l'approche proposée, l'environnement est divisé en plusieurs espaces physiques qui peuvent être représentés par des sessions de collaboration. Les droits d'accès sont affectés à un utilisateur dès qu'il rejoint un espace physique spécifique. Néanmoins, les éléments liés à la collaboration ne sont pas bien considérés lors de la mise en place des politiques et des services de contrôle.

Javanmardi et al. [JHJ06] proposent un framework de contrôle d'accès qui adresse des exigences bien particulières telles que l'intention des utilisateurs. Cependant certaines exigences des environnements ubiquitaires telles que la dynamicité et l'hétérogénéité des machines ne sont pas adressés.

Toninelli et al. [TMKL06] proposent un modèle de contrôle d'accès qui gère les droits d'accès en se basant sur plusieurs informations contextuelles telles que la disponibilité des ressources, la localisation, le temps d'accès, les activités de collaboration en cours, etc. Les auteurs adressent le problème de contrôle d'accès dans les réseaux ad-hoc où la collaboration peut être spontanée et imprévue. Des technologies du Web sémantique sont utilisées afin de faciliter la spécification du contexte de l'environnement. Des méthodes de raisonnement ont été aussi adoptées pour la définition et la spécification des politiques de contrôle d'accès.

Rashwand et Mišić [RM10] proposent un framework qui adresse l'hétérogénéité des dispositifs mobiles et la dynamicité de l'environnement en adaptant les politiques de sécurité en fonction des changements des informations contextuelles. Les auteurs classent les nœuds du réseau selon leurs capacités de traitement. Ils supposent que chaque partition du réseau possède un serveur puissant qui effectue le processus de contrôle. Néanmoins ce type de nœud n'est pas toujours disponible dans les environnements ubiquitaires. En outre, les auteurs n'utilisent pas un langage standard pour la spécification des politiques de contrôle d'accès. Dersingh et al. [DLJ08] essaient de surmonter les problèmes des approches antérieures dans lesquelles la spécification des droits d'accès se base sur les informations contextuelles. Ils proposent une approche qui se base sur des technologies de Web Sémantique pour la modélisation de la sémantique du contexte. Les modèles sont ensuite utilisés par un mécanisme de contrôle d'accès pour l'évaluation des politiques.

2.4.2 Mécanismes de contrôle d'accès qui étendent RBAC

Dans cette sous-section, nous analysons les mécanismes de contrôle d'accès qui se basent sur le modèle RBAC et en l'étendant pour tenir compte de certaines informations contextuelles.

Tripathi et al. [TAK+04b] définissent un modèle de contrôle d'accès pour les environnements sensibles au contexte. Le modèle proposé étend le modèle RBAC traditionnel avec des concepts tels que la description des ressources et les vues de contrôle d'accès. Les privilèges sont déterminés en se basant sur les rôles des utilisateurs. Ces privilèges sont appliqués en se basant sur des contraintes telles que le contexte physique de l'utilisateur. Strembeck et Neumanna [SN04] intègrent des contraintes contextuelles dans le modèle RBAC pour permettre à certaines politiques de se baser sur les

informations contextuelles lors de la prise des décisions d'autorisation. Les auteurs visent à profiter de tous les avantages du modèle RBAC tout en offrant des moyens pour la définition de politiques de contrôle d'accès qui se basent sur les informations contextuelles. Huang et al. [HWCL06] proposent une extension du modèle RBAC dans laquelle les règles de contrôle d'accès décrivent des contraintes avec des conditions en se basant sur le contexte. Lim et al. [LS07] proposent un système intelligent qui utilise un algorithme de réseaux de neurones qui étend le modèle RBAC avec des contraintes de contexte. Les permissions sont adaptées selon le contexte des utilisateurs. Kulkarni et Tripathi [KT08] proposent une autre extension du modèle RBAC dans laquelle les rôles et les permissions peuvent être affectés et révoqués selon le contexte. Dans [KT14], nous avons proposé une extension du modèle RBAC pour permettre la gestion des autorisations dans les environnements de travail collaboratifs où plusieurs membres appartenant à différentes organisations partagent différents types de ressources.

Plusieurs travaux proposent des extensions du modèle RBAC avec des contraintes temporelles et spatiales [WL04] [KB10] [JBLG05]. Wedde et al. [WL04] proposent une approche multi-niveau de contrôle d'accès dans laquelle les droits d'accès dépendent du rôle et de la localisation de l'utilisateur. Kirkpatrick et Bertino [KB10] proposent une architecture qui tient compte des contraintes spatiales dans les environnements qui appliquent le modèle RBAC. Dans la solution proposée, les politiques se basent sur la localisation des utilisateurs en plus de leurs rôles. Les auteurs ont aussi proposé un protocole basé sur GEO-RBAC [BCDP05]. Joshi et al. [JBLG05] proposent un modèle appelé GTRBAC qui fournit des mécanismes pour l'activation et la désactivation des rôles en se basant sur des contraintes temporelles. Plusieurs autres recherches ont été faites dans ce domaine [AC07] [CJ05] [RKY06]. Ces travaux de recherche sont généralement appliqués dans des scénarios de médecine et d'assistance santé dans lesquels les informations contextuelles sont importantes lors de la génération des décisions d'autorisation.

2.4.3 Contrôle d'accès décentralisé

Dans cette sous-section nous analysons les solutions qui proposent des techniques pour la décentralisation du contrôle d'accès dans les environnements ubiquitaires.

Zhou et al. [ZMR05] proposent un mécanisme pour combiner les politiques d'autorisation d'une manière sécurisée. Ces politiques peuvent être définies indépendamment et enregistrées dans des certificats appelés AC (*Attribute certificates*). Ces certificats peuvent être stockés dans un annuaire LDAP, dans un serveur HTTP ou dans un serveur de fichiers. Les auteurs supposent que les mécanismes de sécurité doivent être définis dans chaque domaine indépendamment. Dans chaque domaine, il y a une politique racine et quelques politiques subordonnées. La politique racine spécifie la manière d'utilisation des autres politiques du domaine et la méthode d'authentification pour pouvoir les utiliser. Ce mécanisme ressemble à la notion de "*target*" du standard XACML qui spécifie quelles politiques et règles doivent être utilisées pour répondre à une requête. Le modèle RBAC est utilisé avec des certificats X.509. Le cœur du système est un moteur de contrôle d'ac-

cès qui exécute les fonctions d'authentification et d'autorisation. Bien que les politiques de contrôle d'accès soient distribuées sur plusieurs domaines, l'utilisation d'un seul moteur de contrôle peut présenter des problèmes de passage à l'échelle avec un grand nombre de services et d'utilisateurs.

Quinn et al. [QKF+06] proposent un framework pour la gestion et la décentralisation des applications dans les environnements ubiquitaires. Dans le framework proposé, le contrôle d'accès ainsi que l'administration des politiques de contrôle d'accès sont décentralisés. Le modèle TBAC est utilisé dans un scénario de conférence afin de décrire les mécanismes de contrôle proposés. Les auteurs montrent que l'approche proposée permet de réduire le temps d'accès et les goulots d'étranglement quand le nombre de ressources augmente d'une manière exponentielle.

Mahalle et al. [MTPP13] proposent une approche floue (Fuzzy) pour le modèle TBAC. Ils utilisent la notion des niveaux de confiance pour la gestion d'identité. Le framework proposé, appelé FTBAC, définit trois niveaux : le niveau dispositif qui contient les dispositifs et la communication entre eux, le niveau requête qui calcule les valeurs de confiance, et le niveau contrôle d'accès qui établit les correspondances entre les valeurs de confiance et les permissions et génère les décisions. Chaque nœud du réseau calcule les valeurs de confiance et les droits d'accès des nœuds avec lesquels il communique.

Barolli et Xhafa [BX11] proposent une plateforme, appelé JXTA-Overlay, dont l'objectif est d'améliorer la capacité des technologies Java, JXTA et pair à pair afin de supporter les systèmes collaboratifs et distribués. Dans [AMMBX09], les auteurs proposent un framework de sécurité pour JXTA-Overlay. Dans ce framework, le contrôle d'accès est effectué par plusieurs agents dans le réseau. Ce framework est spécifique à JXTA-Overlay et les auteurs ne spécifient pas comment on peut l'utiliser dans d'autres systèmes collaboratifs.

D'autres travaux utilisent des techniques de délégation pour décentraliser le contrôle. Dans le mécanisme de délégation, un utilisateur délègue les propriétés de son rôle pour un autre utilisateur ou un autre rôle. Dans ce cas, le délégataire hérite les permissions associées au rôle du délégateur.

Liscano et al. [LW05] [LW07] proposent une approche qui ajoute des informations contextuelles au modèle dRBAC (*distributed Role-Based Access Control*) [FPP+02] qui assure le contrôle d'accès via le mécanisme de délégation afin de supporter les interactions ad-hoc dans les systèmes collaboratifs d'entreprise. L'architecture proposée permet de supporter la collaboration spontanée dans les environnements ubiquitaires. Claycomb et Dongwan [CS07] proposent aussi un framework pour la délégation des droits d'accès dans les environnements ubiquitaires.

Yao et al. [YTP05] proposent des techniques pour améliorer l'efficacité et la flexibilité de la délégation dans le modèle *Role Based Cascaded Delegation* (RBCD) [TYW04]. Ils utilisent les modèles de confiance décentralisées basées sur les rôles [TYW04] qui combinent la gestion de confiance avec RBAC. Dans ces modèles, les privilèges sont délégués aux rôles sans l'intervention d'un administrateur ce qui est important dans les situations d'urgence. D'après les auteurs, la délégation est efficace et permet d'assurer le passage à l'échelle. Ils supposent que le processus d'autorisation doit être décentralisé pour assurer la continuité de la communication et réduire les

coûts de traitement dans les environnements ubiquitaires où les machines peuvent avoir des capacités limitées.

Dans des situations complexes, le mécanisme de délégation peut devenir difficile à gérer surtout dans le cas de délégations multiples. Des problèmes de sécurité peuvent ainsi apparaître quand un utilisateur concerné par plusieurs délégations quitte les sessions de communication. Les architectures proposées supposent que des gestionnaires de délégation et d'autres serveurs sont déjà mis en place dans le réseau.

2.4.4 Positionnement

Le Tableau 2.3 présente une étude comparative des systèmes de contrôle dans les environnements ubiquitaires. Nous avons pris en compte les critères suivants :

- La décentralisation.
- Le type d'adaptation considéré : Ceci reflète la capacité du système à s'adapter aux changements.
- Le niveau de facilité d'utilisation du système dans les environnements de travail collaboratifs où les utilisateurs peuvent intervenir pour la spécification des politiques de contrôle d'accès.
- La possibilité de détecter des incohérences dans les politiques de contrôle d'accès.

Nous utilisons des terminologies de comparaisons tels que *élevé*, *bas*, *moyen* et les terminologies standard *oui* et *non* pour indiquer si le système est caractérisé ou non par un critère donné.

Types du système de contrôle d'accès	**Décentralisation**	**Adaptation (aspect dynamique)**	**Facilité d'utilisation dans les CWEs**	**Possibilité d'incohérence des politiques**
Centralisé et se base sur les informations contextuelles	non	Adaptation des politiques	faible	élevé
Centralisé et étend RBAC	non	Adaptation des politiques	faible	moyen
Décentralisés et utilise RBAC	oui	Adaptation des politiques (si avec délégation	faible	moyen
Décentralisé sans RBAC	oui	non	faible	élevé

Tab. 2.3 – *Comparatif des systèmes de contrôle d'accès dans les environnements ubiquitaires*

Les systèmes centralisés qui se basent sur les informations contextuelles (ligne 1 du tableau) proposent l'adaptation des politiques de contrôle d'ac-

cès. Selon ceux qui proposent ces systèmes, le modèle RBAC n'est pas assez flexible et ce n'est pas facile de l'utiliser dans les environnements ubiquitaires et il ne peut être utilisé que dans les systèmes dans lesquels les rôles et les permissions changent peu fréquemment au cours des processus de collaboration. A notre avis, il est difficile de mettre en place ces solutions dans les environnements de travail collaboratifs (CWEs) où plusieurs utilisateurs, qui collaborent ensemble peuvent intervenir pour la spécification des politiques de contrôle d'accès. La notion de rôle est importante pour la structuration de la collaboration dans ce type d'environnements. En conséquence, considérer la notion de rôle dans les mécanismes de contrôle d'accès est un besoin dans les environnements de travail collaboratifs. En outre, dans les systèmes qui se basent sur des informations contextuelles, des incohérences peuvent se présenter à cause de l'énorme quantité d'informations contextuelles utilisée dans les politiques de contrôle d'accès.

Les systèmes centralisés qui utilisent le modèle RBAC et l'étendent avec des informations contextuelles (ligne 2 du tableau) sont plus faciles à utiliser dans les environnements de travail collaboratif. En outre, ils permettent de réduire les incohérences dans les politiques de contrôle d'accès puisque ces politiques se basent principalement sur les rôles des utilisateurs. L'adaptation des politiques est le type d'adaptation retenu dans ces systèmes.

L'inconvénient principal de ces systèmes est que toutes les politiques sont disponibles pour une seule entité de contrôle, ce qui peut introduire des problèmes de passage à l'échelle avec l'augmentation du nombre d'utilisateurs, d'applications et de politiques de contrôle d'accès. Afin d'assurer le passage à l'échelle, plusieurs travaux (sous-section 2.4.3) proposent des solutions pour la décentralisation des systèmes de contrôle d'accès. Selon plusieurs auteurs [BDCdVS02] [CE03] [WL01], les travaux futurs de contrôle d'accès doivent considérer plusieurs entités de contrôle et chaque entité gère les autorisations indépendamment des autres. Cela offre la possibilité de définir indépendamment des domaines de sécurité qui sont libres à déterminer leurs propres politiques. Nous remarquons que parmi ces travaux, il existe certains qui se basent sur RBAC (Tableau 2.3 ligne 3) et d'autres qui utilisent des informations contextuelles autres que les rôles (Tableau 2.3 ligne 4). Selon le tableau, les travaux qui se basent sur RBAC sont, d'une part, plus faciles à utiliser dans les environnements de travail collaboratifs, et d'autre part, permettent de réduire les incohérences potentielles. Parmi les travaux de décentralisation qui se basent sur RBAC, il y a certains qui proposent des mécanismes de délégation. Ces travaux proposent l'adaptation des politiques en fonction des relations de délégation.

Selon le Tableau 2.3, le type d'adaptation le plus utilisé dans ces travaux est l'adaptation des politiques de contrôle d'accès suite aux changements de contexte. Les autres types d'adaptation ne sont pas considérés dans les systèmes de contrôle d'accès. Les travaux qui proposent la décentralisation du contrôle et qui se basent sur RBAC sont les plus similaires à notre travail. Ces travaux utilisent généralement des mécanismes de délégation qui peuvent introduire des problèmes de sécurité et des incohérences dans les environnements dynamiques où les utilisateurs rejoignent et quittent les sessions de collaboration fréquemment. En plus, même dans les travaux qui proposent la décentralisation des entités de contrôle, la mise en place de ces entités est statique et ne s'adapte pas aux évolutions des besoins de l'application.

C'est pour ces raisons que nous proposons une adaptation dynamique de l'architecture de contrôle d'accès. Cette adaptation architecturale consiste à adapter et déployer les entités de contrôle d'accès en fonction de la structure de collaboration et des machines des utilisateurs qui collaborent. Cette adaptation doit s'effectuer tout en assurant la décentralisation du contrôle d'accès. En outre, le déploiement des entités de contrôle d'accès dans les machines qui constituent le réseau doit être adaptatif et doit tenir compte des changements tels que le départ et l'arrivée des utilisateurs, le changement de rôle ou de groupe, etc. Notre approche se base par ailleurs sur le modèle RBAC afin de faciliter la spécification des politiques dans les CWEs et de réduire les incohérences des politiques de contrôle d'accès.

Conclusion

Notre travail se situe autour de trois principaux domaines : le contrôle d'accès, l'informatique ubiquitaire et les environnements de travail collaboratif. Dans notre approche, nous utilisons des techniques de représentation de connaissances et d'adaptation des applications en cours d'exécution tout en tenant compte des principales exigences de contrôle d'accès et de collaboration. La représentation sémantique des connaissances en utilisant les ontologies OWL assure une meilleure flexibilité du système. En effet, la collaboration peut être restructurée dynamiquement en définissant des hiérarchies de rôles et des sessions de collaboration à travers des fichiers OWL qui sont basés sur une syntaxe XML. L'avantage principal de l'utilisation des ontologies OWL est la possibilité de raisonnement qui permet d'inférer de nouvelles connaissances à travers l'application d'un ensemble de règles qui peuvent évoluer en fonction des besoins des applications. Dans ce chapitre, nous avons présenté les notions de base de ces domaines. Ensuite, nous avons étudié et analysé les travaux existants qui traitent des problématiques similaires à les nôtres et nous avons détaillé les avantages et les limites des différentes approches.

3 Framework pour l'adaptation des mécanismes de contrôle d'accès

Sommaire

Dans le chapitre 2, nous avons présenté un état de l'art des domaines auxquels nous nous intéressons. Dans le reste de ce travail de thèse, nous présentons nos apports et nous faisons recours aux langages et aux notions que nous avons introduits dans le chapitre 2. Ce chapitre présente notre première contribution : framework pour l'adaptation des mécanismes de contrôle d'accès. Ce framework permet d'adaptation les architectures logicielles de contrôle d'accès dans les environnements collaboratifs ubiquitaires. Trois parties principales constituent ce chapitre.

D'abord, nous présentons l'architecture de contrôle d'accès et la communication entre les différentes entités. Nous justifions les choix de décentralisation de cette architecture et de répartition des composants de contrôle d'accès.

Dans la deuxième partie de ce chapitre, nous présentons l'ontologie générique de contrôle d'accès (*Generic Access Control Ontology* ou GACO en anglais). Pour cela, nous commençons par introduire les technologies du Web Sémantique utilisées pour l'expression de GACO et les mécanismes de raisonnement avec OWL. Ensuite, nous détaillons les éléments de GACO et les règles qui y sont associées. Puis nous expliquons comment GACO peut être étendu pour pouvoir l'utiliser dans les applications collaboratives. A la fin de cette partie, nous présentons le problème de monotonicité d'OWL et la méthode que nous avons suivie afin d'y remédier.

Dans la troisième partie de ce chapitre, nous décrivons le framework en se basant sur la description de l'architecture décentralisée de contrôle d'accès et l'ontologie GACO. Nous présentons dans cette partie l'approche d'adaptation sur laquelle se base le processus d'adaptation du framework pour la reconfiguration de l'architecture de contrôle d'accès. Ensuite, nous présentons l'architecture du framework et les différentes entités qui le constituent. Enfin, nous présentons les détails d'implantation du framework à travers un diagramme de classe UML.

3.1 Architecture de contrôle d'accès

Dans cette section, nous proposons une architecture pour la décentralisation des mécanismes de contrôle d'accès dans les environnements collaboratifs ubiquitaires.

3.1.1 Décentralisation du contrôle d'accès

La gestion des politiques de sécurité devient de plus en plus complexe à cause de l'augmentation du nombre d'utilisateurs, d'applications et de contraintes de sécurité. Le passage à l'échelle dans le domaine de contrôle d'accès n'a pas été bien étudié dans la littérature et seulement les mécanismes de délégation entre les nœuds terminaux (principe *end to end*) ont été étudiés.

Dans les architectures centralisées, le contrôle d'accès est effectué par une seule entité centrale. Cette entité génère toutes les décisions d'autorisation pour toutes les requêtes d'accès dans le système. Vu que le nombre de requêtes peut augmenter en fonction du nombre d'utilisateurs et de ressources, l'entité qui gère le contrôle peut devenir un goulot d'étranglement et incapable de générer les décisions d'autorisation dans des délais raisonnables.

Les méthodes de réplication ont été parmi les premières solutions proposées pour adresser les problèmes de passage à l'échelle et elles sont toujours utilisées dans de nombreux systèmes décisionnels. Cependant, des problèmes de passage à l'échelle peuvent avoir lieu dans les mécanismes de contrôle d'accès même en utilisant les méthodes de réplication. Supposons par exemple que plusieurs entités logicielles de contrôle d'accès sont mises en place dans un système. Dans ce cas, la liste des politiques de contrôle d'accès doit être disponible pour chaque entité dans le réseau. Généralement, quand le nombre d'utilisateurs et de ressources (fichiers, documents, etc.) augmente, la taille de la liste des politiques augmente également ; et chaque entité doit parcourir toute la liste de politiques à chaque fois une décision d'autorisation est demandée. En conséquence, la charge au niveau des entités de contrôle d'accès peut augmenter rapidement en fonction du nombre de politiques malgré l'utilisation des méthodes de réplication. D'autres solutions doivent être mises en place en plus des méthodes de réplication afin d'assurer le passage à l'échelle.

Dans notre travail, nous proposons la mise en place de plusieurs entités de contrôle d'accès où chaque entité gère les autorisations pour un groupe d'utilisateurs qui ont des intérêts communs et qui partagent des ressources spécifiques. Dans ce cas, chaque entité gère l'accès à une partie des ressources et non à la totalité. Ce qui fait que chaque entité utilise un sous ensemble de politiques et ne tient pas compte obligatoirement de toutes les politiques dans le système. Cette approche a plusieurs avantages. D'une part, elle permet de réduire le nombre de requêtes d'accès à une ressource pour chaque entité vu que chaque entité gère l'accès pour un sous ensemble d'utilisateurs, et d'autre part, elle permet de réduire le nombre de politiques considérées par chaque entité. En suivant cette approche, chaque administrateur d'une session peut gérer les politiques indépendamment dans une partie de l'environnement collaboratif. Dans notre cas, nous supposons qu'un

ensemble de ressources est partagé dans chaque session et les administrateurs gèrent les politiques dans chaque session indépendamment.

3.1.2 Choix de l'architecture de contrôle d'accès

Dans cette sous-section, nous présentons comment nous pouvons utiliser les éléments du standard XACML pour la définition de l'architecture décentralisée de contrôle d'accès.

Selon le diagramme de flots de données du standard XACML présenté dans la Figure 2.4, la manière dont le PEP (*Policy Enforcement Point*) obtient une décision d'autorisation peut diminuer les performances du processus de prise de décision et surtout si nous considérons toutes les étapes de recherche et d'extraction d'attributs. Dans les implémentations 2.0 et 3.0 du standard XACML, le gestionnaire de contexte peut devenir un goulot d'étranglement avec l'augmentation du nombre de requêtes. Le gestionnaire de contexte a trois rôles principaux : (1) traduire la requête au format standard et l'envoyer au PDP (*Policy Decision Point*) ; (2) trouver les attributs et les rendre disponibles pour le PDP ; (3) traduire la décision au format natif du PEP. Afin de simplifier le processus de prise de décision, nous proposons d'intégrer le gestionnaire de contexte dans une autre entité qui peut être le PDP ou bien le PEP. Le choix de cette entité pour intégrer le gestionnaire de contexte dépend de la nature du système.

Dans le cas où le système ne tient pas compte des informations contextuelles lors de la prise de décision, le PDP ne demande pas des attributs supplémentaires au gestionnaire de contexte. Par conséquent, ce dernier peut être intégré dans le PEP. L'avantage de ce choix est que le PEP sera capable de traduire une requête au format natif à une requête au format standard de XACML et vice-versa. L'autre avantage de ce choix est que l'utilisation du PDP est standard vu qu'il reçoit une requête au format standard et renvoie la décision au format standard. Dans ce cas, le PDP est indépendant de tous les PEPs du système.

Si le système utilise les informations contextuelles lors de la prise de décision et ces dernières ne sont pas fournies dans les requêtes par le PEP, le PDP peut demander au gestionnaire de contexte ces informations. Dans ce cas, le gestionnaire de contexte peut être intégré dans le PDP. L'inconvénient de ce choix est que nous perdons le découplage entre les PEPs et le PDP. En effet, Le PDP devrait utiliser le format natif du PEP afin de communiquer avec ce dernier.

Puisque dans notre travail nous ne tenons compte que des rôles des utilisateurs lors de la définition des politiques de contrôle d'accès, nous intégrons le gestionnaire de contexte dans le PEP. Cette solution permet de simplifier le processus de prise de décision tout en assurant le découplage entre les PEPs et le PDP ainsi qu'une utilisation standard du PDP. La Figure 3.1 décrit les différentes étapes du processus de prise de décision :

1. Le PEP intercepte la requête de l'utilisateur.
2. Le PEP traduit la requête au format standard (*XACML Request*) et l'envoie au PDP.
3. Le PDP prend une décision d'autorisation en se basant sur les politiques et les algorithmes de combinaison de politiques.

4. Le PDP retourne la décision d'autorisation au format standard (*XACML Response*) au PEP.
5. Le PEP autorise ou interdit l'accès en se basant sur la décision du PDP.

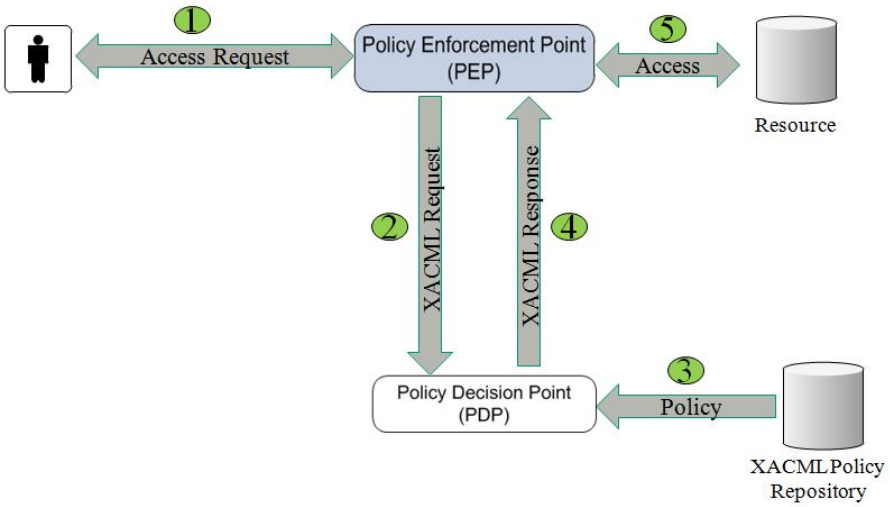

FIGURE 3.1 – *Processus de prise de décision*

Dans ce qui suit, nous présentons les architectures possibles pour la mise en place du mécanisme de contrôle d'accès en se basant sur le processus de prise de décisions détaillé dans la Figure 3.1. Notre objectif est de concevoir une architecture de contrôle d'accès qui permet le passage à l'échelle dans les environnements collaboratifs ubiquitaires. Le mécanisme de contrôle d'accès dans une telle architecture doit gérer un grand nombre d'utilisateurs, de ressources et de politiques de contrôle d'accès. Les architectures de contrôle d'accès que nous présenterons décrivent la manière de répartir des entités de contrôle d'accès (PEPs et PDPs) dans le réseau. Selon la répartition et le nombre de ces entités, nous identifions deux types d'architectures : une architecture centralisée (Figure 3.2) et une architecture décentralisée (Figure 3.3).

Dans l'architecture centralisée, représentée dans la Figure 3.2, un point d'application de politique (PEP) est associé à chaque utilisateur. Chaque PEP intercepte la requête de l'utilisateur et la transfère à un PDP central qui se charge de la prise des décisions d'autorisation. Selon la décision du PDP, le PEP autorise ou interdit l'accès à une ressource. Dans cette architecture, la spécification des politiques de contrôle d'accès est centralisée vu que toutes les politiques sont disponibles pour un seul PDP central. L'application des politiques est aussi centralisée vu que seulement un PDP est capable de retourner les décisions d'autorisation.

Dans l'architecture décentralisée, représentée dans la Figure 3.3, plusieurs PDPs sont mis en place. Chaque PDP contrôle l'accès pour un ensemble d'utilisateurs. La répartition et le nombre des PDPs peuvent dépendre de plusieurs facteurs. En effet, un PDP peut contrôler l'accès dans une zone géographique bien particulière qui regroupe un ensemble d'utilisateurs ou aussi contrôler l'accès de certains utilisateurs qui partagent des intérêts communs. Un ensemble de politiques est disponible pour chaque

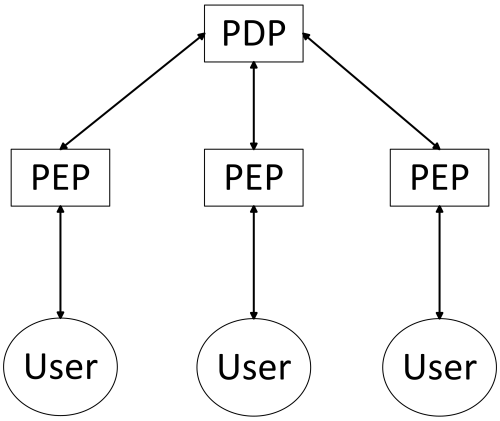

FIGURE 3.2 – *Architecture centralisée*

PDP. L'application des politiques est décentralisée vu que la prise des décisions d'autorisation est confiée à plusieurs PDPs. Dans notre travail, nous utilisons cette architecture décentralisée où un PDP est associé à chaque session de collaboration. Chaque PDP contrôle l'accès aux ressources partagées dans une session. Pour cela, un ensemble de politiques est défini dans chaque session par un administrateur. Puisque nous utilisons le modèle RBAC, ces politiques doivent spécifier les droits d'accès associés aux différents rôles impliqués dans les sessions.

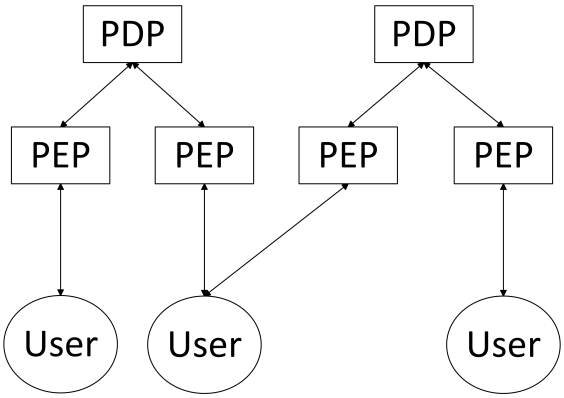

FIGURE 3.3 – *Architecture décentralisée*

3.2 Ontologie générique de contrôle d'accès (GACO)

Dans cette section, nous présentons l'ontologie générique de contrôle d'accès GACO. Cette ontologie permet principalement l'expression des situations de collaboration et des besoins de contrôle d'accès. Selon la classification proposée dans [GP99], GACO appartient à la classe d'ontologies génériques. En effet, elle peut être étendue pour être réutilisée dans différents domaines. L'objectif de l'utilisation de GACO est d'effectuer des inférences et des déductions à travers des mécanismes de raisonnement. Dans notre travail, le raisonnement permet l'identification des entités logicielles de contrôle d'accès (PDP et PEP) qui doivent être déployées afin d'assurer la sécurité des ressources. Dans le reste de cette section, nous décrivons les

langages utilisés pour l'expression de GACO et le raisonnement. Ensuite, nous détaillons les éléments de GACO et les règles de raisonnement qui y sont associées.

3.2.1 Ontologies OWL

Une ontologie peut définir deux types de vocabulaires [DLJo8]. Le premier type de vocabulaire est le nom des classes (ou concepts). Une classe représente un ensemble d'objets abstraits. Par exemple, une classe peut être une personne, un véhicule, etc. L'autre type de vocabulaire est le nom des propriétés qui relient deux classes. Ces deux types de vocabulaires peuvent être représentés avec le langage OWL [SWMo4]. La majorité des langages d'ontologies permettent de spécifier une taxonomie de classes. OWL permet la spécification des taxonomies de classes et des propriétés. Les ontologies OWL sont largement utilisées dans les systèmes informatiques qui se caractérisent par un niveau élevé d'autonomie et de flexibilité. Dans ce qui suit, nous présentons les notions de base des ontologies OWL.

Individu/instance

Un individu ou instance est un objet du domaine représenté par l'ontologie. OWL ne prend pas en compte l'hypothèse du nom unique (*Unique Name Assumption* ou UNA). Cela veut dire qu'un seul individu peut avoir deux noms différents. Par conséquent, le fait d'avoir deux noms d'individu différents n'implique pas que ces deux individus ne sont pas les mêmes. Pour cela, il faut déclarer explicitement si les individus sont différents ou les mêmes. La propriété *owl :sameAs* est utilisée pour déclarer que deux individus sont les mêmes. Tandis que la propriété *owl :differentFrom* est utilisée pour déclarer que deux individus sont différents.

Classe/concept

Une classe ou concept est un ensemble d'individus. OWL utilise une description formelle pour spécifier les conditions qui doivent être satisfaites par un individu pour qu'on puisse le considérer comme une instance d'une classe donnée. Les classes peuvent être organisées dans une hiérarchie (ou taxonomie). Une classe peut être une sous-classe d'une autre classe, appelée super-classe. Dans ce cas, tout individu appartenant à la sous-classe hérite les propriétés de la super-classe. Par exemple, si on considère la classe *Personne*, les sous classes d'une classe *Personne* dans une organisation peuvent être exprimées avec des rôles tels que *Etudiant*, *Professeur*, etc. Un des avantages du langage OWL-DL est qu'il permet aux moteurs d'inférence de tenir compte des relations d'héritage.

Propriété/relation

Une propriété ou relation est une relation binaire entre deux individus. OWL définit deux types de propriétés : propriétés d'objets et propriétés de type de donnée. Les propriétés d'objets modélisent le lien entre deux individus. Tandis que les propriétés de type de donnée modélisent le lien entre un individu et une valeur d'un type de donnée. Les types de données

sont équivalents à la notion d'attribut dans quelques formalismes tels que la programmation orientée objet. Dans OWL, une propriété peut être définie comme une sous-propriété d'une autre propriété. Pour chaque propriété, un domaine et une image (ou portée) doivent être définis. Le domaine d'une propriété est l'ensemble de classes à lesquelles appartiennent les individus qui sont à l'origine de la propriété. L'image d'une propriété est l'ensemble de classes ou de types de données qui représentent la destination de la propriété.

OWL définit des relations entre les propriétés telles que la relation inverse (*owl :inverseOf*). Si une propriété $P1$ est marquée comme étant l'inverse d'une autre propriété $P2$, alors pour tout individu x,y : $P1(x, y)$ si et seulement si $P2(y, x)$.

OWL définit un ensemble de caractéristiques pour les propriétés. En effet, une propriété peut être fonctionnelle, inverse fonctionnelle, symétrique ou transitive.

Une propriété P est dite fonctionnelle si pour tout individu x,y,z : $P(x, y)$ et $P(x, z)$ implique $y = z$. Cela veut dire que P ne peut avoir plus qu'une seule valeur y pour un individu x. Une propriété $P1$ est inverse fonctionnelle s'il ne peut pas y avoir deux instances $x1$ et $x2$ telles que $P1(x1, y)$ et $P1(x2, y)$.

Une propriété P est dite symétrique si pour tout individu x, y, si (x, y) est une instance de P, alors (y, x) est aussi une instance de P. Cela veut dire que si x est relié à y par une instance de P, alors y est relié à x par autre instance de la même propriété.

Une propriété P est dite transitive si pour tout individu x,y,z : si les deux couples (x, y) et (y, z) sont deux instances de P, alors on peut inférer que (x, z) est aussi une instance de P.

OWL permet la création d'autres types de classes plus complexes qui peuvent être construites en définissant des restrictions sur leurs propriétés. Elles peuvent également être construites en utilisant des combinaisons booléennes telles que l'union et l'intersection.

Restrictions

Les classes peuvent être décrites en définissant des restrictions sur les propriétés. OWL définit plusieurs types de restrictions telles que les restrictions universelles, existentielles, de cardinalité, de valeur, etc.

Les restrictions existentielles (*owl :someValuesFrom*) sont les plus utilisées dans les ontologies OWL. Ces restrictions obligent un ensemble d'individus d'une classe d'avoir au moins une relation (propriété) donnée avec un individu membre d'une autre classe. Un autre type de restrictions dites universelles (*owl :allValuesFrom*) obligent un ensemble d'individus d'une classe de n'avoir une relation donnée qu'avec des individus d'une classe spécifique. Les restrictions de cardinalité spécifient le nombre de valeurs que peut avoir un individu pour une relation donnée. Et finalement, les restrictions de valeur obligent les individus d'une classe d'avoir une valeur spécifique pour une propriété donnée.

Classes disjointes

Deux classes sont dites disjointes quand elles ne peuvent pas avoir des instances communes. Les classes disjointes sont spécifiées en utilisant la

propriété *owl :disjointWith*. Par exemple, les deux classes *Masculin* et *Feminin* peuvent être déclarées disjointes. Les classes disjointes ne sont pas des classes complémentaires. Si *Masculin* et *Feminin* sont déclarées complémentaires, alors le fait qu'un individu ne soit pas masculin permettra d'inférer qu'il est féminin.

Importation d'ontologies

Un des objectifs principaux du Web Sémantique est de permettre la réutilisation des ontologies. Une ontologie développée pour répondre à des besoins bien particuliers ne peut pas répondre aux besoins d'autres applications. OWL permet l'importation entre les ontologies. Si une ontologie importe une autre ontologie, elle hérite tous les éléments et les conditions sémantiques de l'ontologie importée. Par conséquent, la relation d'importation est transitive : Si une ontologie A importe une ontologie B, et B importe une ontologie C, alors A importe aussi C.

Généralement, l'importation se fait en indiquant l'URL de l'ontologie importée dans le fichier OWL de l'ontologie qui importe. Cela se fait en utilisant la propriété *owl :imports*. Ensuite, l'ontologie qui importe peut définir de nouveaux éléments tels que des classes, des propriétés, des règles, etc. Selon la classification définie dans 2.3.2, les ontologies de domaine importent généralement les ontologies de haut niveau et génériques en les étendant avec des éléments spécifiques.

3.2.2 Raisonnement avec les ontologies OWL

Inférence

Le fait que le langage OWL-DL se base sur la logique de description permet de déduire des connaissances implicites qui ne sont pas déclarées explicitement dans une ontologie. Ce type de raisonnement est appelé *inférence* [BCM+03]. Le processus d'inférence est effectué par des logiciels spécifiques appelés *moteurs d'inférence*.

Les principales tâches de raisonnement qu'un moteur d'inférence peut effectuer sont : la subsomption, la vérification de la satisfiabilité ou la cohérence, et la vérification des instances. La subsomption est la tâche principale des moteurs d'inférence. Elle permet de vérifier si une classe A est une sous-classe d'une autre classe B ou non. Un moteur d'inférence peut rendre explicites toutes les relations hiérarchiques qui ne sont pas explicitement déclarées. Un autre type d'inférence effectuée par les moteurs d'inférence est la vérification de la satisfiabilité qui permet d'indiquer si l'instanciation d'un concept entraîne l'apparition d'une contradiction. Cette tâche est appelée aussi vérification de la cohérence des concepts. La troisième tâche principale des moteurs d'inférence est la vérification des instances qui permet de vérifier si un individu donné appartient à une classe ou non.

Dans la logique de description, le raisonnement applique l'hypothèse du monde ouvert (*open-world assumption* ou OWA en anglais). En appliquant cette hypothèse, un fait qui n'est pas déclaré comme étant vrai est tout simplement inconnu. Cette hypothèse est l'opposé de l'hypothèse du monde clos (*closed-world assumption* ou CWA en anglais). En appliquant CWA, un

fait qui n'est pas déclaré comme étant vrai doit être faux. CWA est utilisé dans les systèmes qui disposent de toutes les informations nécessaires comme les systèmes de gestion de base de données. Cependant, OWA est utilisé dans les systèmes qui ne disposent pas de toutes les informations. C'est le cas des systèmes de représentation de connaissances où de nouvelles informations peuvent être découvertes. Cette hypothèse a une influence sur la manière de construire une ontologie. En effet, si on considère deux classes A et B, on ne peut pas considérer que les instances de A ne sont pas des instances de B à moins d'avoir déclaré que A et B sont disjointes.

Le raisonnement dans OWL-DL est caractérisé par l'aspect de monotonicité. Cela veut dire qu'on ne peut pas retirer des faits de l'ontologie. Si un moteur d'inférence a déduit qu'un fait f est vrai, il ne peut pas dans une prochaine inférence déduire que f est faux ; dans ce cas f devient à la fois vrai et faux. La monotonie se définit par le fait que l'ajout de nouveaux faits ne remet pas en cause les conclusions tirées précédemment.

Raisonnement avec les règles SWRL

Le langage SWRL (*Semantic Web Rule Langage*) [HPSB+04] est un langage de règle proposé par W3C. SWRL combine OWL-DL avec le langage RuleML (*Rule Markup Language*) [HPSB+04]. Il étend OWL-DL en incluant des clauses de Horn [Hor51]. Les règles SWRL ont la forme d'une implication entre un antécédent et un conséquent. Une règle SWRL a la forme suivante :

$$a_1 \wedge \ldots \wedge a_n \rightarrow b_1 \wedge \ldots \wedge b_n$$

Dans cette expression, $a_1 \wedge \ldots \wedge a_n$ est l'antécédent de la règle, et $b_1 \wedge \ldots \wedge b_n$ est appelé le conséquent. L'antécédent et le conséquent se composent de zéro ou plusieurs atomes. Un antécédent ayant zéro atome est considéré comme vrai (i.e., satisfait par n'importe quelle interprétation). Cependant, une conséquence vide est considérée comme fausse (i.e., n'est pas satisfaite par n'importe quelle interprétation). Un atome peut se présenter sous la forme d'un prédicat unaire, d'un prédicat binaire ou d'un prédicat n-aires. Généralement, un prédicat unaire est sous la forme $C(x)$ où C est un concept et x est une variable. Un prédicat binaire est sous la forme $P(x, y)$ où P est une propriété et x et y sont des variables. Un prédicat n-aires est représenté par des fonctions utilisées dans les règles SWRL, appelées *built-ins*. Les règles SWRL permettent d'augmenter l'expressivité des ontologies OWL en exprimant des relations qui ne peuvent pas être définies avec OWL seul.

3.2.3 Description de l'ontologie générique de contrôle d'accès GACO

Dans cette sous section, nous présentons GACO qui permet principalement d'établir une représentation sémantique des situations de collaboration. Le but essentiel de cette représentation est de modéliser et d'identifier les besoins de contrôle d'accès de haut niveau dans les sessions de collaboration. L'environnement collaboratif se compose d'un ou plusieurs groupes dans lesquels les utilisateurs collaborent dans des sessions de collaboration en fonction des rôles qui leurs sont attribués. Comme nous l'avons mentionné dans la section 3.1.2, nous nous basons sur la notion de session pour

assurer la décentralisation du mécanisme de contrôle d'accès dans les environnements collaboratifs.

La Figure 3.4 présente les principaux éléments de GACO. Les concepts sont représentés par des ellipses, les propriétés sont représentées par des flèches allant d'un concept vers un autre et les types de données sont représentés avec des lignes hachurées.

Les recommandations de nomenclature des éléments des ontologies ont été résumées par Théreaux [Theo3]. Les principales recommandations sont :

1. Les URIs doivent être les plus courts possible
2. Utiliser les politiques de nommage (tout en minuscule ou première lettre en majuscule)
3. Utiliser la forme singulière
4. Ne pas utiliser les espaces

Pour le nommage des éléments de l'ontologie, nous respectons ces recommandations et nous suivons les principes suivants :

1. Les noms des concepts commencent par une majuscule.
2. Les noms des relations sont en minuscules.
3. Les types de données commencent par une majuscule.
4. Si le nom d'un élément se compose de plusieurs mots, ils sont collés et chacun d'eux commence par une majuscule.

Dans ce qui suit, nous décrivons les concepts et les relations de GACO.

Les différents nœuds qui participent aux activités collaboratives sont représentés par le concept *Node*. Un nœud peut représenter un être humain ou une entité logicielle autonome. Nous considérons comme un nœud toute entité qui peut collaborer et accéder à des ressources. Un nœud doit utiliser ou être hébergé dans un dispositif qui fournit une infrastructure matérielle qui assure la communication et l'accès aux ressources. Ces dispositifs sont modélisés par le concept *AccessPoint*. La propriété uses relie le concept *Node* au concept *AccessPoint*. Cette propriété est déclarée fonctionnelle puisqu'un nœud ne peut pas utiliser plus qu'un dispositif à un instant t. Sa propriété inverse, appelée *usedByNode*, permet d'identifier le nœud qui utilise un dispositif donné à un moment donné. Une propriété de type de donnée, appelée *hasId*, est associée à chaque *AccessPoint*. Cette propriété permet d'identifier un *AccessPoint* par une chaîne de caractère de type String qui représente généralement son adresse IP. Cette adresse IP est utilisée tout au long du processus d'adaptation que nous décrirons dans la section suivante.

Un nœud peut jouer un ou plusieurs rôles qui représentent ses fonctions ou ses responsabilités dans un groupe. Les rôles sont modélisés par le concept *Role*. L'attribution d'un rôle à un nœud est modélisée via la propriété non fonctionnelle *hasRole*. La notion de rôle est fondamentale dans les mécanismes de contrôle d'accès qui se base sur le modèle RBAC. En conséquence, tout changement au niveau de l'attribution des rôles aux utilisateurs peut influer sur leurs droits d'accès.

Ayant un ou plusieurs rôles, un nœud peut appartenir à un ou plusieurs groupes représentés par le concept *Group*. Un nœud donné peut jouer un rôle $r1$ dans un groupe $g1$ et un autre rôle $r2$ dans un groupe $g2$. La propriété

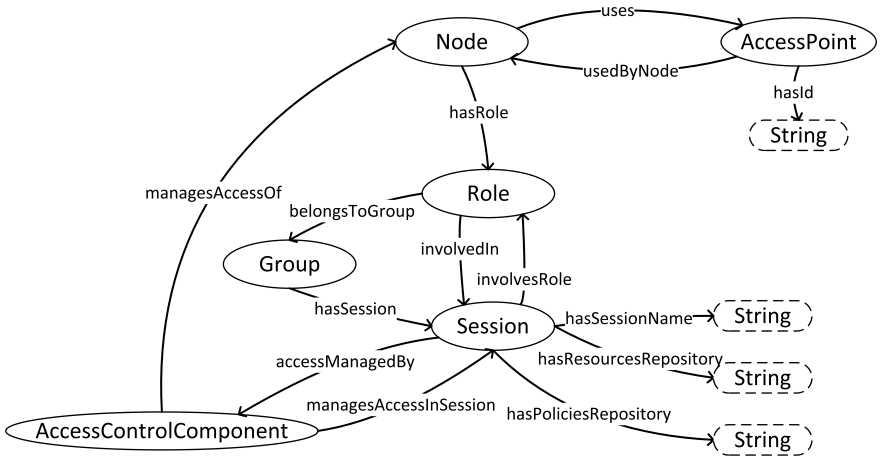

FIGURE 3.4 – *Principaux éléments de GACO*

belongsToGroup, qui relie le concept *Role* au concept *Group*, exprime l'appartenance d'un rôle à un groupe. Cette propriété est non fonctionnelle car un rôle peut appartenir à un ou plusieurs groupes. La propriété *hasMember* est la relation inverse de la relation *belongsToGroup*. Dans chaque groupe, des sessions de collaboration sont établies afin d'assurer la collaboration entre les nœuds. Les sessions, représentées par le concept *Session*, représentent un ensemble de nœuds qui partagent des intérêts communs en partageant un ensemble de ressources. La propriété de type de donnée *hasSessionName* permet d'associer à chaque session une chaîne de caractère qui représente son nom. L'association des sessions à chaque groupe est modélisée par la propriété *hasSession* qui relie le concept *Group* au concept *Session*. Cette propriété est non fonctionnelle car plusieurs sessions peuvent être établies dans un même groupe. Un nœud peut collaborer dans une ou plusieurs sessions en fonction de son rôle. Cela est modélisé par la propriété non fonctionnelle *involvedIn* qui relie le concept *Role* au concept *Session*. La propriété inverse *involvesRole* permet de désigner l'ensemble des rôles qui sont impliqués dans l'activité collaborative de la session.

Le contrôle d'accès est effectué dans chacune des sessions indépendamment. Pour cela un ensemble de politiques de contrôle d'accès doivent être définies pour chaque session. Ces politiques spécifient les droits d'accès des rôles qui collaborent et partagent des ressources dans la session. Deux propriétés de type de donnée sont définies : la propriété *hasResourcesRepository* permet de spécifier l'URL du répertoire qui héberge les ressources partagées dans la session, tandis que la propriété *hasPoliciesRepository* indique l'URL du répertoire qui héberge les politiques de contrôle d'accès associées à la session.

Les concepts et les propriétés que nous avons détaillés permettent de décrire l'environnement de collaboration à un moment donné. Afin de déterminer les besoins de contrôle d'accès, nous définissons le concept *AccessControlComponent*. Ce concept représente une entité qui doit être mise en place afin de contrôler l'accès d'un nœud aux ressources d'une session donnée. Cela est modélisé par les propriétés fonctionnelles *managesAccessOf* et *managesAccessInsession* : la propriété *managesAccessOf* relie le concept *AccessControlComponent* au concept *Node*. Cette propriété indique le besoin de

contrôler l'accès aux ressources d'un nœud donné. La propriété *managesAccessInSession* indique la session dans laquelle l'entité *AccessControlComponent* doit contrôler l'accès. Puisque ces deux propriétés sont fonctionnelles, une entité de contrôle d'accès est associée à un seul nœud et à une seule session de collaboration. Cette entité traduit le besoin de contrôle d'accès d'un nœud donné dans une session bien déterminée. Cela est pris en compte par le système afin de mettre en place les entités logicielles nécessaires pour satisfaire ce besoin. La propriété *accessManagedBy* est la propriété inverse de *managesAccessInsession*.

GACO ne regroupe qu'un petit nombre de concepts et de relations. Ceci a plusieurs avantages. D'une part, ça simplifie la compréhension et l'utilisation de GACO par les concepteurs des applications. D'autre part, le traitement d'un petit nombre d'éléments rend les moteurs d'inférence et de raisonnement plus performants.

3.2.4 Raisonnement avec GACO

Une instance de GACO permet de représenter les différentes entités qui constituent l'environnement collaboratif. Cette instance regroupe un ensemble d'individus reliés avec des instances des propriétés décrites dans GACO. Afin de mieux profiter des ontologies OWL et des capacités de raisonnement, nous définissons un ensemble de règles qui seront appliquées à des instances du modèle. Ces règles permettent d'inférer de nouvelles connaissances. Dans notre approche, les règles permettent de déterminer les besoins de contrôle d'accès selon les situations de collaboration. Nous définissons deux types de règles : (1) des règles génériques qui sont appliquées indépendamment des besoins applicatifs, et (2) des règles spécifiques à un domaine particulier. Dans ce qui suit, nous présentons la principale règle générique associée avec GACO.

Cette règle permet d'inférer les entités de contrôle d'accès qui doivent être mises en place afin de sécuriser l'accès aux ressources dans les différentes sessions de collaboration. L'utilisation d'OWL seul ne permet pas d'effectuer certaines inférences complexes qui ne sont possibles qu'avec des règles SWRL. Par exemple, l'instanciation des concepts de l'ontologie n'est pas possible avec OWL seul.

La règle *access_control_components_generation*, détaillée dans la Figure 3.5, permet de créer les composants nécessaires qui contrôlent l'accès des nœuds aux ressources partagées dans les sessions. En effet, pour chaque individu n de la classe *Node* qui joue un rôle r dans une session s, cette règle crée une instance de la classe *AccessControlComponent* qui gère l'accès aux ressources du nœud n dans la session s. Le *built-in swrlx :createOWLThing* est utilisé pour la création des instances des composants. Les arguments du *built-in* sont le nœud et la session. Le choix de ces arguments est justifié comme suit : pour chaque nœud qui appartient à une session, nous créons un seul individu de la classe *AccessControlComponent* qui gère les accès du nœud dans la session. Cet individu sera instancié pour chaque couple (s,n) où n est un nœud qui joue un rôle dans la session s. L'appartenance d'un rôle r à une session s est déterminée par la propriété $involvedIn(?r, ?s)$. Ces propriétés sont définies par les ontologies de domaine qui étendent GACO afin de spécifier les situations de collaboration entre les rôles.

$$\text{Node}(?n) \wedge \text{Role}(?r) \wedge \text{hasRole}(?n, ?r) \wedge \text{involvedIn}(?r, ?s) \wedge \text{Session}(?s) \wedge \text{swrlx:createOWLThing}(?acc, ?n, ?s) \rightarrow \text{AccessControlComponent}(?acc) \wedge \text{managesAccessOf}(?acc, ?n) \wedge \text{managesAccessInSession}(?acc, ?s)$$

FIGURE 3.5 – *Règle access_control_components_generation*

Exemple d'application de la règle

Considérons une instance de GACO illustrée dans la Figure 3.6. Cette instance contient deux individus *node*1 et *node*2 de la classe *Node*. Ces deux nœuds jouent respectivement les rôles *role*1 et *role*2. Ils appartiennent au même groupe *group*1 et collaborent dans une session *session*1.

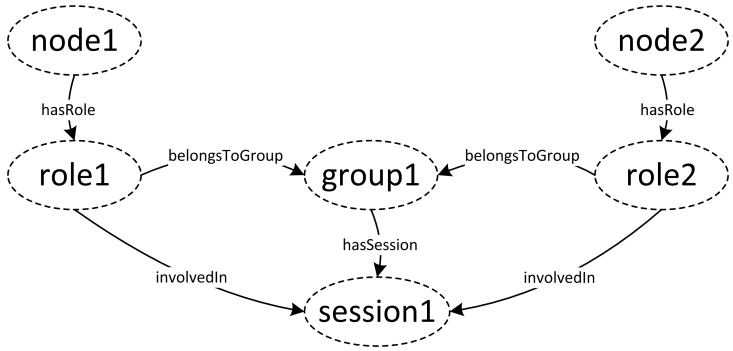

FIGURE 3.6 – *Exemple d'application de la règle : état initial*

L'application de la règle déduit une nouvelle instance de GACO illustrée dans la Figure 3.7. Cette règle a créé les deux individus *acc*1 et *acc*2 de la classe *AccessControlComponent*. L'individu *acc*1 indique le besoin de contrôler les accès du nœud *node*1 aux ressources partagées dans la session *session*1, tandis que l'individu *acc*2 indique le besoin de contrôler les accès du nœud *node*2 aux ressources de la même session.

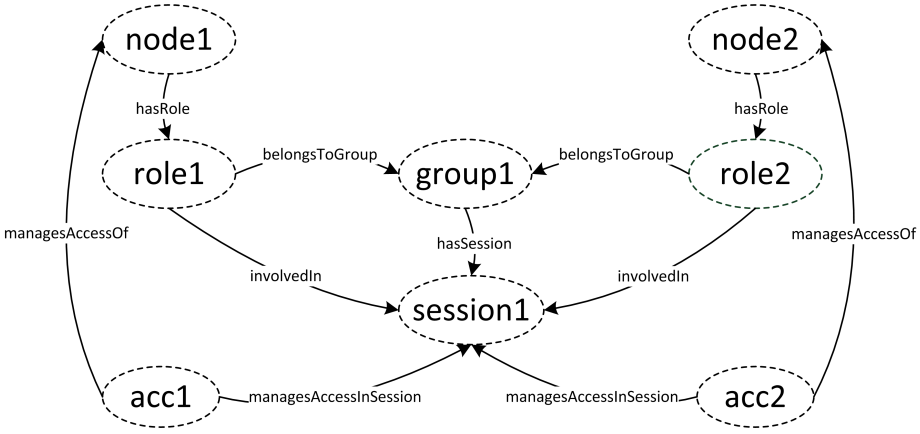

FIGURE 3.7 – *Exemple d'application de la règle : état final*

3.2.5 Extension de GACO

Extensibilité de GACO

Nous avons conçu GACO de telle sorte que ses éléments soient les plus génériques possible. Les concepts et les éléments présents dans GACO sont indépendants des domaines. Cet aspect de généricité permet la réutilisation de GACO pour modéliser les besoins de contrôle d'accès dans différents environnements collaboratifs.

Afin de garder l'aspect de généricité, nous n'avons pas inclus des individus dans GACO. Les individus seront créés dans des instances de GACO afin d'exprimer les situations de collaboration et les besoins de contrôle d'accès.

Une application qui utilise GACO doit l'étendre en définissant les éléments spécifiques à un domaine bien particulier. La définition de ces éléments se fait à travers la création d'une deuxième ontologie qui étend les concepts et les relations de GACO. Selon la classification fournie dans la section 2.3.2, cette ontologie est appelée ontologie de domaine. Ce mécanisme est possible avec OWL grâce au mécanisme d'importation d'ontologies. Une ontologie qui importe GACO a un accès en lecture seule à tous les concepts et les relations et elle peut définir des sous-concepts et des sous-relations spécifiques au domaine. Par exemple, dans une application d'ingénierie, l'ontologie de domaine peut définir les concepts *Développeur* et *Concepteur* comme des sous concepts du concept *Role* de GACO. L'extension de GACO permet d'identifier les besoins de contrôle d'accès dans des situations de collaboration complexes impliquant différents groupes, rôles, sessions, etc.

Processus de création de l'ontologie de domaine

La Figure 3.8 présente les principales étapes à suivre pour la création de l'ontologie de domaine. Nous décrivons les différentes étapes comme suit :

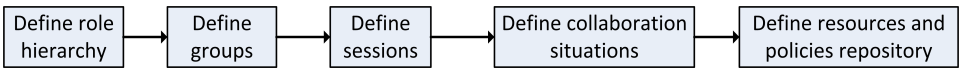

FIGURE 3.8 – *Étapes de création de l'ontologie de domaine*

Étape 1 : Définition de la hiérarchie de rôles
Cette étape consiste à définir tous les rôles qui peuvent participer à l'activité collaborative. Tous ces rôles sont des sous concepts du concept *Role* de GACO et peuvent être organisés dans une hiérarchie. Ces rôles doivent être définis comme disjoints afin de faciliter l'exécution des règles. En effet, au cours du raisonnement, un individu d'un rôle $R1$ ne peut pas être considéré comme une instance d'un autre rôle $R2$.

Étape 2 : Définition des groupes
Cette étape consiste à définir la liste des groupes de l'activité collaborative. Un groupe représente un ensemble d'utilisateurs qui ont des objectifs communs. Plusieurs sessions peuvent être établies dans un même groupe afin de permettre aux utilisateurs de collaborer et de partager des ressources. Les groupes sont définis comme des instances du concept *Group*.

Étape 3 : Définition des sessions
Cette étape consiste à définir les sessions de collaboration de chaque groupe. Ces sessions sont définies comme des instances du concept *Session* de GACO. Un nom doit être attribué pour chaque session en utilisant la propriété de type de donnée *hasSessionName*.

Étape 4 : Définition des situations de collaboration
Cette étape consiste à spécifier les rôles qui sont impliqués dans l'activité collaborative des sessions de collaboration. Cette étape permet d'indiquer, dans chaque session, les rôles qui peuvent collaborer et partager des ressources. Cela est compliqué à effectuer avec OWL seul car nous devons attribuer chaque instance de rôle à une session donnée. Pour cette raison, nous utilisons des règles SWRL. Un exemple de règle est illustré dans la Figure 3.9.

gaco :Node(?nd) ∧ Designer(?rd) ∧ gaco :hasRole(?nd, ?rd) ∧
gaco :Node(?ni) ∧ Integrator(?ri) ∧ gaco :hasRole(?ni, ?ri) ∧
gaco :belongsToGroup(?rd, ?g) ∧ gaco :belongsToGroup(?ri, ?g) ∧
differentFrom(?ri, ?rd) ∧ gaco :Session(?s) ∧ gaco :hasSessionName(?s,
"design_integrator_s") ∧ gaco :hasSession(?g, ?s) →
gaco :involvedIn(?ri, ?s) ∧ gaco :involvedIn(?rd, ?s)

FIGURE 3.9 – *Exemple de règle pour la définition des situations de collaboration*

Dans cet exemple, nous supposons que l'ontologie de domaine contient (1) les rôles *Designer* et *Integrator* qui sont définis comme des sous-concepts du concept *Role*, et (2) une session ayant le nom *design_integrator_s* qui est également définie comme une instance du concept *Session*. Cette règle indique que pour tout individu *nd* et *ni* instances du concept *Node*, tel que :
- *nd* est *ni* jouent respectivement les rôles *Designer* et *Integrator*,
- *nd* et *ni* appartiennent au même groupe *g*,
- la session *design_integrator_s* est associée au groupe *g*,

nous pouvons déduire que *nd* et *ni* sont impliqués dans la session *design_integrator_s*.

Étape 5 : Définition des URLs des ressources et des politiques
Cette étape consiste à définir deux URLs. Un URL qui indique l'emplacement du répertoire qui héberge les ressources en utilisant la propriété de type de données *hasResourcesRepository*, et un autre URL qui indique l'emplacement des politiques de contrôle d'accès en utilisant la propriété *hasPoliciesRepository*. Ce dernier sera utilisé par l'entité qui prend les décisions d'autorisation.

3.2.6 Problème de monotonicité d'OWL

Une instance de GACO représente l'état de la collaboration à un moment donné. Au cours du temps et en fonction des changements des situations, plusieurs instances peuvent être créées et modifiées pour représenter l'état actuel de la collaboration. Dans les systèmes adaptatifs, ces instances sont généralement traitées par des moteurs d'inférence afin de déduire de nouvelles connaissances. Les connaissances déduites à un instant t doivent rem-

placer les connaissances qui ont été inférées à un instant $t-1$. Par exemple, si à un instant t, une entité *AccessControlComponent* a été créée pour contrôler l'accès d'un utilisateur dans une session donnée, et si à un instant $t+1$, l'utilisateur quitte cette session et rejoint une autre, l'entité qui est déjà créée doit être remplacée par une autre qui contrôle l'accès du même utilisateur dans la nouvelle session. Malheureusement, ceci n'est pas possible avec les règles SWRL à cause de la nature monotone d'OWL/RDF. En effet, la nature monotone d'OWL implique qu'une information présente dans une instance d'une ontologie ne peut pas être retirée ou modifiée par le traitement des règles et il n'est pas possible d'écrire une règle qui permet de retirer des informations.

La seule méthode qui permet de retirer des informations dans une instance est l'utilisation des APIs OWL. Ces APIs offrent des méthodes pour manipuler (supprimer, ajouter, modifier) des individus dans une instance d'ontologie. Cependant, l'utilisation des APIs pour retirer plusieurs informations fréquemment est coûteuse et peut diminuer les performances du système. En effet, des algorithmes complexes doivent être mis en place afin de pouvoir décider ce qui doit être retiré ou non au cours de chaque mise à jour de l'instance.

La plupart des travaux [STV10] qui utilisent des règles de raisonnement avec les ontologies OWL adoptent la solution suivante : à chaque fois qu'il y a des changements dans l'environnement, une nouvelle instance de l'ontologie est créée afin de capturer l'état du monde ; ensuite, cette instance sera traitée par les règles associées à l'ontologie afin d'effectuer des raisonnements ; le résultat est une nouvelle instance de l'ontologie qui contient de nouveaux individus. Le problème de cette solution est que le résultat du raisonnement à un instant t écrase le résultat du raisonnement précédent à un instant $t-1$. Ceci peut influer sur les performances des systèmes qui utilisent le raisonnement avec les ontologies OWL. Le résultat de l'instant $t-1$ aurait pu être mis à jour en définissant des règles pour l'adaptation des instances si OWL supportait des raisonnements non monotoniques.

Nous utilisons cette méthode et nous discuterons son effet sur notre solution. Dans le chapitre 4, nous proposons des algorithmes afin de combler le désavantage de la nature monotone d'OWL.

3.3 Description du Framework

Dans cette sous-section, nous présentons l'approche d'adaptation qui se base sur la boucle de traitement autonomique. Ensuite, nous détaillons notre framework pour l'adaptation des architectures logicielles de contrôle d'accès dans les environnements collaboratifs ubiquitaires. Pour cela, nous décrivons l'architecture et les détails d'implantation. Pour assurer l'adaptation nous nous basons sur une approche dirigée par les modèles. Nous détaillons les modèles utilisés dans notre framework ainsi que les transformations de ces modèles au cours du processus d'adaptation.

3.3.1 Approche d'adaptation

Dans cette section, nous présentons l'approche d'adaptation que nous avons suivie pour l'implantation de notre framework de contrôle d'accès.

Cette approche est représentée par le modèle de la boucle autonomique [KC03] qui a été proposé par IBM. Ce modèle, illustré dans la Figure 3.10, est souvent appelé MAPE-K (*Monitor*, *Analyze*, *Plan*, *Execute*, *Knowledge*). Il est de plus en plus utilisé pour décrire le comportement des systèmes autonomiques que nous avons décrits dans la sous-section 2.1.5. Un système autonomique se compose d'un ou plusieurs gestionnaires autonomiques (ou *Autonomic managers* en anglais) qui interagissent avec des ressources matérielles et logicielles que l'on appelle éléments gérés (ou *Managed element* en anglais). Dans les systèmes classiques non autonomiques, on trouve simplement des éléments gérés. Cependant ces éléments peuvent être adaptés en ajoutant des gestionnaires autonomiques qui les observent et les contrôlent. Un gestionnaire autonomique couplé avec un ou plusieurs éléments gérés forme un élément autonomique qui est responsable de la gestion de son comportement interne et de ses interactions avec l'environnement.

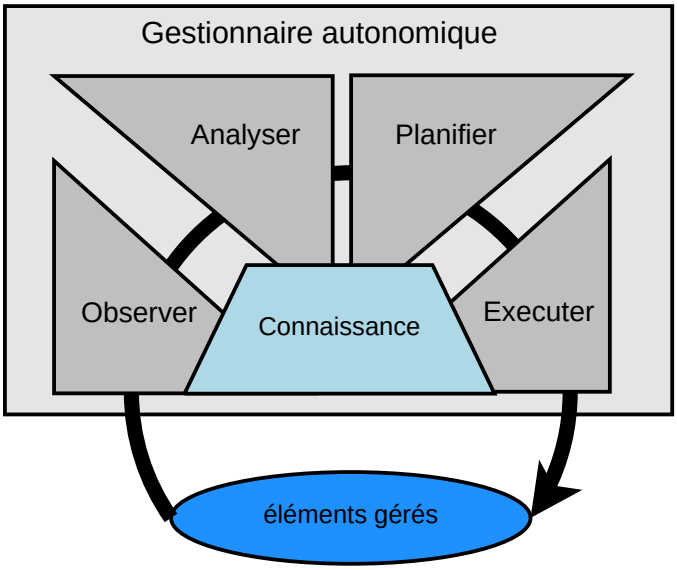

FIGURE 3.10 – *Structure d'un élément autonomique*

Le gestionnaire autonomique est une entité logicielle configurée généralement par des administrateurs humains en définissant des politiques et des stratégies qui décrivent les comportements du système. Les systèmes autonomiques utilisent généralement des capteurs qui fournissent des informations sur l'état de l'environnement. Ces informations sont utilisées ensuite par les gestionnaires autonomiques afin de planifier et exécuter de actions en se basant sur les politiques définies. Le processus d'adaptation des gestionnaires autonomiques est composé de quatre principales fonctions : l'observation, l'analyse, la planification et l'exécution.

Dans la boucle de contrôle autonomique, une base de connaissance est partagée entre les différentes fonctions. elle peut contenir différent types de données telles que des typologies d'information, des métriques, des politiques, etc. Elle peut représenter un historique des états et du comportement du système. Les quatre fonctions de la boucle ont la possibilité de la consulter ou/et y rajouter des informations.

Les fonctions du processus d'adaptation peuvent être décrites comme suit :

Fonction d'observation

La fonction d'observation (ou *monitoring* en anglais) permet de capturer les détails et les propriétés des ressources de l'environnement. Pour désigner l'action d'observer, on utilise par abus de langage le verbe "monitorer" issu du mot anglais "*monitoring*". Cette fonction filtre toutes informations captées jusqu'à trouver un symptôme qui doit être analysé.

Fonction d'analyse

Les données d'observation sont ensuite traitées par la fonction d'analyse de la boucle de contrôle autonomique MAPE-K. Cette fonction permet de déterminer les moments où des changements doivent être effectués. Elle permet aussi d'identifier les sources de transition vers un nouvel état du système. Si des changements sont requis, une requête est automatiquement envoyée à la fonction de planification.

Fonction de planification

La fonction de planification produit et structure un ensemble d'actions à effectuer sur les éléments gérés. Une action peut être une commande simple ou un ensemble de commandes complexe. Dans le cas où plusieurs commandes complexes doivent être effectuées, la fonction de planification doit planifier l'exécution de ces actions afin d'éviter les incohérences. Lors de cette phase, les actions déterminent quelles parties du système doivent être changées et comment.

Fonction d'exécution

Cette fonction se charge d'appliquer les actions déterminées par la fonction de planification tout en respectant leur ordre d'exécution. Ces actions peuvent être appliquées sur une ou plusieurs machines. Ceci permet de changer le comportement des ressources gérées afin de s'adapter aux changements de l'état du système.

3.3.2 Application de l'approche et architecture du framework

Le modèle de la boucle de contrôle MAPE-K est générique et peut être instancié pour détailler les architectures de plusieurs systèmes autonomiques. Dans cette section, nous décrivons comment l'utilisons pour l'implantation de notre framework. Pour cela, nous détaillons le fonctionnement du processus d'adaptation en passant par les phases d'observation, d'analyse, de planification et d'exécution. Nous présentons les interactions de ces quatre fonctions avec les différents modèles qui constituent la base de connaissance du framework. Chaque fonction du processus d'adaptation traite une instance d'un modèle et retourne une ou plusieurs instances en fonction des besoins du système. Notre but est de mettre en œuvre une adaptation dynamique de l'architecture de contrôle d'accès afin de sécuriser l'accès aux ressources dans les différentes sessions de l'environnement collaboratif. Le framework que nous présenterons est générique et peut être utilisé pour sécuriser l'accès aux ressources dans différents domaines.

Lors de la conception du framework, nous avons associé un ensemble d'entités pour chaque fonction de la boucle de contrôle autonomique. La Figure 3.11 illustre l'architecture du framework et ses différentes entités qui permettent d'effectuer les processus d'adaptation. Chaque processus d'adaptation assure le déploiement des composants logiciels qui permettent de sécuriser l'accès aux ressources. Les quatre principales entités du framework sont l'observateur de la collaboration (*Collaboration Monitor*), l'analyseur (*Analyzer*), le planificateur de contrôle d'accès (*Access Control Planner*) et le gestionnaire de déploiement (*Deployment Manager*). Les données capturées ainsi que les politiques d'adaptation sont modélisées par l'ontologie de domaine (*Domain Access Control Ontology*) qui étend GACO. Nous décrivons dans ce qui suit l'objectif de la mise en place de chaque entité du framework.

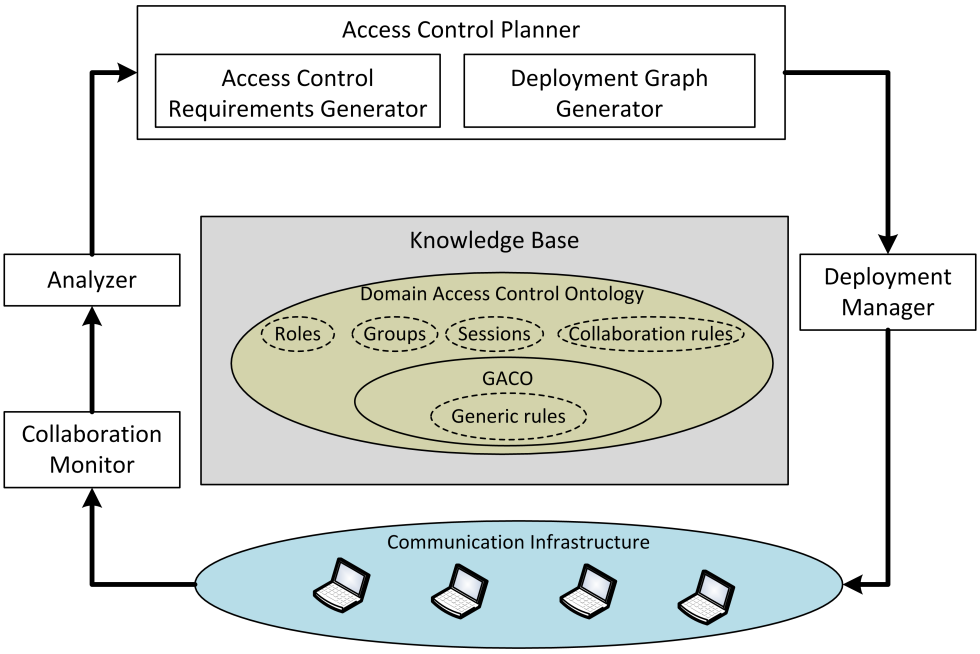

FIGURE 3.11 – *Architecture du framework*

L'observateur de la collaboration - Collaboration Monitor

Cette entité permet de capturer les actions des utilisateurs au cours des activités collaboratives. Ces actions décrivent toutes informations relatives aux utilisateurs, aux rôles et aux groupes. Elles peuvent représenter la connexion d'un utilisateur avec un rôle donné, l'appartenance d'un utilisateur à un groupe, la déconnexion d'un utilisateur, etc. Cette entité peut faire partie du serveur de l'application qui utilise le framework. Les actions capturées par cette entité sont ensuite transmises à l'analyseur (*Analyzer*).

L'analyseur - Analyzer

Cette entité permet de détecter les changements dans l'environnement qui nécessitent une adaptation. Ces changements représentent toute action qui a un effet sur la structure des sessions de collaboration. Les paramètres qui peuvent déclencher un processus d'adaptation sont : le nombre de ses-

sions, le nombre d'utilisateurs qui collaborent dans chaque session et les rôles impliqués dans chaque session. Afin de fournir une vue des entités qui collaborent à un moment donné, l'analyseur se base sur un modèle spécifique au domaine de l'application considérée. Ce modèle est représenté par l'ontologie de domaine (*Domain Access Control Ontology*) qui étend GACO avec des rôles des entités qui interviennent dans la collaboration et leurs organisations dans les différentes sessions.

Le planificateur de contrôle d'accès - Access Control Planner

L'objectif de cette entité est de générer un graphe de déploiement qui répond aux besoins de contrôle d'accès à un moment donné. Ce graphe décrit les composants de contrôle d'accès qui doivent être déployés dans chaque machine ainsi que la manière dont ils sont connectés. Afin de pouvoir générer les graphes de déploiement, le planificateur de contrôle d'accès utilise deux entités : le générateur des besoins de contrôle d'accès (*Access Control Requirements Generator*) et le générateur des graphes de déploiement (*Deployment Graph Generator*).

Le générateur des besoins de contrôle d'accès fournit un schéma qui traduit les besoins de contrôle d'accès dans chaque session de collaboration. Les éléments qui sont représentés dans ce schéma sont les sessions, les composants de contrôle d'accès et les nœuds. Ces éléments peuvent être implantés avec différentes technologies et en suivant divers standards de contrôle d'accès. L'implantation réelle de ses éléments est décrite au niveau du générateur des graphes de déploiement.

Le générateur des graphes de déploiement fournit un schéma de déploiement des composants de contrôle d'accès en se basant sur le schéma de haut niveau fourni par le générateur des besoins de contrôle d'accès. Cette entité sert à la mise en œuvre du mécanisme de contrôle d'accès qui permet de gérer les requêtes d'accès et les décisions d'autorisation. C'est à ce niveau que le choix de l'architecture de contrôle d'accès est effectué. Comme nous l'avons expliqué dans la section 3.1.2, nous nous basons sur des politiques de décentralisation du mécanisme de contrôle. Le schéma de déploiement contient des PDPs et des PEPs en fonction de la structure des sessions et des nœuds qui participent à l'activité collaborative. Les connexions entre ces composants sont également détaillées dans ce schéma. Le schéma de déploiement généré peut être donc utilisé afin d'effectuer un déploiement réel et effectif des composants logiciels de contrôle d'accès.

Le gestionnaire de déploiement - Deployment Manager

Le gestionnaire de déploiement se charge du déploiement et de la reconfiguration des composants de contrôle d'accès en se basant sur les schémas de déploiement fournis par le planificateur de contrôle d'accès. Le processus de déploiement sera détaillé dans le chapitre suivant.

3.3.3 Processus d'adaptation : phases d'analyse et de planification

La Figure 3.12 représente les schémas générés par chaque entité du framework au cours des phases d'analyse et de planification. Un schéma fourni par une entité présente le paramètre d'entrée d'une autre entité qui utilise

un ensemble de techniques permettant d'effectuer des transformations. Les entités que nous détaillons dans cette section sont : l'analyseur, le générateur des besoins de contrôle d'accès et le générateur des graphes de déploiement.

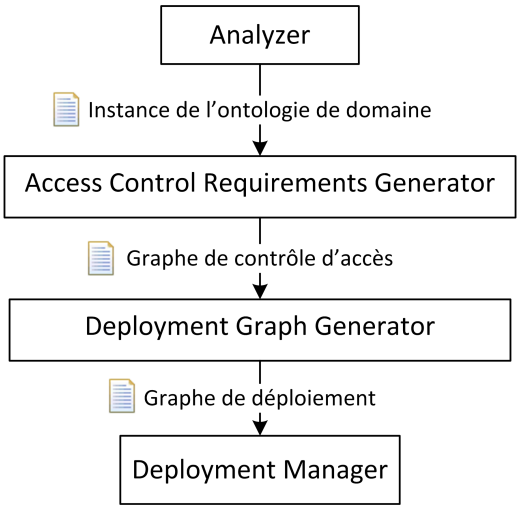

FIGURE 3.12 – *Transformations au cours des phases d'analyse et de planification*

Schéma fourni par l'analyseur

Afin de représenter les changements dans l'environnement qui nécessitent une adaptation, l'analyseur utilise l'ontologie de domaine de l'application considérée. Le schéma généré par l'analyseur est une instance de l'ontologie qui contient des individus reliés par des propriétés représentant les nœuds, leurs dispositifs, leurs rôles et les groupes auxquels ils appartiennent. L'ensemble de ces individus représente les aspects de collaboration de l'application à un moment donné. Cette instance regroupe des individus de l'ontologie de domaine et ceux de GACO.

Schéma fourni par le générateur des besoins de contrôle d'accès

Le générateur des besoins de contrôle d'accès tient compte de chaque instance fournie par l'analyseur afin de construire un graphe, appelé graphe de contrôle d'accès, qui exprime les besoins de contrôle d'accès à un moment donné. Après chaque réception d'une instance de l'ontologie de domaine, le générateur des besoins de contrôle d'accès se base sur le mécanisme d'inférence d'OWL et sur le traitement des règles SWRL afin d'inférer une nouvelle instance qui permet ensuite de créer un graphe de contrôle d'accès. En effet, l'instance inférée contient explicitement tous les éléments qui doivent être présents dans le graphe de contrôle d'accès.

Supposons que nous ayons une instance de l'ontologie de domaine. L'exécution des règles SWRL permettra de :

1. Établir les sessions et affecter chaque individu du concept *Role* à une session en utilisant les propriétés *involvedIn* et *involvesRole*. Ceci est réalisé à travers l'exécution des règles SWRL de l'ontologie de domaine. Ces règles sont définies lors de la quatrième étape du processus de

création de l'ontologie de domaine (Figure 3.8). Elles permettent d'indiquer comment les rôles communiquent entre eux dans chaque session.

2. Créer des individus du concept *AccessControlComponent* qui permettent de sécuriser l'accès des différents nœuds dans chaque session. Ces individus seront liés aux nœuds par la relation *managesAccessOf* et aux sessions par la relation *managesAccessInSession*. Ceci est fait par la règle access_*control_components_generation* de GACO. Un exemple d'application de cette règle est illustré dans Figure 3.6 et Figure 3.7.

Une fois la nouvelle instance est produite par l'application des règles, elle sera parcourue afin d'extraire les éléments du graphe de contrôle d'accès qui représentent les composants, les nœuds, les dispositifs et les sessions. Ce graphe est exprimé en GraphML [1]. GraphML est un format de fichier basé sur XML pour l'expression des graphes. Il permet l'expression de plusieurs types de graphes tels que les graphes orientés et non orientés et les graphes hiérarchiques. En plus, il supporte les représentations graphiques et le référencement vers des données externes. Nous avons choisi ce langage car il utilise une syntaxe simple qui facilite l'analyse et le décodage des graphes.

Les sommets du graphe de contrôle d'accès représentent les composants de contrôle d'accès. Chaque sommet est représenté par une étiquette composée de cinq champs : l'identificateur du composant, le type du composant (dans notre cas, c'est un *AccessControlComponent*), l'identificateur du dispositif d'où proviennent les requêtes d'accès, la session dans laquelle il contrôle l'accès, et un entier qui sera utilisé par le générateur des graphes de déploiement.

La Figure 3.13 montre un exemple de graphe de contrôle d'accès exprimé en GraphML. Cet exemple décrit un graphe appelé *accessControlSample* qui contient deux composants *acc*1 et *acc*2 de type *AccessControlComponent* qui contrôlent l'accès dans une session *session*1. Le composant *acc*1 contrôle les requêtes qui proviennent du dispositif ayant l'adresse IP 192.168.1.1. Tandis que le composant *acc*2 contrôle les requêtes qui proviennent du dispositif ayant l'adresse IP 192.168.1.2. Nous décrirons dans la prochaine section comment le quatrième attribut sera utilisé par le générateur des graphes de déploiement.

Schéma fourni par le générateur des graphes de déploiement

Le graphe de contrôle d'accès sera utilisé pour produire un graphe de déploiement qui décrit l'ensemble des composants physiques qui doivent être déployés dans les machines des utilisateurs et la façon dont ils sont connectés. Pour cela, nous devons choisir l'architecture de contrôle d'accès qui spécifie la répartition des composants logiciel de contrôle d'accès ainsi que les connexions qui doivent être établies entre eux. Nous avons retenu l'architecture décentralisée et le modèle de communication décrits dans la section 3.1.2. Ce modèle de communication permet l'interconnexion des composants de contrôle d'accès en mode synchrone. Comme nous l'avons mentionné dans la section 3.1.2, les deux types de composants de contrôle d'accès sont : les PDPs et les PEPs. Des liens de connexions doivent relier

1. http://graphml.graphdrawing.org/

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
  <graph id="accessControlSample" edgedefault="directed">
    <node id="acc1">
      <data key="attribute1">AccessControlComponent</data>
      <data key="attribute2">192.168.1.1</data>
      <data key="attribute3">session1</data>
      <data key="attribute4">1</data>
    </node>
    <node id="acc2">
      <data key="attribute1">AccessControlComponent</data>
      <data key="attribute2">192.168.1.2</data>
      <data key="attribute3">session1</data>
      <data key="attribute4">0</data>
    </node>
  </graph>
</graphml>
```

FIGURE 3.13 – *Exemple de graphe de contrôle d'accès*

chaque PEP à un PDP afin de lui permettre d'envoyer les requêtes d'accès au PDP auquel il est connecté. Le schéma de déploiement associe à chaque session un PDP et un ensemble de PEPs. La mise en place d'un PDP dans chaque session permet de décentraliser le contrôle d'accès et donc assurer un passage à l'échelle dans l'environnement collaboratif. Les PEPs qui interceptent les requêtes d'accès et qui appliquent les décisions d'autorisation sont déployés chacun dans un dispositif. Chaque PEP contrôle les accès provenant d'un dispositif dans une session donnée. Par conséquent, le nombre de PEPs déployés dans un dispositif dépend du nombre de sessions auxquelles appartient l'utilisateur de ce dispositif.

En conséquence, un graphe de déploiement contient un ensemble d'entité de type PDP et PEP reliées entre elles. Il peut être décrit en GraphML comme suit :

- Les sommets représentent les composants PEPs et PDPs. Chaque sommet est représenté par une étiquette composée de quatre champs : l'identificateur du composant, son type (PDP ou PEP), la session dans laquelle il contrôle l'accès aux ressources et l'identificateur du dispositif où il doit être déployé.
- Les arrêtes représentent les liens entre ces entités.

La Figure 3.14 illustre un exemple de graphe de déploiement. Ce graphe contient un PDP appelé $pdp1$, deux PEPs appelés $pep1$ et $pep2$. Le $pdp1$ doit être déployé dans le dispositif ayant l'adresse IP 192.168.1.1, tandis que $pep1$ et $pep2$ doivent être déployés dans les deux dispositifs ayant respectivement les adresses IP 192.168.1.1 et 192.168.1.2. Le composant $pdp1$ gère les décisions d'autorisation dans la session $session1$. Les deux composants $pep1$ et $pep2$ sont connectés au $pdp1$ et gèrent les requêtes des dispositifs ayant les adresses IP 192.168.1.1 et il 192.168.1.2.

Processus de génération des graphes de déploiement

Afin de générer un graphe de déploiement à partir d'un graphe de collaboration, nous utilisons des techniques de transformation de graphes. Plus précisément, nous utilisons la hiérarchie de Chomsky [Cho56], considérée

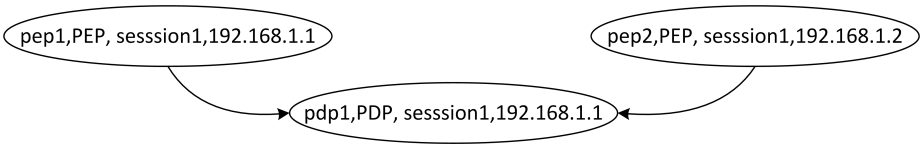

FIGURE 3.14 – *Exemple d'un graphe de déploiement*

comme la hiérarchie la plus connue de la théorie des langages formels. Cette hiérarchie fut définie en terme de grammaires appelées *grammaires génératives et transformationnelles* de Chomsky.

Une grammaire G peut être décrite par un quadruplet $G = (NT, T, S, P)$ où NT est l'ensemble des symboles non-terminaux ou variables de la grammaire, T est l'ensemble des symboles terminaux de la grammaire, S est l'axiome ou racine de la grammaire et P est l'ensemble des règles de la grammaire. Les parties gauches des règles de P contiennent au moins un symbole non terminal. Pour profiter du mécanisme de transformation de graphes, nous utilisons la grammaire de graphe définie par Guenoun [Gueo6]. Dans cette grammaire, l'application d'un ensemble de production à un graphe qui contient des symboles non-terminaux produit un ensemble de graphes qui contient des symboles terminaux. Une production ou règle de P est de la forme d'un triplet (L, K, R). Une règle est applicable à un graphe G s'il y a une occurrence de L dans G. L'application d'une règle implique la préservation du sous-graphe K de G, la suppression de l'occurrence du sous-graphe $Del = (L \setminus K)$ et le rajout d'une copie du graphe $Add = (R \setminus K)$.

Dans notre cas, nous appliquons un ensemble de règles à un graphe de contrôle d'accès qui contient des symboles non-terminaux afin de produire un ou plusieurs graphes de déploiement qui contiennent des nœuds terminaux de types PEP et PDP. Nous expliquons dans ce qui suit le cas où on génère un seul graphe et l'autre cas où nous avons besoin de générer un ensemble de graphes.

Le diagramme d'activité, illustré par la Figure 3.15, représente le comportement du générateur du graphe de déploiement suite à la réception d'un graphe de contrôle d'accès. Il représente les étapes nécessaires qui permettent la génération d'un graphe de déploiement.

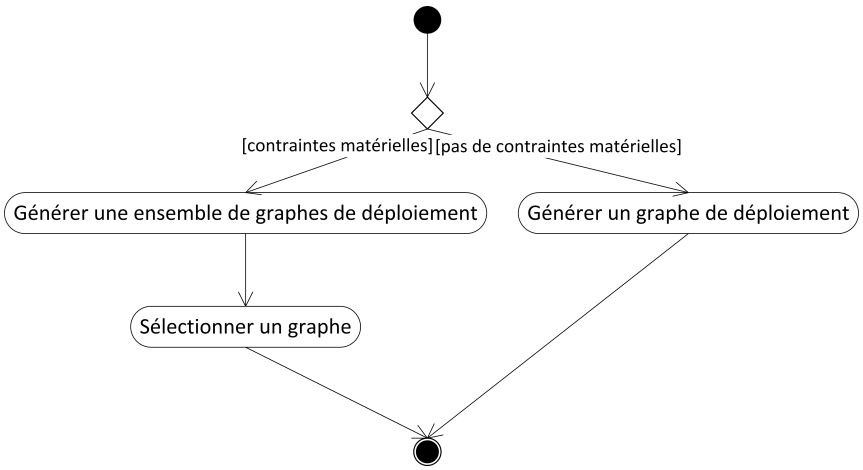

FIGURE 3.15 – *Processus de génération des graphes de déploiement*

On distingue deux cas :

1. Le cas où les machines des utilisateurs ont des capacités limitées et peuvent avoir des contraintes de ressources matérielles. Les ressources qui peuvent être critiques dans ces machines sont la batterie, la mémoire, le CPU, etc. Dans un tel environnement, le déploiement doit tenir compte de ces contraintes imposées par les ressources matérielles. Dans ce cas, nous établissons une grammaire de graphe GG_{mult}, illustrée par le Tableau 3.1, qui permet de générer un ensemble de graphes de déploiement et nous utilisons une fonction de sélection qui permet de sélectionner le graphe le plus adapté à l'état actuel des ressources matérielles.
   Les nœuds non-terminaux sont représentés par un quadruplet $(id, access\_control\_component, m, s)$ où id représente l'identificateur du nœud, $access\_control\_component$ représente le type du composant, m représente l'identificateur de la machine dont l'utilisateur est impliqué dans la session s. Les nœuds terminaux sont représentés aussi par un quadruplet $(id, type, s, m)$ où id représente l'identificateur du nœud, $type$ représente le type du composants (PEP ou PDP), s représente la session à laquelle il appartient et m représente l'identificateur de la machine sur laquelle il doit être déployé. La règle $r1$, illustrée dans la Figure 3.16, permet de chercher le sous graphe $L1$ composé des deux nœuds $acc1$ et $acc2$ de type access_control_component et qui gèrent respectivement l'accès des deux machines $m1$ et $m2$ dans une session s. Ce sous-graphe est supprimé par la règle et remplacé par le sous-graphe $R1$ composé d'un PDP $pdp1$ déployé sur la machine $m1$, auquel sont connectés deux PEPs $pep1$ et $pep2$ déployés respectivement sur les deux machines $m1$ et $m2$. Nous aurions pu mettre un seul nœud de type access_control_component dans $L1$, mais nous supposons qu'il y a au moins deux utilisateurs qui collaborent dans une session donnée.

$GG_{mult} = (NT, T, S, P)$ où : $NT = \{(id, "access\_control\_component", m, s)\}$, $T = \{(id, "PEP", s, m), (id, "PDP", s, m)\}$, $S = \{\}$ *et* $P = \{r1, r2\}$
$r1 = (L1, K1, R1)$ où : $L1 = \{(acc1, "access\_control\_component", m1, s),$ $(acc2, "access\_control\_component", m2, s)\}$; $K1 = \{\}$; $R1 = \{(pep1, "PEP", s, m1) \rightarrow (pdp1, "PDP", s, m1),$ $(pep2, "PEP", s, m2) \rightarrow (pdp1, "PDP", s, m1)\}$
$r2 = (L2, K2, R2)$ où : $L2 = \{(pdp1, "PDP", s, m1),$ $(acc1, "access\_control\_component", m2, s)\}$; $K2 = \{(pdp1, "PDP", s, m1)\}$; $R2 = \{(pep1, "PEP", s, m2) \rightarrow (pdp1, "PDP", s, m1)\}$

Tab. 3.1 – *Grammaire de graphe* GG_{mult}

La règle $r2$, illustrée dans la Figure 3.17, permet de cher-

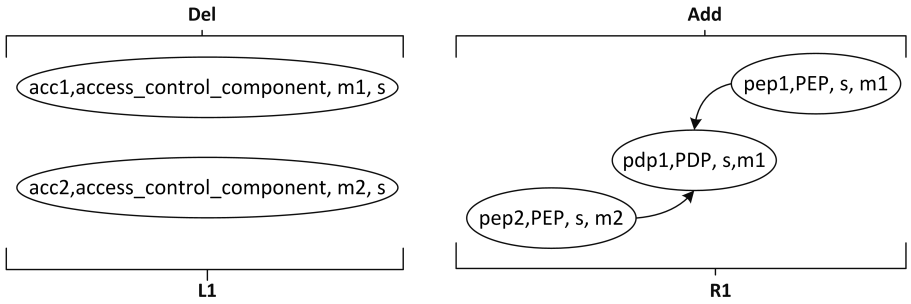

FIGURE 3.16 – *Illustration de la règle r*1 *de la grammaire de graphe* GG_{mult}

cher le sous-graphe $L2$ qui contient un nœud $acc1$ de type access_control_component et un PDP $pdp1$ dans deux machines différentes $m2$ et $m1$ et qui sont associés à une même session s. Le nœud $acc1$ sera supprimé et remplacé par un composant $pep1$ de type PEP déployé sur la même machine $m2$ et connecté au PDP $pdp1$.

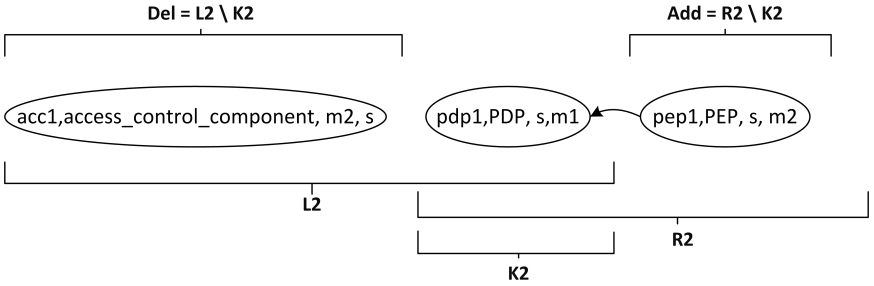

FIGURE 3.17 – *Illustration de la règle r*2 *de la grammaire de graphe* GG_{mult}

Afin d'appliquer ces règles, nous utilisons un outil de transformation de graphes appelé GMTE (*Graph Matching and Transformation Engine*) [DBR10] [RGD+08] [HBRDPH13]. Dans notre cas, le moteur applique récursivement chaque règle selon son ordre dans la grammaire aux graphes de contrôle d'accès. L'application des règles de la grammaire GG_{mult} à un graphe de contrôle d'accès produit toutes les combinaisons possibles des graphes de déploiement. Par exemple, si on suppose que le graphe de contrôle d'accès contient deux nœuds $(acc1, "access\_control\_component", m1, s)$ et $(acc2, "access\_control\_component", m2, s)$, le résultat de l'application des règles produit un graphe de déploiement $g1$ dans lequel le PDP est déployé dans la machine $m1$ et un autre graphe de déploiement $g2$ dans lequel le PDP est déployé dans la machine $m2$. Afin de sélectionner le graphe le plus adapté à l'état actuel des ressources matérielles, nous utilisons la fonction de sélection proposée dans [RDCJ09].

2. Dans l'autre cas, nous considérons que les ressources matérielles sont toujours disponibles dans les machines des utilisateurs. Dans ce cas, le déploiement des composants logiciels de contrôle d'accès peut ne pas tenir compte des contraintes imposées par les ressources matérielles. Dans ce cas, nous définissons une grammaire de graphes GG_{simple}, illustrée par le Tableau 3.2, qui permet de générer un seul graphe de déploiement à partir d'un graphe de contrôle d'accès. Dans les nœuds non-terminaux, nous rajoutons un attribut *int* qui s'écrit dans la règle

sous la forme d'un entier permettant de distinguer les nœuds. Cet entier peut avoir deux valeurs 0 ou 1. Dans chaque session, nous choisissons un nœud auquel on associe un entier de valeur 1. Ce nœud sera considéré comme un nœud central qui héberge le composant PDP de la session. Les autres nœuds auront un entier ayant comme valeur 0. Les nœuds terminaux de la grammaire GG_{simple} ont la même forme que ceux de la grammaire GG_{mult}.
La règle $r1$, détaillée dans la deuxième ligne du Tableau 3.2, permet de chercher un sous graphe $L1$ composé d'un nœud de type *access_control_component* auquel est associé un entier ayant comme valeur 1 est une machine $m1$. Ce nœud sera supprimé est remplacé par un sous-graphe $R1$ composé d'un PDP $pdp1$ déployé sur la machine $m1$ auquel est connecté un PEP $pep1$ déployé sur la même machine.
La règle $r2$, détaillée dans la troisième ligne du Tableau 3.2, permet de chercher un sous-graphe $L2$ composé de deux nœuds d'une même session s : un PDP $pdp1$ déployé dans la machine $m1$ et un nœud $acc1$ de type *access_control_component* qui gère les accès d'une machine $m2$ et auquel est associé un entier ayant comme valeur 0. En appliquant la règle $r2$, le nœud $acc1$ sera supprimé et remplacé par un composant $pep1$ de type PEP déployé sur la machine $m2$ et connecté au PDP $pdp1$.
L'application de la grammaire GG_{simple} à un graphe de contrôle d'accès produit un graphe de déploiement qui associe à chaque session un PDP qui gère les décisions d'autorisation et autant de PEP que de machines. Tous les composants PEPs et PDPs présents dans le graphe de déploiement doivent être déployés par le gestionnaire de déploiement.

$GGsimple = (NT, T, S, P)$ où : $NT = \{(id, "access\_control\_component", m, s, int)\}$, $T = \{(id, "PEP", s, m), (id, "PDP", s, m)\}$, $S = \{\}$ et $P = \{r1, r2\}$
$r1 = (L1, K1, R1)$ où : $L1 = \{(acc1, "access\_control\_component", m1, s, 1)\}$; $K1 = \{\}$; $R1 = \{(pep1, "PEP", s, m1) \rightarrow (pdp1, "PDP", s, m1)\}$
$r2 = (L2, K2, R2)$ où : $L2 = \{(pdp1, "PDP", s, m1),$ $(acc1, "access\_control\_component", m2, s, 0)\}$; $K2 = \{(pdp1, "PDP", s, m1)\}$; $R2 = \{(pep1, "PEP", s, m2) \rightarrow (pdp1, "PDP", s, m1)\}$

Tab. 3.2 – *Grammaire de graphe* GG_{simple}

3.3.4 Conception du framework

Dans cette section, nous présentons la conception de notre framework à travers un diagramme de classe UML. Ensuite, nous décrivons comment il peut être utilisé dans les systèmes collaboratifs ubiquitaires pour sécuriser

l'accès aux ressources. Gamma et al. [GHJV95] définissent le terme framework comme suit :

« *un framework est un ensemble de classes qui définissent une architecture qui peut être réutilisée par des logiciels spécifiques.* »

Un framework est une boite à outil destiné à être utilisée par les développeurs de logiciels. Il joue le rôle d'un patron de conception pour faciliter le développement d'applications bien spécifiques. Les framework sont plus ou moins similaires aux patrons de conception. Néanmoins, les patrons de conception sont plus abstraits et moins spécifiques que les framework. L'avantage des frameworks est qu'ils peuvent être développés en utilisant des langages de programmation et ils peuvent être exécutés et réutilisés. Un framework dicte l'architecture des applications qui l'utilisent. En effet, il définit la structure globale de l'application, les relations entre les classes et leurs responsabilités.

Diagramme de classe du framework

Le diagramme de classes UML, illustré par la Figure 3.18, représente une vue simplifiée des principales classes qui constituent notre framework. La conception du framework est guidée par la boucle de traitement autonomique. En effet, chaque fonction de la boucle est modélisée par une classe qui coopère avec d'autres classes afin d'atteindre des objectifs bien définis.

La classe *Analyzer* permet d'effectuer la fonction d'analyse de la boucle autonomique. Elle contient principalement un attribut qui représente une instance de l'ontologie de domaine. Elle contient également une méthode *model_changed()* qui permet de mettre à jour l'instance en fonction des changements effectués dans l'application et de notifier le framework de contrôle d'accès du changement. Cette classe peut être étendue par l'application qui utilise le framework afin de redéfinir la méthode qui permet la mise à jour de l'instance de l'ontologie.

La fonction de planification de la boucle est représentée par les deux classes *AccessControlReqGenerator* et *DeploymentGraphGenerator*. La classe *AccessControlReqGenerator* utilise la classe *OntologyManager* qui permet d'effectuer les inférences et exécuter les règles SWRL afin d'obtenir une nouvelle instance qui traduit les besoins de contrôle d'accès. La classe *AccessControlGraphGenerator* est aussi utilisée par la classe *AccessControlReqGenerator* afin de générer un graphe de contrôle d'accès. Ce graphe est une instance de la classe *Graph* qui consiste en un ensemble d'instances de la classe *Node* et un ensemble d'instances de la classe *Edge*. La deuxième classe associée à la fonction de planification est la classe *DeploymentGraphGenerator*. Cette classe utilise le moteur de transformation de graphe GMTE pour la génération d'un ou plusieurs graphes de déploiement, et la classe *Selector* pour la sélection.

La classe *Handler* permet de coordonner les échanges entre les trois classes *Analyzer*, *AccessControlReqGenerator* et *DeploymentGraphGenerator*. Les relations entre la classe *Handler* et ces trois classes sont bidirectionnelles. Cela veut dire que la classe *Handler* a une référence vers chacune des trois classes qui ont à leurs tours une référence vers la classe *Handler*. Quand un objet de la classe *Analyzer* produit une instance de l'ontologie de domaine, elle le passe à un objet de la classe *Handler*, qui a son tour le passe à

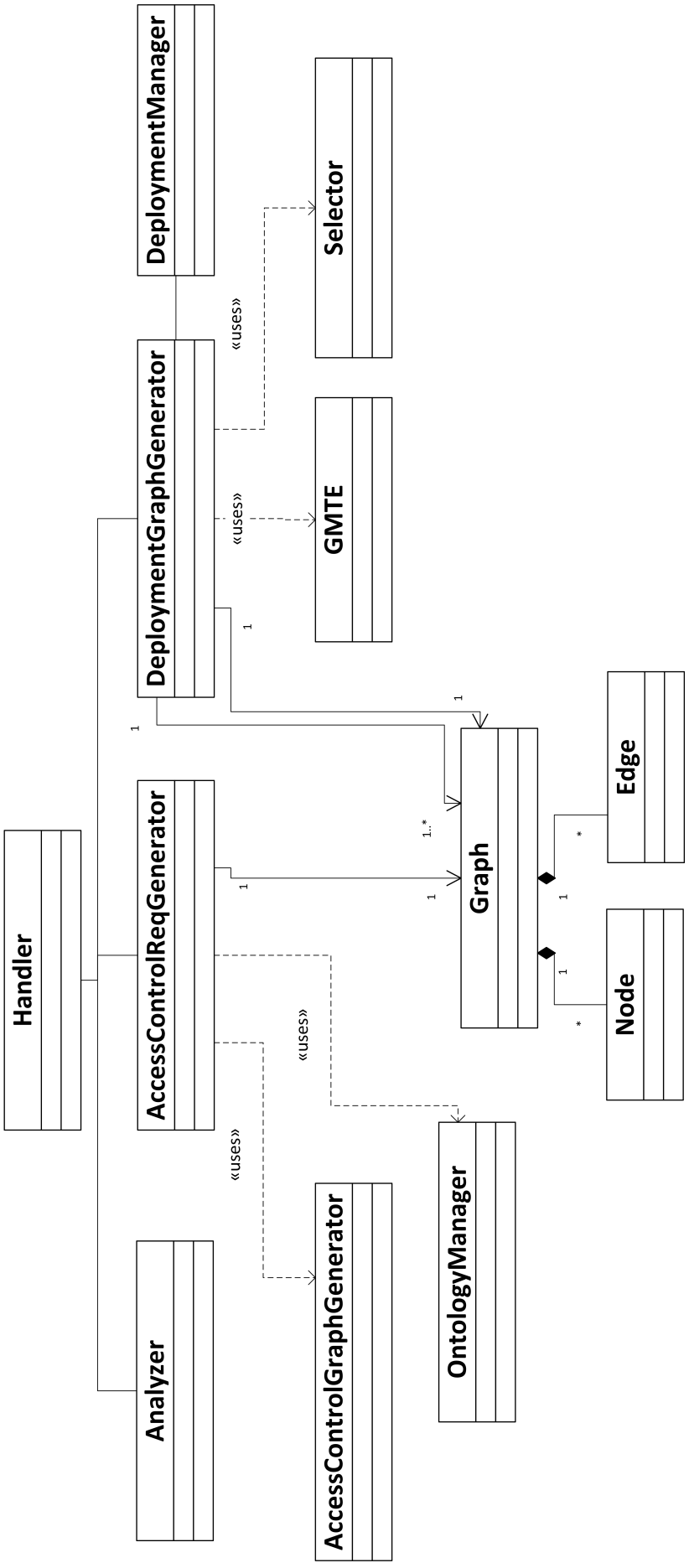

FIGURE 3.18 – *Diagramme de classe UML du framework*

un objet de la classe *AccessControlReqGenerator*. Cet objet génère un graphe de contrôle d'accès et le retourne à l'objet *Handler* qui le passe à un objet de la classe *DeploymentGraphGenerator*. Ce dernier génère un graphe de déploiement qui sera intercepté par un objet de la classe *DeploymentManager* afin de déployer les composants de contrôle d'accès sur les machines des utilisateurs. Ces étapes constituent un processus d'adaptation qui sera exécuté dans un thread. Nous discutons dans le chapitre suivant le mécanisme de traitement des threads afin d'éviter les incohérences de déploiement des composants.

Utilisation du framework

Notre framework définit une structure sur laquelle peut se baser les applications qui offrent des moyens de collaboration. Il est destiné à être utilisé dans les applications collaboratives qui se basent sur une architecture client-serveur. Les serveurs des applications gère la communication entre les clients et contrôle l'accès aux ressources à travers notre framework. Les composants de contrôle d'accès sont répartis sur la machine centrale et sur les différents dispositifs des utilisateurs. Nous décrivons dans ce qui suit les étapes que doivent suivre les concepteurs des applications collaboratives afin d'utiliser notre framework.

Étape 1 : Création de l'ontologie de domaine Le concepteur de l'application doit définir l'ontologie de domaine qui contient tous les éléments spécifiques au domaine de l'application considérée. Cette ontologie contient également les règles qui expriment les situations de collaboration. La création de l'ontologie de domaine peut être réalisée en utilisant un éditeur d'ontologie OWL tel que Protégé [2] .

Étape 2 : Prise en compte des changements de l'état de l'application Le serveur de l'application doit implémenter la méthode *modelChanged()* de la classe *Analyzer*. Cela peut se faire à travers une classe qui hérite de la classe *Analyzer* et qui implémente la méthode *modelChanged()*. Cette méthode doit être appelée par le serveur de l'application suite aux changements qui nécessitent l'exécution un processus adaptation. Cette méthode peut être implémentée avec différentes façons selon la logique métier de l'application.

Il existe des APIs pour la création et la modification des instances des ontologies telles qu'OWL API [3] . Les développeurs des applications collaboratives doivent maîtriser ces APIs afin de pouvoir les utiliser. Pour cela, nous proposons une API qui peut être utilisée par les développeurs pour la création et la mise à jour de l'instance de l'ontologie de domaine. Cette API est indépendante de l'ontologie de domaine. Cela veut dire que plusieurs applications appartenant à différents domaines peuvent l'utiliser. En conséquence, le code de l'application reste inchangé et cohérent même si l'ontologie de domaine change.

Les services proposés par cette API sont :

– connect (String userId, String ip, List<String> roles, List<String> groups) : Cette fonction est utilisée quand un utilisateur se connecte

2. Disponible sur http://protege.stanford.edu/
3. Disponible sur http://owlapi.sourceforge.net/

au serveur d'application. Elle utilise son identifiant, l'adresse IP de son dispositif, la liste de ses rôles et la liste des groupes qu'il veut rejoindre. Notamment, elle crée des instances des concepts *Node*, *Role*, *AccessPoint* et les relations entre elles.
- quit (String userId) : cette fonction est utilisée quand un utilisateur ayant l'identifiant *userId* se déconnecte. Elle prend en charge la mise à jour de l'instance de l'ontologie en supprimant le nœud de l'utilisateur, son dispositif et ses rôles.
- addToGroup (String userId, String group) : cette fonction est utilisée quand un utilisateur demande de rejoindre un groupe, elle se charge d'ajouter tous les rôles de cet utilisateur au groupe indiqué.
- removeFromGroup (String userId, String group) : cette fonction est appelée quand un utilisateur quitte un groupe donné. Elle permet de supprimer tous les rôles d'un utilisateur ayant l'identifiant *userId* du groupe indiqué par l'argument *group*.
- addRole (String userId, String role) : cette fonction est utilisée pour ajouter un rôle *role* à un utilisateur ayant l'identifiant *userId*. Ce rôle sera ajouté à tous les groupes auxquels appartient cet utilisateur.
- removeRole (String userId, String role) : cette fonction est utilisée pour retirer le rôle *role* de l'utilisateur ayant l'identifiant *userId*. Ce rôle sera retiré de tous les groupes auxquels appartient l'utilisateur.
- changeRole (String userId, String previousRole, String newRole) : Cette fonction est utilisée quand un utilisateur ayant l'identifiant *userId* change de rôle au cours de la collaboration. L'ancien rôle sera retiré de tous les groupes auxquels il appartient. Tandis que le nouveau rôle sera ajouté à tous les groupes auxquels il appartient.

Étape 3 : Instanciation du framework Afin de pouvoir exécuter les processus d'adaptation du framework, un serveur d'application doit créer une instance de la classe *Handler* qui sera utilisée pour créer une instance de la classe *Analyzer*. Le serveur doit également initialiser l'ontologie de domaine en utilisant la méthode *initialize()* fournie par l'API. Une fois cela est fait, le framework est prêt à effectuer des processus d'adaptation et de mettre en place et reconfigurer les composants de contrôle d'accès. A chaque fois un changement de l'état de l'application aura lieu, la méthode *modelChanged()* de la classe *Analyzer* est appelée afin de mettre à jour l'instance de l'ontologie de domaine et effectuer un processus d'adaptation.

Conclusion du chapitre

Dans ce chapitre, nous avons présenté notre approche pour l'adaptation de l'architecture de contrôle d'accès. D'abord, nous avons décrit nos choix conceptuels pour la décentralisation des composants de contrôle d'accès dans les environnements collaboratifs ubiquitaires. Nous avons justifié le choix de répartition des entités du standard XACML afin de décentraliser le contrôle et assurer des échanges standard de requêtes d'accès et de décisions d'autorisation. Ensuite, nous avons présenté l'ontologie GACO et nous avons expliqué comment les concepteurs d'applications peuvent l'étendre

pour exprimer les concepts et les relations spécifiques au domaine considéré. Dans la troisième partie du chapitre, nous avons décrit notre framework de contrôle d'accès et nous avons détaillé le processus d'adaptation qui assure la reconfiguration dynamique de l'architecture logicielle de contrôle d'accès en fonction du changement du contexte. Dans le chapitre suivant nous détaillons le processus de déploiement des composants dans les machines qui constituent le réseau.

Gestion du Déploiement

4

Sommaire

Ce chapitre est consacré à notre apport pour le déploiement des composants logiciels dans les environnements collaboratifs ubiquitaires. Cet apport est exploité, d'une part, pour le déploiement des composants de contrôle d'accès du framework proposé dans le chapitre 3, et d'autre part, pour le déploiement des composants du framework de communication présenté dans [BRSV+09], [KTD12a] et [KTD12b]. Dans la première partie de ce chapitre, nous présentons les phases de conception et d'exécution des deux frameworks. La dernière étape de la phase d'exécution consiste à déployer les composants logiciels de communication et de contrôle d'accès dans les dispositifs des utilisateurs. Cette étape est détaillée dans la deuxième partie du chapitre dans laquelle nous présentons toutes les techniques de déploiement que nous avons proposées. Nous commençons la deuxième partie de ce chapitre par présenter la technologie OSGi (*Open Services Gateway initiative*) [Theo7]. Ensuite, nous détaillons le processus de déploiement et les interactions entre les différentes entités qui assurent le déploiement[EMKI+14]. Puis, nous proposons une politique qui permet de tenir compte des résultats de déploiement précédents afin d'éviter les incohérences de déploiement. Puis, nous décrivons les problèmes causés par la nature monotone de OWL et nous proposons des solutions afin d'y remédier. A la fin du chapitre, nous décrivons comment et quand les processus d'adaptation doivent être déclenchés en fonction du niveau de dynamicité de l'environnement collaboratif afin d'assurer la fiabilité du déploiement.

4.1 Cadre d'application du déploiement

Dans cette sous-section, nous présentons le processus d'adaptation du framework de communication et du framework de contrôle d'accès afin de décrire le cadre dans lequel nous appliquons le mécanisme de déploiement que nous décrirons dans la section suivante.

L'objectif du framework de communication est d'assurer une adaptation dynamique des architectures logicielles de communication dans les environnements collaboratifs ubiquitaires dans lesquels un ensemble d'utilisateurs jouent un ou plusieurs rôles dans différents groupes. Dans chaque groupe, les utilisateurs utilisent différents types de dispositifs et communiquent dans les sessions de collaboration en fonction de leurs rôles et de leurs responsabilités. Ils peuvent rejoindre/quitter des groupes et activer/désactiver un ensemble de rôles au cours de l'activité collaborative. Afin d'assurer une adaptation aux changements, un ensemble de composants logiciels de communication doivent être sélectionnés et déployés dans les différents dispositifs afin de permettre la communication entre les utilisateurs dans les sessions auxquelles ils appartiennent. Puisque les rôles des utilisateurs et leurs appartenances aux groupes peuvent changer au cours de la collaboration, le déploiement des composants doit être adaptatif afin de répondre aux besoins des utilisateurs au cours de la collaboration. Le déploiement des composants de communication permet d'établir les sessions de collaboration en fonction des rôles des utilisateurs. Après chaque changement au niveau de l'activité collaborative, les actions d'adaptation permettent de reconfigurer dynamiquement les composants de communication. Par exemple, si un utilisateur rejoint un groupe et demande de communiquer en audio, le framework est capable de déployer dans son dispositif un composant de communication qui lui permet d'échanger des flux audio avec les autres utilisateurs de la même session.

La Figure 4.1 présente les phases de conception et d'exécution du framework de communication et du framework de contrôle d'accès.

4.1.1 Phase de conception

Le framework de communication se base sur les ontologies OWL et les techniques de raisonnement afin d'effectuer les adaptations. Il utilise principalement une ontologie générique de collaboration (*Generic Collaboration Ontology* ou GCO) [STV10] qui permet d'exprimer les situations et les besoins de collaboration à chaque instant de l'activité collaborative. Lors de la phase de conception, les concepteurs des applications collaboratives peuvent étendre GCO avec des éléments spécifiques au domaine considéré. Ils peuvent aussi définir les règles de raisonnement SWRL qui expriment les situations de collaboration. La dernière étape de la phase de conception des applications qui utilise le framework de communication consiste à définir le modèle middleware de communication. Le modèle qui a été retenu est le modèle de la communication basée sur les évènements ou EBC (*Event-Based Communication*) [MC02]. Ce modèle introduit trois types de composants : les producteurs d'évènements (*Event Producer* ou EP), les consommateurs d'évènements (*Event Consumer* ou EC) et les gestionnaires de canal (*Channel Manager* ou CM). Ce modèle est instancié lors de la phase d'exécution

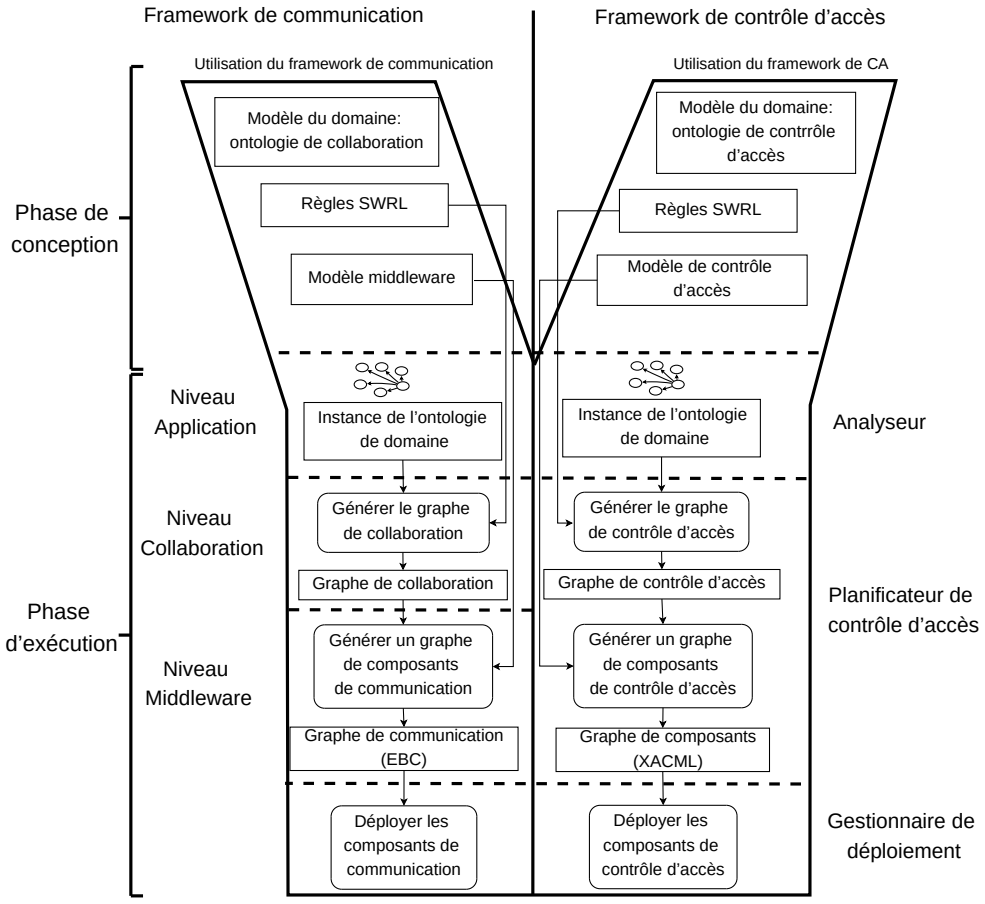

FIGURE 4.1 – *Phases de conception et d'exécution des deux frameworks*

du framework. Une instance de ce modèle est composée d'un ensemble de composants de type EP et EC connectés à des composants de type CM. Les EP envoient des données au CM auquel ils sont connectés et les CM envoient les données aux ECs qui y sont connectés.

La phase de conception des applications qui utilisent le framework de contrôle d'accès consiste à définir les éléments de l'ontologie de domaine qui étendent GACO et à spécifier les règles SWRL qui permettent d'exprimer les situations de collaboration. Ces deux étapes sont détaillées dans la section 3.2.5. En plus de ces deux étapes, les concepteurs d'application doivent définir le modèle de contrôle d'accès pour la gestion des requêtes et des décisions d'autorisation. Ce modèle définit les composants de contrôle d'accès, leurs répartitions et les interconnexions qui doivent être établies entre eux. Le modèle que nous avons retenu pour le framework de contrôle d'accès est décrit dans la section 3.1.2 ; Il se base sur l'architecture fournie par le standard XACML. Une instance de ce modèle contient un ensemble de composants de type PEP connectés à un ensemble de composants de type PDP.

4.1.2 Phase d'exécution

Le framework de communication se base sur une approche de modélisation qui permet une adaptation multi-niveaux. Les niveaux qui ont été rete-

nus sont : le niveau application, le niveau collaboration et le niveau middleware. A chaque niveau est associé un modèle qui se compose d'un ensemble d'entités reliées entre elles.

Le niveau application permet de capturer les changements dans l'environnement collaboratif. Pour cela, les éléments de l'ontologie de domaine, définis lors de la phase de conception, sont utilisés. Une instance de cette ontologie représente les entités qui participent à l'activité collaborative, leurs caractéristiques et la façon dont elles sont organisées.

Après chaque modification de l'instance de l'ontologie de domaine, le niveau collaboration permet de fournir un graphe de collaboration en se basant sur le mécanisme d'inférence et le traitement des règles SWRL définies lors de la phase de conception. Chaque graphe, représenté par une instance de l'ontologie générique de collaboration, identifie l'ensemble de composants de communication qui permettent aux utilisateurs de communiquer et d'échanger des données.

Les graphes de collaboration sont utilisés par le niveau middleware afin de générer un graphe de déploiement selon le modèle middleware choisi lors de la phase de conception. Ce graphe contient une description de tous les composants qui doivent être déployés dans les dispositifs. Dans le cas où le modèle EBC est retenu, les sommets du graphe représentent les éléments du modèle EBC (EP, EC, CM) et possèdent une étiquette composée de l'identificateur du composant, de son type, du type de données qu'il gère (texte, audio ou vidéo), le nom de la session dans laquelle il échange les données et l'identificateur de la machine où il doit être déployé ; Les arrêtes du graphe représentent les liens entre les entités EBC. Ce graphe permet de décrire les composants physiques qui doivent être déployés et la façon dont ils doivent être connectés.

La Figure 4.2 montre un exemple de graphe de déploiement généré par le processus d'adaptation du framework de communication. Le fichier GraphML correspondant décrit un graphe, appelé $depl\_graph1$, qui contient trois éléments du modèle EBC : un EP appelé $ep1$, un CM appelé $cm1$ et un EC appelé $ec1$. Les deux balises $< edge >$ du fichier montrent les liens établis entre les trois éléments. Un premier lien de type push relie $ep1$ à $cm1$, et un autre lien de même type relie $cm1$ à $ec1$. Les trois entités permettent d'échanger des flux de données de type texte dans la session $session1$. Le producteur d'évènements $ep1$ et le gestionnaire de canal $cm1$ doivent être déployés dans le dispositif identifié par l'adresse IP 198.162.1.4. Tandis que le consommateur $ec1$ doit être déployé dans le dispositif identifié par l'adresse IP 198.162.1.5. La Figure 4.3 montre une représentation graphique du graphe $depl\_graph1$.

Dans le chapitre 3, nous avons détaillé la boucle MAPE-K qui représente la phase d'exécution du framework de contrôle d'accès. La phase d'exécution durant les phases d'analyse et de planification permet de produire un graphe de déploiement qui décrit les composants de type PEP et PDP et la façon dont ils doivent être connectés afin de gérer les requêtes et les décisions d'autorisation. Un exemple d'un tel graphe est illustré par la Figure 3.14. Afin de déployer et reconfigurer les différents composants de communication et de contrôle d'accès, nous avons besoin d'un gestionnaire de déploiement qui interagit avec les dispositifs représentant les éléments gérés d'un élément autonomique.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
    <graph id="depl_graph1" edgedefault="directed">
    <node id="ep1">
      <data key="type">EP</data>
      <data key="flow">text</data>
      <data key="session">session1</data>
      <data key="ip">198.162.1.4</data>
    </node>
    <node id="cm1">
      <data key="type">CM</data>
      <data key="flow">text</data>
      <data key="session">session1</data>
      <data key="ip">198.162.1.4</data>
    </node>
    <node id="ec1">
      <data key="type">EC</data>
      <data key="flow">text</data>
      <data key="session">session1</data>
      <data key="ip">198.162.1.5</data>
    </node>
    <edge source="ep1" target="cm1">
      <data key="attribute1">push</data>
    </edge>
    <edge source="cm1" target="ec1">
      <data key="attribute1">push</data>
    </edge>
  </graph>
</graphml>
```

FIGURE 4.2 – *Exemple de graphe de déploiement*

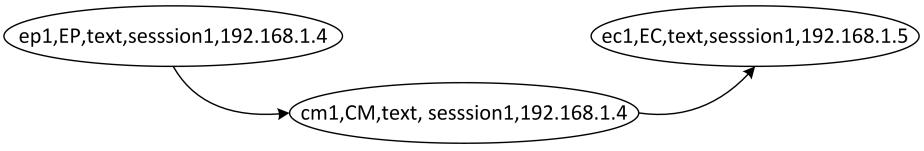

FIGURE 4.3 – *Exemple de graphe de déploiement EBC*

4.2 Déploiement des composants

Dans cette section, nous présentons toutes les étapes du processus de déploiement des composants. Pour cela, nous commençons par décrire la plateforme OSGi sur laquelle nous nous basons pour la manipulation à chaud des composants logiciels.

4.2.1 OSGi

OSGi Alliance est un consortium international pour l'innovation technologique. En 2007, il a créé la spécification de la plateforme OSGi qui fournit une architecture pour les fournisseurs de services, les développeurs et les vendeurs de logiciels pour la gestion des services d'une manière dynamique et qui permet un passage à l'échelle. Le framework OSGi forme le point central de cette spécification. Il permet la gestion de composants et d'applications téléchargeables appelés *bundles*. En utilisant le framework OSGi, les bundles peuvent être téléchargés, installés et désinstallés en temps d'exécution. Par conséquent, les applications reconfigurables développées en utilisant le framework OSGi offrent des services aux utilisateurs même au cours des activités de maintenance et/ou d'évolution. Dans notre cas, les besoins de collaboration peuvent changer au cours des activités collaboratives, ce qui entraîne l'évolution de l'architecture logicielle afin de s'adapter à ces changements et répondre aux nouveaux besoins.

Le framework OSGi permet aux applications développées en Java d'être dynamiquement reconfigurables. Il permet, non seulement de profiter de l'aspect dynamique des architectures orientées services, mais aussi de changer dynamiquement un ensemble de composants qui ont été déployés en temps de conception ou en temps d'exécution.

Un bundle est un fichier JAR qui regroupe un ensemble de classes Java et éventuellement d'autres ressources qui offrent des services aux utilisateurs finaux. Il contient principalement une classe *Activator* utilisée pour le démarrage et l'arrêt du bundle. Après le démarrage du bundle, le code et les ressources peuvent être directement utilisés par les autres bundles dans le framework.

Un ensemble d'opérations ont été définies afin de pouvoir reconfigurer les bundles. Elles sont représentées par un ensemble de méthodes qui agissent sur le cycle de vie des bundles tout en permettant d'installer, de démarrer, de mettre à jour et de désinstaller un bundle. Elles peuvent être invoquées par d'autres composants dans le système ou par le framework OSGi via des lignes de commande. L'exécution d'une méthode cause la transition d'un bundle d'un état vers un autre. La Figure 4.4 montre les différents états que peut avoir un bundle.

Les transitions qui peuvent changer l'état d'un bundle sont décrites comme suit :

- Installation (*Install*) : le framewrok OSGi utilise l'URL du fichier JAR afin de pouvoir l'installer. Le fichier JAR peut être récupéré d'une machine en local ou d'un serveur sur le réseau. Après la récupération du bundle, le framework OSGi détecte les ressources qui doivent importées ou exportées et le nom de la classe *Activator*. Après son installation, le bundle passe à l'état *Installed*.

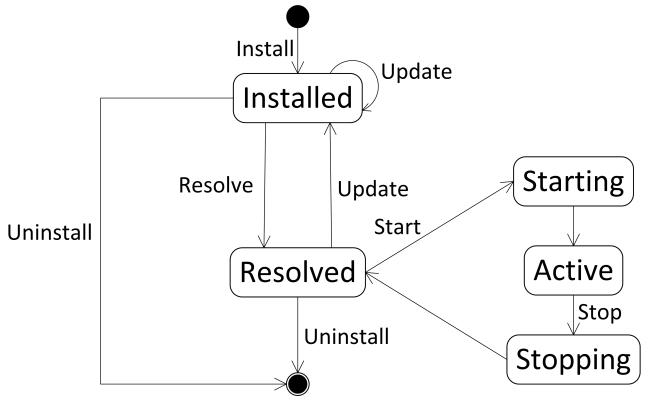

FIGURE 4.4 – *Diagramme états transitions d'un bundle*

- Résolution des dépendances (*Resolve*) : si, après son installation, le bundle a besoin d'importer d'autres paquetages Java, le framework doit résoudre les dépendances en vérifiant si ces paquetages sont bien disponibles. Ce processus est effectué suite à la première demande de démarrage du bundle. Si la résolution des dépendances s'est bien passée, le bundle passe de l'état *Installed* à l'état *Resolved*.
- Démarrage (*Start*) : afin de démarrer un bundle, le framework OSGi instancie la classe *Activator* et invoque la méthode *start* en lui passant un objet *BundleContext* comme paramètre. L'objet *BundleContext* permet l'interaction entre le framework OSGi et le bundle. Après l'invocation de la méthode *start*, le bundle passe à l'état *Starting* qui indique que le bundle est en cours de démarrage. Si le bundle démarre avec succès, il passe à l'état *Active*.
- Arrêt (*Stop*) : le framework OSGi peut appeler la méthode *stop* de l'objet *Activator* pour arrêter un bundle. Cet appel cause le passage à l'état *Stopping* qui indique que le bundle est en train de s'arrêter. Pour arrêter un bundle, le framework effectue les tâches suivantes : (1) arrête les services fournis par le bundle et notifie les composants du système qui l'utilisent et (2) fait passer le bundle vers l'état *Resolved*.
- Désinstallation (*Uninstall*) : Quand cette méthode est appelée, le framework OSGi supprime toutes les ressources reliées au bundle. Si le bundle est à l'état Active, il passe successivement aux états *Stopping*, *Resolved* et finalement à l'état *Uninstalled*. S'il est à l'état *Installed* ou *Resolved*, il passe directement à l'état *Uninstalled*.
- Mise à jour (*Update*) : la mise à jour d'un bundle signifie la migration d'une version vers une autre version plus récente en faisant des changements seulement au niveau de l'implantation du bundle. Notons que lors d'une mise à jour les interfaces ne doivent pas changer.

Après chaque transition d'un bundle d'un état vers un autre, les autres bundles du framework OSGi peuvent s'inscrire pour être notifiés des changements qui ont lieu. Quand il y a un grand nombre de bundles dans le framework OSGi, la gestion des évènements peut devenir complexe au niveau du framework. Par exemple, si un grand nombre de bundles doivent être démarrés, un évènement de démarrage doit être déclenché pour chaque bundle. Afin de contrôler les évènements de démarrage, un service appelé

StartLevel Service a été spécifié pour permettre aux administrateurs d'imposer un ordre de démarrage des bundles.

Nous utilisons la technologie OSGi pour la raison suivante : dans les environnements ubiquitaires, il est difficile d'avoir, au préalable, les outils qui soutiennent la collaboration et qui assurent le contrôle d'accès aux ressources. En effet, des sessions de collaboration peuvent être établies dynamiquement au cours des activités collaboratives. La transformation des composants de collaboration et de contrôle d'accès en bundle OSGi permet de les récupérer, les installer et les démarrer à chaud lors de l'exécution du système. L'avantage de l'utilisation d'OSGi est que ceci se fait d'une manière transparente à l'utilisateur.

Les composants de communication et de contrôle d'accès qui sont déployés dans les machines des utilisateurs sont des bundles OSGi. Afin de pouvoir déployer ces composants, les dispositifs doivent avoir une installation minimale qui consiste en un conteneur OSGi et une JVM.

Les implémentations open source de la spécification OSGi les plus utilisées sont Felix [1], Equinox [2] et Knopplerfish [3] .

4.2.2 Processus de déploiement des composants

Après la génération d'un graphe de déploiement par le niveau middleware du framework de communication ou par le planificateur du framework de contrôle d'accès, le gestionnaire de déploiement se charge de déployer les composants nécessaires dans les dispositifs des utilisateurs et d'établir les connexions entre eux. Le gestionnaire de déploiement est implanté par la classe *DeploymentManager* de la Figure 3.18. Dans cette sous-section, nous présentons le processus de déploiement et l'interaction du gestionnaire de déploiement avec les différentes entités du système.

Le processus de déploiement des composants est illustré par la Figure 4.5. Trois principales entités coopèrent ensemble afin de déployer et reconfigurer les composants. Ces entités sont le gestionnaire de déploiement (*Deployment Manager*), le registre de composants (*Components Repository*) et l'agent de déploiement (*Deployment Agent*). Le gestionnaire de déploiement se situe dans la même machine serveur qui héberge le framework et il peut être appelé en instanciant un objet de la classe *DeploymentManager* et en invoquant la méthode *action()* qui prend comme paramètre le fichier GraphML de déploiement. Ce fichier décrit tous les composants qui doivent être déployés ou reconfigurés. L'agent de déploiement permet de gérer le cycle de vie des bundles dans le dispositif où il est hébergé. En effet, il se charge d'exécuter les opérations demandées sur chaque bundle durant l'activité collaborative. La troisième entité est le registre de composant qui représente un serveur HTTP contenant tous les fichiers JAR des bundles qui peuvent être utilisés. Ce registre contient également un fichier appelé *repository.xml* qui fournit une description de tous les bundles disponibles. Un exemple de ce fichier est illustré par la Figure 4.6. Cet exemple indique que trois bundles CM, EP et EC, qui assurent la communication selon le mode EBC, sont disponibles dans le registre. A chaque bundle est associé un URL qui indique

1. http://felix.apache.org/
2. http://www.eclipse.org/equinox/
3. http://www.knopflerfish.org/

l'emplacement du bundle et comment il peut être récupéré par un agent de déploiement.

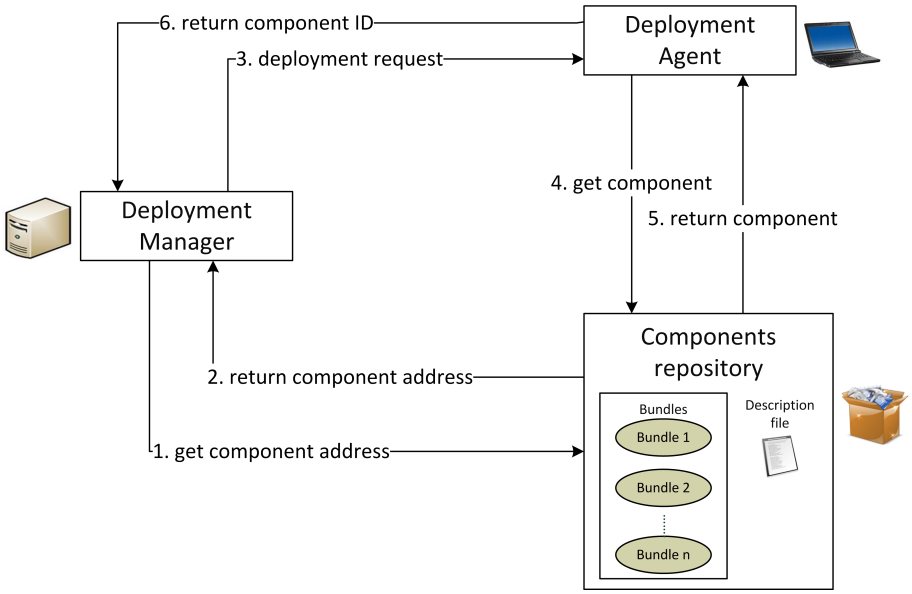

FIGURE 4.5 – *Processus de déploiement des composants*

```
<?xml version="1.0" encoding="UTF-8"?>
<repository>
    <tech name="SOCKET">
        <comm name="EBC">
            <type name="CM">
                <component address="http://140.93.2.140:7000/bundles/CM.jar"/>
            </type>
            <type name="EP">
                <component address="http://140.93.2.140:7000/bundles/EP.jar"/>
            </type>
            <type name="EC">
                <component address="http://140.93.2.140:7000/bundles/EC.jar"/>
            </type>
        </comm>
    </tech>
</repository>
```

FIGURE 4.6 – *Exemple du fichier* repository.xml

Quand le gestionnaire de déploiement est appelé pour exécuter un ensemble d'actions sur des bundles, il parcourt le graphe de déploiement afin de trouver tous les composants qui doivent être déployés dans chaque dispositif. Ayant la description de tous les composants, le gestionnaire de déploiement demande l'adresse de chaque composant auprès du registre de composants (étape 1). Ensuite, le registre retourne au gestionnaire de déploiement les adresses de chacun des composants (étape 2) en se basant sur leurs descriptions et sur le fichier *repository.xml*. A la fin de la deuxième étape, le gestionnaire de déploiement dispose de l'URL de chaque composant, de l'action qui doit être effectuée sur le composant (fournie par le graphe de déploiement), les connexions de ce composant avec les autres composants et de l'adresse IP du dispositif concerné.

Lors de la troisième étape, le gestionnaire de déploiement se connecte à chacun des agents de déploiement en envoyant un message lui indiquant

la liste des actions à effectuer sur chaque composant et les connexions qui doivent être établies avec les autres composants.

Ayant toutes les informations nécessaires sur les composants, chacun des agents de déploiement se charge d'exécuter les actions demandées sur chaque composant. Si une action de déploiement d'un nouveau composant est demandée, l'agent le télécharge via le protocole HTTP auprès du registre de composants et l'installe dans le dispositif (étapes 4 et 5). Après avoir effectué les actions demandées, chacun des agents retourne les identifiants des bundles déployés et leurs états de déploiement (étape 6) afin que le gestionnaire de déploiement puisse produire un fichier qui montre le résultat de déploiement.

Le graphe de déploiement, passé en paramètre pour le gestionnaire de déploiement, est représenté par un fichier GraphML dont la structure est illustrée par la Figure 4.7. Ce graphe doit contenir toutes les informations nécessaires pour exécuter les actions de déploiement dans les dispositifs des utilisateurs. Il consiste en un ensemble de nœuds représentant les dispositifs des utilisateurs.

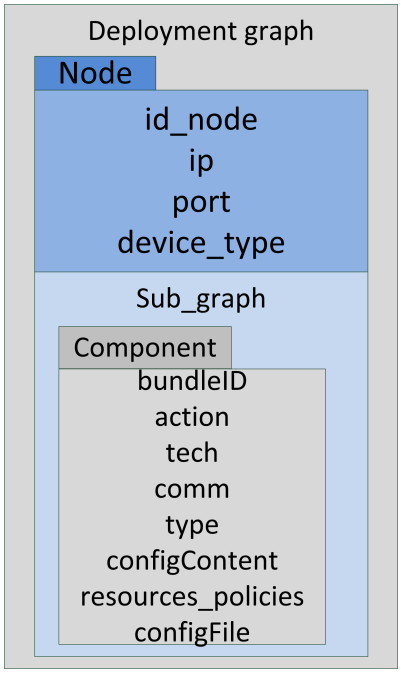

FIGURE 4.7 – *Graphe de déploiement*

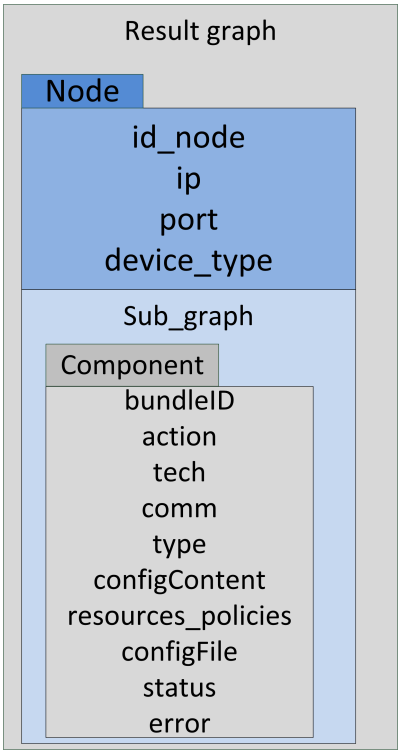

FIGURE 4.8 – *Résultat de déploiement*

Les informations qui sont associées à chaque nœud sont représentées par les attributs suivants : *id_node*, *ip*, *port*, *device_type*. L'attribut *id_node* représente une adresse IP qui identifie le dispositif de l'utilisateur. Les deux attributs *ip* et *port* représentent respectivement l'adresse IP ou est démarré l'agent de déploiement et le numéro de port sur lequel il est en écoute des connexions du gestionnaire de déploiement. L'attribut *device_type* permet d'indiquer le type du dispositif et son système d'exploitation.

Les composants (ou bundles) associés à chaque dispositif sont décrits dans un sous-graphe appelé *Sub_graph*. Ce sous-graphe est composé de

l'ensemble de composants (*Component*) qui doivent être configurés dans le dispositif. Les attributs de chaque composant sont décrits comme suit :

- L'attribut *bundleID* représente l'identifiant du composant. Il est construit d'un triplet $< type, ip, s >$ où $type$, ip et s représente respectivement le type du composant, l'adresse IP du dispositif où il est déployé et la session dans laquelle le composant est impliqué.
- L'attribut *action* représente l'action à effectuer sur le composant. Les actions possibles sont *deployAndStart* pour installer et démarrer le composant, *uninstall* pour désinstaller le composant et *config* pour configurer ou mettre à jour le composant.
- L'attribut *tech* permet d'indiquer la technologie utilisée pour la communication entre les différents composants.
- L'attribut *comm* indique le mode de communication entre les composants. Il présente l'architecture ou le modèle de communication utilisé. Dans le cas du framework de communication, cet attribut peut prendre comme valeur EBC, P2P, etc. Tandis que pour le framework de communication, nous choisissons par défaut l'architecture XACML.
- L'attribut *type* indique le type du composant. Il dépend du mode de communication choisi. Par exemple, le type peut être EP, EC ou CM dans le mode de communication EBC ; et PEP ou PDP dans le mode de communication XACML.
- L'attribut *configContent* indique les connexions qui doivent être établies entre le composant avec les autres composants. Si le composant joue le rôle d'un serveur (CM ou PDP), l'attribut se compose d'une paire $< s, p >$ où s représente la session dans laquelle il gère les messages et p représente le port sur lequel le composant est en écoute. Par contre, si le composant joue le rôle d'un client (PEP, EP ou EC), l'attribut se compose d'un triplet $< s, ip, p >$ où s représente la session, ip représente l'adresse IP du serveur auquel le composant doit se connecter et p représente le port d'écoute du composant serveur.
- L'attribut *configFile* indique le chemin du fichier qui contient les paramètres de connexions avec les autres composants.
- L'attribut *resources_policies* indique l'URL des ressources de la session si le composant est de type PEP ; ou l'URL des politique associées à la session si le composant est de type PDP.

Les arêtes du graphe de déploiement représentent les connexions entre les composants. La Figure 4.9 illustre un exemple d'arête d'un graphe de déploiement. Dans cet exemple, le composant ayant l'identifiant $PEP\_198.162.1.2\_s1$ est relié au composant ayant l'identifiant $PDP\_198.162.1.1\_s1$ par un lien de type push. L'attribut transition de l'arête indique le temps d'attente entre le déploiement du PDP et le déploiement du PEP.

```
<edge source="PEP_198.162.1.2_s1" target="PDP_198.162.1.1_s1">
  <data key="transition">1000</data>
  <data key="attribute1">push</data>
```

FIGURE 4.9 – *Exemple d'arête d'un graphe de déploiement*

Après avoir réalisé toutes les actions, chaque agent de déploiement re-

tourne le résultat de chaque action. Ensuite, le gestionnaire de déploiement se base sur ces résultats afin de produire un fichier GraphML qui contient tous les résultats de déploiement. La structure de ce fichier est illustrée par la Figure 4.8. Dans ce fichier, deux attributs sont rajoutés pour chaque composant : *status* et *error*. Les valeurs possibles de ces attributs sont illustrées par le Tableau 4.1. Pour chaque action, l'attribut *status* peut prendre deux valeurs : une valeur qui indique que l'action s'est bien effectuée telle que "*deployed* and *started*", "*uninstalled*" et "*configured*" ; et une autre valeur qui indique que l'action a échoué telle que "*deployment failed*", "*uninstallation failed*" et "*configuration failed*". Si une action a échoué, l'attribut *error* prend comme valeur l'origine de l'erreur qui a causé le dysfonctionnement.

Action	**valeur de status**	**valeur de error**
deployAndStart	Deployed and started	Null
	Deployment failed	Origine de l'erreur
Uninstall	Uninstalled	Null
	Uninstallation failed	Origine de l'erreur
Config	Configured	Null
	Configuration failed	Origine de l'erreur

Tab. 4.1 – *Valeurs possibles des attributs status et error*

Les graphes de déploiement (voir Figure 3.14 et Figure 4.3) générés par le moteur de transformation de graphe GMTE sont composés d'un ensemble de nœuds qui représentent les composants. Cette structure ne contient pas toutes les informations nécessaires telles que les actions de déploiement et les interconnexions entre les composants. En plus, elle ne peut pas être traitée par le gestionnaire de déploiement. Par conséquent, le graphe de déploiement, généré par GMTE, doit être transformé en un fichier GraphML qui respecte la structure présentée dans la Figure 4.7. Dans la suite de ce manuscrit, nous utilisons le terme "fichier de déploiement" pour indiquer le graphe de déploiement final après la transformation du graphe de déploiement généré par le GMTE. Un exemple de transformation est illustré par les Figure 4.10 et Figure 4.11. Nous avons choisi de représenter seulement deux composants à déployer afin de simplifier le fichier GraphML.

Le graphe de déploiement, illustré par la Figure 4.10, est constitué d'un composant $pep1$ de type PEP et d'un autre composant $pdp1$ de type PDP. Les deux composants sont déployés dans le même dispositif identifiée par l'adresse IP 192.168.1.2. Les nœuds du graphe représentent les composants $pep1$ et $pdp1$ à déployer. Ce graphe doit être transformé en un fichier GraphML, illustré par la Figure 4.11. La structure de ce graphe est conforme à la structure présentée dans la Figure 4.7. Il est composé d'un seul nœud qui représente le dispositif identifié par l'adresse IP 192.168.1.2. Les informations associées au dispositif telles que l'adresse IP, le port d'écoute de l'agent de déploiement et le type du dispositif sont représentées par les attributs ip, $port$ et $device\_type$. Ce nœud contient un sous-graphe, ayant comme identifiant l'adresse IP du dispositif suivi de deux point (192.168.1.2 :), qui représente l'ensemble des composants ($pep1$ et $pdp1$) à déployer. Le premier composant est identifié par le bundle ID $pep\_192.168.1.2\_s1$ et doit être déployé et démarré dans le dispositif. L'attribut $configContent$ indique qu'il doit être connecté au composant qui est en écoute sur le port 4501 du même dispo-

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
    <graph id="depl_graph1" edgedefault="directed">
    <node id="pep1">
      <data key="type">PEP</data>
      <data key="session">session1</data>
      <data key="ip">198.162.1.2</data>
    </node>
    <node id="pdp1">
      <data key="type">PDP</data>
      <data key="session">session1</data>
      <data key="ip">198.162.1.2</data>
    </node>
    <edge source="pep1" target="pdp1">
      <data key="attribute1">push</data>
    </edge>
  </graph>
</graphml>
```

FIGURE 4.10 – *Exemple de graphe de déploiement généré par GMTE*

sitif. Le deuxième composant, identifié par le bundle ID $pdp\_192.168.1.2\_s1$, doit être également déployé et démarré dans le même dispositif et être à l'écoute des connexions des composants de type PEP sur le port 4501. Les autres attributs indiquent que les deux composants communiquent en sockets et suivent le modèle de l'architecture XACML. En plus des deux composants, le fichier de déploiement contient une arrête, représentée par la balise $< edge >$, qui permet de relier le composant de type PEP au composant de type PDP.

Nous décrivons dans la section 4.2.4 le processus de transformation d'un graphe généré par GMTE en un fichier de déploiement qui peut être passé en paramètre pour le gestionnaire de déploiement.

4.2.3 Politique de sensibilité aux résultats de déploiement

Dans les environnements collaboratifs où les dispositifs des utilisateurs ont des capacités limitées, le framework de contrôle d'accès utilise la grammaire de graphe illustrée dans le Tableau 3.1 afin de générer tous les graphes de déploiement possibles. Ensuite, la fonction de sélection est utilisée afin de sélectionner le graphe de déploiement le plus adapté aux ressources disponibles dans chaque dispositif.

Comme nous l'avons mentionné dans la sous-section 4.2.2, le gestionnaire de déploiement retourne le résultat de déploiement de chacun des composants dans tous les dispositifs. Les deux attributs *status* et *error* indiquent l'état de déploiement et l'origine de l'erreur si un échec de déploiement a eu lieu.

Tenir compte des valeurs de ces deux attributs est nécessaire lors du prochain processus de déploiement. En effet, les composants de communication et de contrôle d'accès peuvent être désinstallés, reconfigurés ou réinstallés

```
<graphml>
  <graph id="depl_graph1" edgedefault="directed">
    <node id="192.168.1.2">
      <data key="ip">192.168.1.2</data>
      <data key="port">5000</data>
      <data key="device_type">android</data>
      <graph id="192.168.1.2:" edgedefault="directed">
        <node id="192.168.1.2::pep_192.168.1.2_s1">
          <data key="bundleID">pep_192.168.1.2_s1</data>
          <data key="action">deployAndStart</data>
          <data key="tech">SOCKET</data>
          <data key="comm">XACML</data>
          <data key="type">PEP</data>
          <data key="configContent">client.session=s1;client.ip=192.168.1.2;client.port=4501</data>
          <data key="resources_policies">URL_resources</data>
          <data key="configFile">./config/config</data>
        </node>
        <node id="192.168.1.2::pdp_192.168.1.2_s1">
          <data key="bundleID">pdp_192.168.1.2_s1</data>
          <data key="action">deployAndStart</data>
          <data key="tech">SOCKET</data>
          <data key="comm">XACML</data>
          <data key="type">PDP</data>
          <data key="configContent">server.session=s1;server.port=4501</data>
          <data key="resources_policies">URL_policies</data>
          <data key="configFile">./config/config</data>
        </node>
      </graph>
    </node>
    <edge source="pep_192.168.1.2_s1" target="pdp_192.168.1.2_s1">
      <data key="transition">1000</data>
      <data key="attribute1"></data>
    </edge>
  </graph>
</graphml>
```

FIGURE 4.11 – *Graphe de déploiement après transformation*

en fonction des besoins de l'environnement collaboratif. Par exemple, supposons que :

- Le gestionnaire de déploiement envoie une action de déploiement d'un composant $c1$ dans un dispositif $d1$,
- L'installation de $c1$ dans $d1$ a échoué lors de la dernière action de déploiement.

Dans ce cas l'action d'installation de $c1$ dans $d1$ doit être remplacée par une autre action afin d'éviter un autre échec de déploiement qui pourra causer un dysfonctionnement de l'application. Par exemple, si l'installation d'un composant $pdp1$, qui contrôle l'accès aux ressources dans une session $s1$, dans un dispositif $d1$ a échoué à un instant t ; et un autre composant $pdp2$, qui contrôle l'accès dans une session $s2$, doit être installé dans le même dispositif $d1$, alors il y a une grande chance que l'installation de $pdp2$ aboutit à un échec. En conséquence, une politique de sensibilité aux résultats de déploiement doit être définie afin d'éviter au maximum ce type de dysfonctionnements.

Pour cela, nous définissons une politique, appelée *DeploymentAware()*, qui est illustrée par la Figure 4.12. Elle permet d'assurer la sensibilité de la fonction de sélection aux résultats de déploiement. Malgré que les échecs de déploiement soient peu fréquents, nous mettons en place cette politique afin d'éviter les dysfonctionnements de l'application causés par le déploie-

DeploymentAware (A_q) {
Soit $\{c_1, \ldots, c_n\}$ l'ensemble de composants à déployer de A_q
Soit $D = \{D_i i \in [1..N]$, dispositif où le déploiement de c_i a échoué$\}$
Soit $red = 0$
Pour chaque $c_i \in A_q$
Soit d_i^q où doit être déployé le composant c_i
Si $d_i^q \in D$ alors red = red + 1
fin pour
retourner red
}

FIGURE 4.12 – *Politique DeploymentAware()*

ment des composants dans les environnements hétérogènes dans lesquels différents types de dispositifs interviennent dans la collaboration.

La politique *DéploymentAware()* affecte au graphe de déploiement, passé en paramètre et généré par le GMTE, une valeur qui représente le nombre de composants c_i dont le déploiement a abouti à un échec lors de l'action de déploiement précédente. L'objectif de l'utilisation de cette politique est de minimiser le nombre de redéploiements de composants dont le déploiement précédent n'a pas réussi. Cela permet d'éviter l'installation d'un composant dans un dispositif suite à l'échec de l'installation de ce composant dans le même dispositif.

Comme le montre la Figure 4.12, chaque graphe de déploiement est représenté par A_q. En premier lieu, la politique parcourt le fichier qui décrit les résultats de déploiement et détermine les composants qui n'ont pas été bien déployés et les identifiants des dispositifs associés. Ensuite, elle parcourt tous les composants du graphe de déploiement. Si le composant c_i doit être déployé dans un dispositif d_i^q et le déploiement de c_i dans d_i^q a échoué lors du dernier processus de déploiement, alors la valeur de la variable *red* est incrémentée. Après avoir calculé la valeur *red* pour tous les graphes de déploiement, la fonction de sélection choisit le graphe ayant la valeur la plus petite. Nous expliquons dans la sous-section suivante comment ce graphe sera traité pour générer la structure détaillée dans la Figure 4.7.

4.2.4 Génération du graphe de déploiement et problème de monotonicité

Comme nous l'avons mentionné dans la sous section 3.2.6, nous ne pouvons pas mettre à jour l'instance d'une ontologie avec OWL/SWRL à cause de la nature monotone d'OWL. Cela nous pose deux problèmes : un problème de performance au niveau de l'utilisation du GMTE et un problème d'incohérence des graphes de déploiement qui sont utilisés par le gestionnaire de déploiement. Nous expliquons dans la suite de cette sous-section ces deux problèmes et les solutions que nous proposons pour les résoudre.

Utilisation du GMTE dans le processus d'adaptation

Avec OWL et SWRL, le raisonnement est effectué à chaque fois l'état de la collaboration change afin de générer les graphes de contrôle d'accès et

les graphes de collaboration. Ensuite, ces graphes sont passés en paramètre pour le GMTE pour la génération d'un graphe de déploiement initial qui décrit tous les composants à déployer. La nature monotone d'OWL nous oblige à passer en paramètre toute la nouvelle instance de l'ontologie au GMTE sans tenir compte des traitements qui ont été effectués précédemment. Le fait d'effectuer des transformations par le GMTE sur toute l'instance même après un changement mineur de l'état de collaboration peut diminuer les performances des processus d'adaptation.

Pour remédier à ce problème, nous avons besoin de solutions pour optimiser l'utilisation du GMTE dans chaque processus d'adaptation. L'application des règles de raisonnement sur les instances des ontologies de domaine permet de générer des graphes qui traduisent les besoins de contrôle d'accès et de collaboration dans chaque session. Le graphe contient un ensemble d'éléments associés à chaque session. Par conséquent, chaque graphe peut être divisé en un ou plusieurs graphes qui représentent chacun des éléments spécifiques à une session donnée. Puisque ces graphes sont transformés avec GMTE en graphes de déploiement, nous proposons de diviser les graphes de contrôle d'accès et de collaboration en sous graphes et de n'appliquer les règles de transformation que sur les graphes qui ont changé par rapport au processus de déploiement précédent. Cela peut se faire en utilisant des caches qui enregistrent le graphe de déploiement associé à chaque session et ne le mettre à jour que si des changements ont eu lieu dans la session. Pour détecter les changements au niveau de chaque session, nous utilisons la fonction *ModifiedSessions()* illustrée par la Figure 4.13. Cette fonction permet de retourner la liste des sessions qui ont été modifiées (MS) à un instant t et qui requiert l'appel à GMTE afin de générer le nouveau graphe de déploiement qui y est associé. Pour cela, la fonction se base sur la partie de l'instance actuelle (CG) et celle générée durant le processus d'adaptation précédent (OG) associée à chaque session afin de détecter les changements. La fonction commence par déterminer les parties de l'instance qui ont été supprimées et celles qui ont été mises à jour. Ensuite, elle détermine l'ensemble des sessions qui ont été rajoutées à l'instant t ainsi que les éléments associés.

Ensuite, le GMTE est utilisé pour appliquer la grammaire de graphe à l'ensemble des graphes associés aux sessions modifiées. Ceci est illustré par la fonction *ApplyAndConcatenate()* dans la Figure 4.14. Cette fonction permet d'appliquer l'ensemble de règles de la grammaire seulement aux graphes modifiés. Le résultat est ensuite stocké dans une liste R qui contient les graphes de déploiement associés à toutes les sessions. Afin de rassembler tous les graphes de déploiement dans un même graphe, la fonction *concatenate()* est utilisée. Dans le cas où la grammaire de graphe génère un ensemble de graphe de déploiement pour chaque session, la fonction de concaténation permet de calculer le produit cartésien de l'ensemble de graphes associés à toutes les sessions. Ce graphe de déploiement doit être transformé en un fichier GraphML pour pouvoir le passer en paramètre pour le gestionnaire de déploiement. Nous présentons toutes les étapes de transformation dans la sous-section suivante.

ModifiedSessions() {
Soit $CS = \{s_1, .., s_N\}$ la liste des sessions actuelles
Soit $OS = \{s_1, .., s_P\}$ la liste des sessions à un l'instant t-1
Soit $CG = \{CGi, i \in [1..N]$, la partie de l'instance relative à S_i de $CS\}$
Soit $OG = \{OG_i, i \in [1..P]$, la partie de l'instance relative à S_i de $OS\}$
Soit MS la liste des sessions modifiées
Pour chaque i ∈ [1..P]
    Si $OG_i \notin CG$, alors ajouter (MS, OGi)
    Si non alors
        comparer (OG_i, CG_i)
        Si $OG_i \neq CG_i$, alors ajouter (MS, CG_i)
fin pour
Pour chaque$i \in [1..N]$
    Si $CG_i \notin OG$, alors ajouter (MS, CG_i)
fin pour
retourner MS
}

FIGURE 4.13 – *La fonction ModifiedSessions()*

ApplyAndConcatenate(){
Soit $MS = \{MSi, i \in [1..N]$, un graphe modifié$\}$
Soit R la liste des graphes de déploiement pour toutes les sessions
Soit *rules* l'ensemble de règles à appliquer à MS
Pour chaque $i \in [1..N]$
Soit R_i le résultat de transformation de MS_i par GMTE
$R_i = apply(rules, MS_i)$
$ajouter(R_i, R)$
fin pour
$concatenate(R_1, .., R_N)$
}

FIGURE 4.14 – *La fonction ApplyAndConcatenate()*

Génération du fichier de déploiement

A cause de la nature monotone d'OWL, les processus d'adaptation s'effectuent de la façon suivante : après chaque changement au niveau de l'état de la collaboration, une instance de l'ontologie de domaine est créée et utilisée pour le déclenchement d'un processus d'adaptation. Ces processus sont indépendants et génèrent chacun un fichier de déploiement. Le problème de cette approche est que les graphes de déploiement sont indépendants les uns des autres. Par exemple, si deux composants $pdp1$ et $pdp2$ doivent être déployés à un instant t afin de contrôler l'accès aux ressources respectivement dans les deux sessions $s1$ et $s2$, alors un fichier de déploiement est généré afin de demander le déploiement de ces deux composants. Si à un instant $t+1$, une nouvelle session $s3$ est établie, un autre composant $pdp3$ doit être déployé afin de contrôler l'accès aux ressources dans cette session. Vu que les processus d'adaptation sont indépendants, le fichier de déploiement généré à l'instant $t+1$ décrit le besoin de déployer le composant $pdp3$, mais

aussi les deux autres composants *pdp*1 et *pdp*2. L'indépendance des fichiers de déploiement cause des redondances qui peuvent causer des dysfonctionnements. En effet, redéployer un composant qui est déjà déployé est inutile et peut causer des conflits tels que l'utilisation du même port d'écoute dans un dispositif. Afin de résoudre ce problème, nous proposons un ensemble d'actions à effectuer avant de faire appel au gestionnaire de déploiement.

Le diagramme d'activité, illustré par la Figure 4.15, décrit les actions qui permettent de générer un fichier de déploiement en GraphML tout en tenant compte de l'état actuel des composants qui sont déjà déployés. Ces actions, effectuées par une entité appelée *GraphTransformer*, sont appliquées à un graphe généré par le GMTE ou suite à l'appel à la fonction de sélection si nous choisissons de générer un ensemble de graphe de déploiement. La Figure 4.10 et la Figure 4.11 représente respectivement un exemple de déploiement initial et un graphe de déploiement final (ou fichier de déploiement) après application de toutes les actions.

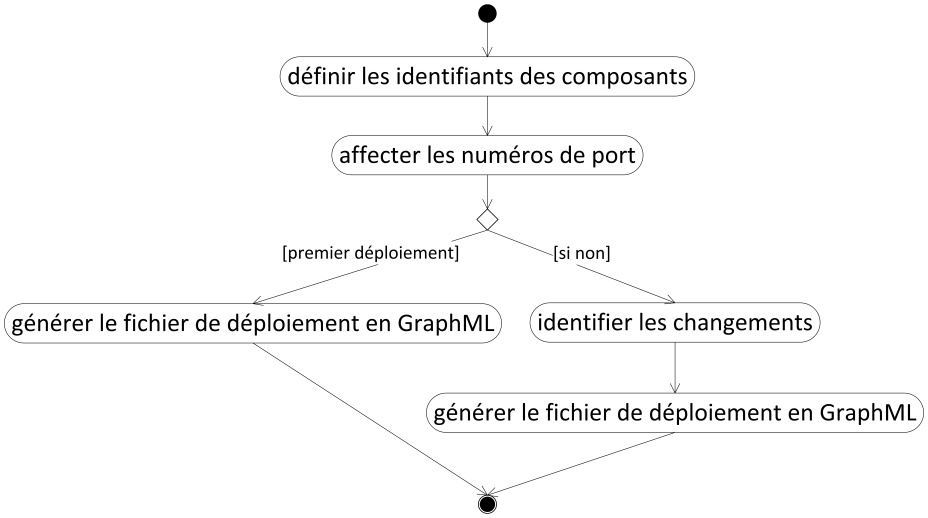

FIGURE 4.15 – *Diagramme d'activité pour la génération du fichier de déploiement*

La première action consiste à affecter pour chaque composant un identifiant qui représente la valeur de l'attribut *bundleID* du graphe de déploiement. Chaque identifiant se compose du type du composant, de l'adresse IP du dispositif où il est déployé et de la session dans laquelle il est impliqué. Ces trois informations existent dans le graphe généré par le GMTE. La deuxième action consiste à affecter un numéro de port pour chaque session. Ces numéros de port représentent le port d'écoute des composants qui jouent le rôle de serveur tel que les composants de type PDP et ceux de type CM. Les composants de type PDP seront à l'écoute des composants de type PEP, tandis que les composants de type CM seront à l'écoute des composants de type EP et EC. Les actions suivantes dépendent du nombre de déploiements qui ont été effectués. Deux cas sont possibles :

Premier déploiement
S'il s'agit du premier déploiement à effectuer, c'est-à-dire c'est le premier processus d'adaptation qui a eu lieu, alors le fichier GraphML qui représente le graphe de déploiement est créé comme suit.

1. D'abord, le fichier GraphML doit être créé et initialisé avec un espace de nommage et un ensemble d'attributs. Ceci est illustré par la Figure 4.16. L'avantage de GraphML est qu'il permet la création d'un ensemble d'attributs qui servent à décrire chaque nœud (les dispositifs et les composants) du graphe. Ces attributs sont définis par des éléments, appelés *key*, caractérisés par un identifiant, un nom, un type et un domaine. L'identifiant est représenté par l'attribut XML *id* et utilisé pour se référer à l'attribut dans le document GraphML. Le nom de l'attribut doit être unique, il est représenté par l'attribut *attr.name* et permet d'identifier la signification de l'attribut. Le type de l'attribut peut être l'un des types définis pour le langage de programmation Java tel que *boolean*, *int*, *long*, *string* et *double*. Le domaine, représenté par l'attribut *for*, indique pour quel élément du graphe (graph, node, edge) l'attribut peut être déclaré. Nous avons décrit les attributs que nous utilisons dans le graphe de déploiement dans la sous section 4.2.2. Tous les attributs ont comme domaine les nœuds du graphe qui peuvent représenter les dispositifs ou bien les composants. Tous les attributs sont de type *string* sauf le numéro de port est de type *int*.

```
<graphml xmlns:xsi="http://www.w3.org.2001/XMLSchema-instance"
xmlns:schemaLocation="http://graphml.graphdrawing.org/xmlns
http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <key id="ip" for="node" attr.name="ip" attr.type="string"/>
  <key id="port" for="node" attr.name="port" attr.type="int"/>
  <key id="device_type" for="node" attr.name="device" attr.type="string"/>
  <key id="bundleID" for="node" attr.name="bundleID" attr.type="string"/>
  <key id="action" for="node" attr.name="action" attr.type="string"/>
  <key id="tech" for="node" attr.name="tech" attr.type="string"/>
  <key id="comm" for="node" attr.name="comm" attr.type="string"/>
  <key id="type" for="node" attr.name="type" attr.type="string"/>
  <key id="configContent" for="node" attr.name="configContent" attr.type="string"/>
  <key id="configFile" for="node" attr.name="configFile" attr.type="string"/>
  <key id="transition" for="node" attr.name="transition" attr.type="string"/>
  <key id="attribute1" for="node" attr.name="attribute1" attr.type="string"/>
```

FIGURE 4.16 – *Initialisation du graphe de déploiement*

2. Après l'étape d'initialisation du graphe, l'entité *GraphTransformer* doit identifier tous les nœuds du graphe de déploiement. Ces nœuds représentent les dispositifs et sont identifiés par leurs adresses IP. Ensuite, pour chaque dispositif il faut :
   - Affecter trois valeurs aux trois attributs *ip*, *port*, *device_type*.
   - Identifier les composants à déployer dans le dispositif et les attributs de chacun. Lors du premier déploiement, l'attribut *action* prend toujours la valeur *deployAndStart*. Les numéros de port qui sont déjà affectés lors de l'étape précédente sont utilisés afin de définir la valeur de l'attribut *configContent* de chaque composant. Un exemple de ce graphe de déploiement est illustré par la Figure 4.11.

A partir du deuxième déploiement

A partir du deuxième déploiement, les fichiers de déploiement doivent être cohérents et doivent tenir compte des actions effectuées lors des processus de déploiement précédents. Puisque chaque processus d'adaptation génère indépendamment un fichier de déploiement, il faut, d'une part, identifier à chaque fois les changements qui ont eu lieu dans le graphe de déploiement,

et d'autre part, générer le fichiers de déploiement en GraphML en se basant sur les changements qui ont eu lieu.

1. Afin de pouvoir identifier tous les changements, l'entité *GraphTransformer* utilise une fonction *Delta()* (Figure 4.17) afin de retourner trois listes : la liste des composants à déployer (A), la liste des composants à désinstaller (R) et la liste des composants reconfigurer (RC). Ceci est fait en comparons les composants de l'ancien graphe avec ceux du nouveau graphe. La fonction *Delta()* prend comme paramètre un graphe de déploiement généré par GMTE (appelé CG) et se base sur le graphe de déploiement précédent appelé OG. Cette fonction commence par déterminer les composants qui existent dans le graphe actuel et qui n'existent pas dans le graphe de déploiement précédents. Ces composants seront ajoutés à liste A. Ensuite, elle ajoute à la liste R les composants qui existent dans OG et qui n'existent pas dans CG. La dernière étape consiste à ajouter à la liste RC les composants à reconfigurer. Ces composants sont des composants clients tels que les PEP, les EP et les EC et doivent être reconfigurés suite au changement de l'adresse des composants serveur tels que les PDP et les CM.

```
Delta(CG) {
    Soient A la liste des composants à déployer
    R la liste des composants à désinstaller
    RC la liste des composants à reconfigurer
    Soit OG le graphe de déploiement précédent
    Soit C la liste des types de composants client
    Soit N = card(CG)
    Soit P = card(OG)
    Soit c_i, i ∈ [1..N] un composant de CG
    Soit o_j, j ∈ [1..P] un composant de OG
    Pour chaque i ∈ [1..N]
        Si c_i ∉ OG, alors ajouter(A, c_i)
    fin pour
    Pour chaque j ∈ [1..P]
        Si o_j ∉ CG, alors ajouter(R, o_j)
    fin pour
    Pour chaque i ∈ [1..N]
        Si c_i ∉ A et c_i ∉ R, alors
            Si c_i ∈ C
            Soit adr_old l'adresse du serveur dans OG
            Soit adr_new l'adresse du serveur dans CG
            Si adr_old ≠ adr_new, alors ajouter(RC, c_i)
    fin pour
}
```

FIGURE 4.17 – *La Fonction Delta()*

2. Après avoir exprimé tous les changements dans les trois listes, le *GraphTransformer* génère le fichier GraphML du graphe de déploiement final. Tout d'abord, il commence par la création et l'initialisation du fichier comme c'est illustré par la Figure 4.16. Ensuite, il met à jour la

liste des dispositifs qui doivent être présents dans le fichier. Enfin, pour chaque dispositif, il :
- affecte trois valeurs aux trois attributs *ip*, *port*, *device_type*.
- identifie les actions à effectuer sur chaque composant. Pour cela, le *GraphTransformer* parcourt la liste des composants à déployer (A), ajoute le composant s'il doit être déployé dans le dispositif et affecte la valeur *deployAndStart* à l'attribut *action*. Ensuite, il parcourt la liste des composants à désinstaller (R), ajoute le composant s'il doit être désinstallé du dispositif et affecte la valeur *uninstall* pour l'attribut *action*. Finalement, il parcourt la liste des composants à reconfigurer, ajoute le composant s'il est déployé dans le dispositif et affecte la valeur *config* à l'attribut *action*. Comme nous l'avons mentionné, les composants clients peuvent être reconfigurés suite au changement de l'adresse IP du composant serveur auquel ils sont connectés. Pour cela l'attribut *configContent* doit être mis à jour avec la nouvelle adresse du composant serveur.

Après la génération de chaque fichier GraphML du graphe de déploiement, le gestionnaire de déploiement se charge d'effectuer les actions planifiées pour chaque composant.

4.2.5 Déclenchement des processus d'adaptation

Suite aux changements du contexte, le processus d'adaptation permet de générer un fichier de déploiement qui répond aux besoins de l'application. Ces changements sont exprimés par une instance de l'ontologie de domaine. Nous proposons de lancer chaque processus d'adaptation dans un thread séparé. Un processus d'adaptation commence par la récupération de l'instance de l'ontologie et finit par le déploiement physique des composants. L'utilisation des threads nous permet de lancer des processus d'adaptation en parallèle et de les suspendre en fonction des besoins.

Dans les environnements ubiquitaires, les changements sont très fréquents et les utilisateurs peuvent changer de rôle ou de groupe à chaque instant de la collaboration. Ceci peut causer des problèmes d'incohérence au niveau de déploiement des composants. En effet, après chaque changement qui nécessite une reconfiguration des composants, deux actions sont possibles :
- Arrêter le processus d'adaptation en cours et déclencher un nouveau processus : dans ce cas, le framework peut rester infiniment sans effectuer des actions de déploiement si les changements sont très fréquents.
- Déclencher un nouveau processus d'adaptation en parallèle avec le processus en cours : dans ce cas les deux processus génèrent deux ou plusieurs fichiers de déploiement. Cela peut causer des incohérences puisque le processus d'adaptation lancé à l'instant t peut se terminer après celui qui est lancé à l'instant $t+1$. Ce problème d'incohérence peut être résolu en affectant des priorités plus élevés pour les processus les moins récents. Ceci permet de s'assurer que le dernier déploiement correspond bien au dernier changement. Cependant, dans un environnement où les changements sont fréquents, l'application doit attendre l'exécution de tous les processus pour assurer la cohérence à un instant t.

Une première solution consiste à attendre la terminaison du processus d'adaptation en cours et lancer le nouveau processus. Le problème de cette solution est que si au cours de l'exécution du processus en cours plusieurs changements ont eu lieu, le résultat de déploiement du processus en cours ne répondra plus aux besoins de l'application. Par conséquent, nous n'adoptons pas cette solution.

Afin de remédier à ce problème, nous proposons un diagramme d'états transitions pour le gestionnaire du framework qui gère les graphes et les achemine vers la bonne entité. Dans le framework de contrôle d'accès, le gestionnaire est représenté par la classe *Handler*. Le diagramme d'états transitions, illustré par la Figure 4.18, permet de décider à quel instant le gestionnaire doit déclencher un processus d'adaptation même si plusieurs changements ont lieu au cours du processus d'adaptation en cours. Pour cela nous définissons un nombre entier *max*. Si le nombre de changement qui ont eu lieu au cours du processus d'adaptation en cours atteint le nombre *max*, le gestionnaire doit arrêter le processus en cours et lancer un nouveau processus afin de tenir compte des nouveaux changements. Après chaque changement :

- Si le gestionnaire est en état inactif, alors il lance un processus d'adaptation et passe à l'état actif.
- Si le gestionnaire est en état actif, alors :
  - Si le nombre de changements qui ont eu lieu est supérieur à *max*, alors il reste en état actif mais lance un nouveau processus d'adaptation pour tenir compte des nouveaux changements.
  - Si le nombre de changements est inférieur au nombre *max*, alors rien ne se produit et le processus d'adaptation continue son exécution. Quand le processus se termine, le gestionnaire passe à l'état inactif.

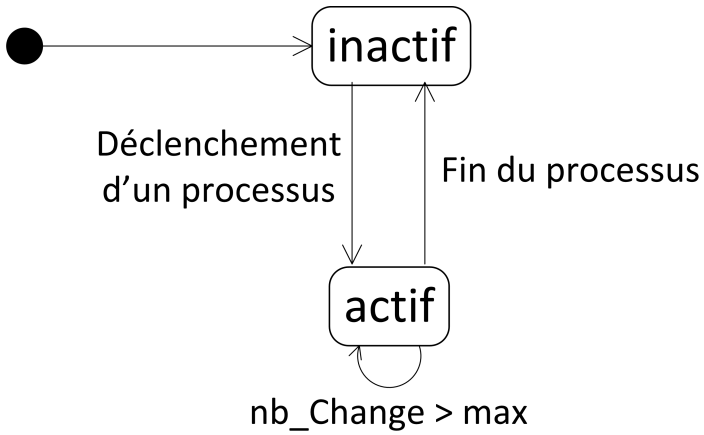

FIGURE 4.18 – *Diagramme d'états transitions*

Le nombre *max* dépend de la nature de l'environnement. Il peut avoir la valeur 1 dans les environnements peu dynamiques où les changements sont peu fréquents. Ce nombre augmente en fonction de la fréquence des changements dans les activités collaboratives.

Conclusion du chapitre

Dans ce chapitre, nous avons présenté notre méthodologie pour le déploiement dynamique des composants logiciels dans les environnements collaboratifs ubiquitaires. Nous avons détaillé toutes les étapes qui doivent être suivies par les systèmes collaboratifs qui ont besoin de déployer et reconfigurer des composants en cours de l'exécution.

En premier lieu, nous avons présenté le cadre dans lequel nous appliquons le déploiement. Nous avons décrit le framework de communication et le framework de contrôle d'accès et nous avons expliqué à quelles étapes d'exécution des deux framework nous nous intervenons pour la mise en place du déploiement.

Ensuite, nous avons présenté la technologie OSGi et le processus de déploiement qui permet la gestion du cycle de vie des composants logiciels en se basant sur des graphes de déploiement fournis par les deux frameworks au cours de l'activité collaborative. Le processus de déploiement que nous avons proposé est générique et peut être utilisé par plusieurs applications qui doivent juste respecter la structure du fichier de déploiement. Un des avantages du processus de déploiement est qu'il retourne à chaque fois les résultats de déploiement. Ceci nous a permis de définir une politique qui permet de tenir compte des résultats de déploiement précédents afin de minimiser les échecs de déploiement des composants.

Afin de résoudre les problèmes causés par la nature monotone d'OWL, nous avons proposé des algorithmes afin d'optimiser l'utilisation du GMTE et générer un fichier de déploiement cohérent qui tient compte des composants déjà déployés dans les dispositifs.

Dans le chapitre suivant, nous présentons un exemple d'application afin d'illustrer l'utilisation du framework de communication et du framework de contrôle d'accès dans les environnements collaboratifs ubiquitaires.

Expérimentations

5

Sommaire

Dans ce chapitre, nous présentons les expérimentations que nous avons réalisées afin de valider le fonctionnement du framework de contrôle d'accès et des techniques de déploiement et de mesurer les performances. Nous commençons ce chapitre par présenter les différents outils et bibliothèques que nous avons utilisés dans notre implantation. Ensuite, nous présentons le projet Galaxy et l'application prototype qui nous a servi pour la réalisation des expérimentations. Ensuite, nous utilisons cette application pour effectuer une évaluation fonctionnelle et une évaluation des performances. Dans l'évaluation fonctionnelle, nous présentons un scénario sur lequel nous nous sommes basés pour étudier les différentes étapes du processus d'adaptation ainsi que le mécanisme de déploiement des composants de contrôle d'accès et de communication. Dans l'évaluation des performances, nous commençons par étudier le temps de réponse du processus d'adaptation en fonction du nombre de participants. Ensuite, nous étudions le temps de génération des décisions d'autorisation dans l'architecture décentralisée que nous avons proposée.

5.1 Outils utilisés

Nous présentons dans cette section les outils que nous avons choisis pour notre implantation. Tout d'abord, nous présentons les outils de traitement des ontologies et des règles SWRL. Ensuite, nous présentons les outils de traitement des politiques de contrôle d'accès.

5.1.1 Outils pour le traitement des ontologies

Il existe de nombreux outils pour la gestion des ontologies OWL et des règles SWRL. On dispose de deux types d'outils :

- Les outils utilisés lors de la phase de conception tels que les éditeurs des ontologies.
- Les outils utilisés lors de la phase d'exécution tels que les bibliothèques ou interfaces de programmation (APIs).

Avant de présenter les différents outils disponibles, nous discutons les versions OWL 1 et OWL 2 du langage OWL. La version OWL 1, apparue en 2004, est la version la plus utilisée par les concepteurs d'applications. Elle permet la représentation des concepts, des propriétés et l'expression de toutes les fonctionnalités présentées dans la section 3.2.1 telles que les restrictions et l'importation des ontologies. La version OWL 2, la dernière version du langage OWL, est devenue une recommandation W3C depuis octobre 2009. Elle est compatible avec OWL 1. En effet, toute ontologie OWL 1 exprimée avec une syntaxe RDF est une ontologie OWL 2 valide. La version OWL 2 ajoute de nouvelles caractéristiques. En effet, elle permet une plus grande expressivité des propriétés, fournit des raccourcis syntaxiques pour l'expression des classes disjointes, permet l'identification d'un individu d'une classe en fonction des valeurs de ses propriétés, etc. Elle définit également la notion de profils qui sont des sous langages qui facilitent l'implantation des ontologies et réduisent la complexité d'inférence et de raisonnement.

Édition des ontologies

Protégé [1] est l'éditeur open source le plus répandu pour la création et l'édition des ontologies OWL. Il a été développé par l'université de Stanford en collaboration avec l'université de Manchester. Son code source, écrit en Java, et extensible avec des plugins, ce qui assure une flexibilité pour le développement des applications. Protégé est soutenu par une grande communauté d'utilisateurs et de développeurs qui fournissent la documentation et contribuent au développement des plugins. Il existe de nombreux plugins dans Protégé tels que le plugin *Jambalaya* pour la visualisation des ontologies et le plugin *SWRLTab* pour l'édition des règles. Les versions 3.x ont été conçues pour l'édition des ontologies de la version OWL 1. A partir de la version 4.0, l'éditeur est conforme avec la version OWL 2.
Il existe d'autres éditeurs d'ontologie tels que Swoop [2], OWLGrEd [3], NeOnToolkit [4], KAON2 [5], etc.

1. http://protege.stanford.edu/
2. http://www.mindswap.org/2004/SWOOP/
3. http://owlgred.lumii.lv/
4. http://neon-toolkit.org/
5. http://kaon2.semanticweb.org/

APIs pour le traitement des ontologies OWL

Les interfaces de programmation (APIs) ou bibliothèques permettent l'accès et la gestion par programme des ontologies OWL. La plupart de ces bibliothèques sont développées en Java. Dans ce qui suit, nous présentons les trois APIs : Jena API [6], OWL API [7] et Protégé-OWL API [8] .

Jena API est principalement conçu pour le traitement des modèles RDF qui représentent des modèles de triplets pour décrire et connecter des ressources anonymes ou identifiées par leurs IRIs dans un graphe orienté. Le RDFS (*Resource Description Framework Schema*), basé sur RDF, est un langage d'ontologie qui permet de créer des hiérarchies simples de concepts et de propriétés. OWL est beaucoup plus riche que RDFS et permet de réaliser tout ce qui peut être fait avec RDFS. Jena API ne supporte pas totalement les deux versions d'OWL. En plus, le raisonneur fourni est incomplet et ne peut pas inférer tout ce qui doit être inféré dans les ontologies OWL.

La bibliothèque OWL API, développée en Java, est l'implémentation de référence pour la création, la manipulation et la sérialisation des ontologies OWL. Elle supporte pleinement les deux versions d'OWL. L'utilisateur d'OWL API n'est pas obligé de maîtriser la syntaxe RDF. En effet, cette API traite l'ontologie comme étant un ensemble d'axiomes et pas comme étant un ensemble de triplets RDF.

Protégé-OWL API est une bibliothèque Java libre pour OWL et RDF. Elle fournit des classes et des méthodes pour charger et enregistrer des fichiers OWL, pour interroger et manipuler des modèles de données OWL et pour effectuer du raisonnement. Elle est en plus optimisée pour le développement d'interfaces graphiques. Elle permet également le développement de plugins dans l'éditeur Protégé ou même dans l'IDE Eclipse.

Outils d'inférence

Ces outils permettent de déduire les connaissances implicites contenues dans une ontologie. Ces outils, appelés souvent moteurs d'inférence, peuvent être intégrés dans un éditeur d'ontologie tel que Protégé ou appelés à travers des APIs. Il existe plusieurs moteurs d'inférence libres tels que Pellet [9], Racer [10] , Fact++ [11] et Hermit [12]. Parmi ces outils, Pellet et le plus répandu grâce à ses performances pour effectuer des inférences pour les ontologies OWL-DL. Il offre également un service pour vérifier la consistance de la base de connaissances avant d'effectuer les inférences.

Outils pour le traitement des règles SWRL

La plupart des moteurs d'inférence tels que Pellet et Hermit permettent d'exécuter des règles SWRL sur une instance d'une ontologie. Par exemple, Pellet interprète les règles SWRL en utilisant la notion de règles DL-sûre qui indique que les règles ne sont appliquées que sur les individus nommés de l'ontologie. Cependant, il ne supporte pas tous les *buit-ins* du langage OWL.

6. http://jena.apache.org/
7. http://owlapi.sourceforge.net/
8. http://protege.stanford.edu/
9. http://clarkparsia.com/pellet/
10. http://racer.sts.tuhh.de/
11. http://owl.man.ac.uk/factplusplus/
12. http://hermit-reasoner.com/

L'utilisation du moteur de règles Jess [13] est une autre alternative pour le traitement des règles SWRL. Jess, écrit en Java, est un outil libre seulement pour un usage académique. Il est parmi les moteurs de règles les plus puissants. Il permet l'application des règles en se basant sur une version améliorée de l'algorithme de Rete [For90]. Il peut être utilisé avec le plugin *SWRLTab* dans Protégé pour l'exécution des règles dans une interface graphique ou bien avec l'API Protégé-OWL pour l'exécution des règles dans un programme.

Choix des outils

Vu la diversité des outils qui existent, nous avons fait un ensemble de choix pour la création de GACO et sa manipulation (Chargement, instanciation et mise à jour des individus, traitement des règles et raisonnement) dans le framework de contrôle d'accès. Nos choix sont illustrés dans le Tableau 5.1. Nous avons choisi Protégé 3.4.6 pour la création et l'édition des ontologies. Ce choix a été conditionné par : (1) le besoin d'un outil qui permet la création et l'édition des règles SWRL et (2) du besoin des *built-ins* SWRL de type *swrlx :createOWLThing* qui permettent de créer des individus dans une règle SWRL. Ces deux besoins sont satisfaits par l'utilisation du plugin *SWRLTab* de Protégé 3.4.6. Nous avons choisi le moteur d'inférence Pellet pour la vérification de la consistance de l'ontologie et les autres tâches d'inférence grâce à sa puissance. Toutefois nous ne l'avons pas retenu pour l'application des règles SWRL car il ne supporte pas le *built-in* de type *swrlx :createOWLThing* dont nous avons besoin. Nous avons retenu Jess comme moteur de règles. L'utilisation de Jess a deux avantages : (1) il peut être activé dans le plugin *SWRLTab* afin d'exécuter les règles et les tester dans Protégé et (2) il peut être utilisé depuis l'API Protégé-OWL qui fournit un ensemble d'interfaces qui permettent la mise en œuvre des règles SWRL. L'interface de programmation *SWRLRuleEngineBridge* permet de créer un pont qui fournit des mécanismes pour :

- Importer les règles SWRL et tous les éléments (classes, individus, propriétés) du modèle de l'ontologie.
- Permettre au moteur Jess d'effectuer les inférences
- Récupérer les nouvelles connaissances dans le modèle de l'ontologie

Fonctionnalité	Choix
Création et édition des ontologies	Protégé 3.4.6
Inférence	Pellet 1.5.2
Création et édition des règles SWRL	SWRLTab dans Protégé 3.4.6
Application des règles SWRL	Jess 7
API pour le traitement des règles	Protégé-OWL API 3.4.6

Tab. 5.1 – *Choix d'outils*

L'interface de programmation *SWRLFactory* est une autre interface de l'API Protégé-OWL qui permet la manipulation des règles SWRL tels que la création, la suppression, l'importation, etc. Elle permet également l'activation et la désactivation dès règles SWRL afin de définir leur ordre d'exécution.

13. http://jessrules.com/

5.1.2 Outils pour le traitement des politiques de contrôle d'accès

Dans cette sous-section, nous présentons les outils qui permettent l'édition des politiques de contrôle d'accès écrites en XACML et leur application dans un environnement collaboratif.

Édition des politiques de contrôle d'accès

Contrairement à l'édition des ontologies, il n'existe pas de nombreux outils libres qui permettent la création et l'édition des politiques XACML. Nous citons ci-dessous ces outils :

WSO2 Identity server [14] est un serveur d'authentification et de gestion des droits développé en 2007. Il fournit un éditeur graphique pour la création et l'édition des politiques XACML. L'éditeur est simple à utiliser et ne nécessite pas une maîtrise du standard XACML. En utilisant cet éditeur, les utilisateurs sont supposés être enregistrés dans un annuaire LDAP. Malgré sa facilité d'utilisation, cet éditeur ne peut être utilisé que depuis un navigateur après la mise en place et la configuration du serveur WSO2 dans l'une des machines ; Ce qui n'est pas toujours possible dans les environnements ubiquitaires.

UMU-XACML-Editor [15] est un autre éditeur de politiques XACML, développé par l'université de Murcia. L'avantage de cet éditeur est qu'il supporte les profils RBAC de XACML ; ce qui permet de définir un RPS et un PPS pour chaque rôle de l'environnement collaboratif. Nous avons choisi cet éditeur pour la création et l'édition des politiques. Plus précisément sa version 1.3.2 qui est compatible avec XACML 2.0.

Bibliothèques pour l'implantation des composants de contrôle d'accès

Il existe plusieurs bibliothèques libres qui permettent l'implantation des composants de contrôle d'accès de type PEP et PDP selon la spécification XACML. Parmi les bibliothèques développées en Java, nous citons les plus répandues telles que Balana [16], OpenAZ [17], enterprise-Java [18], SUN-XACML [19] et JBoss PicketBox [20].

La bibliothèque que nous avons retenue est la bibliothèque JBoss PicketBox. Nous avons utilisé la dernière version stable 2.0.9. Cette bibliothèque fournit les interfaces de programmation nécessaires pour l'implantation des PEPs et des PDPs. Afin d'assurer un échange standard entre les PEPs et les PDPs, les deux interfaces *RequestContext* et *ResponseContext* sont utilisées. L'interface *RequestContext* permet aux PEPs de traduire les requêtes des utilisateurs au format standard. Une requête est représentée par un quadriplet $< id, r, res, a >$ où id représente l'identifiant de l'utilisateur, r représente son rôle, res représente la ressource à laquelle il veut accéder et a représente l'action qu'il veut effectuer sur la ressource. L'interface *ResponseContext* permet au PDP de générer les décisions d'autorisation et les retourner aux PEPs au

14. http://wso2.com/products/identity-server/
15. http://umu-xacmleditor.sourceforge.net/
16. http://xacmlinfo.org/category/balana/
17. http://www.openliberty.org/wiki/index.php/OpenAz_Main_Page
18. http://code.google.com/p/enterprise-java-xacml/
19. http://sunxacml.sourceforge.net/
20. https://community.jboss.org/wiki/PicketBoxXACMLJBossXACML

format standard XACML. Parmi plusieurs autres interfaces, il y a celles qui permettent d'enregistrer les décisions dans des fichiers, de les afficher, etc.

5.2 Description du projet Galaxy

Afin d'illustrer le fonctionnement du framework de contrôle d'accès et du framework de communication, nous avons développé une application prototype dans le cadre du projet Galaxy [21]. Nous illustrons à travers cette application le processus et les techniques de déploiement que nous avons décrits dans le chapitre 4.

Le projet Galaxy est un projet financé par l'Agence Nationale de la Recherche (ANR). L'ANR assure la sélection, le financement et le suivi des projets. Elle est aussi le principal opérateur des Investissements d'Avenir dans le domaine de l'enseignement supérieur et de la recherche. Le projet Galaxy implique plusieurs partenaires académiques et industriels. Les principaux partenaires sont le LAAS-CNRS, Airbus, AKKA technologies, Softeam, IRIT et LIP6.

Le but du projet Galaxy est de concevoir et développer une plateforme de services, appelée *Platefome Galaxy*, afin d'adresser le problème du travail collaboratif sur des modèles pour le développement de systèmes complexes. Il vise à définir une architecture ouverte et flexible conçue spécifiquement pour passer à l'échelle. Cela suppose de traiter les verrous technologiques relatifs à la fragmentation et à la distribution des modèles de grande taille, ainsi qu'à leur synchronisation aux travers des moyens de communication utilisés classiquement par les équipes de développement. Un ensemble d'exigences fondamentales ont été identifiées afin de fournir des solutions pour la gestion des modèles complexes dans les environnements de travail collaboratif : Les principales exigences sont :

- Le système doit être capable de gérer et synchroniser des vues multiples du même modèle,
- Le système doit permettre l'utilisation concurrente d'un modèle par des utilisateurs différents,
- Le système doit gérer les versions des modèles,
- Le système doit permettre la modification concurrente d'un modèle de processus,
- Le système doit permettre l'interconnexion entre différents outils tels que Eclipse MDT Papyrus et Softeam Modelio pour assurer l'échange des modèles entre eux,
- Le système doit offrir des services pour la communication et l'échange de données entre les utilisateurs,
- Le système doit être capable d'identifier les utilisateurs selon leurs rôles et leurs appartenances aux groupes,
- Le système doit détecter les changements de rôles et de groupes de chaque participant et lui fournir automatiquement les outils nécessaires pour qu'il puisse communiquer avec les autres participants,
- Le système doit gérer l'accès aux données d'un modèle,
- Le système doit permettre de gérer les rôles et les groupes pour chaque participant.

21. http://www.irit.fr/GALAXY

Avant de présenter l'architecture de la plateforme Galaxy, nous présentons les principaux mots techniques utilisées :
Modèle de processus (ou *Process Model*) : désigne une description du processus de développement du système.
Modèle de processus générique (*Generic Process Model* ou gpm) : désigne le savoir-faire lié au processus de développement dans une organisation indépendamment d'un projet particulier, il peut être représenté par un diagramme CM_SPEM.
Modèle de processus exécutable (*Enactable Process Model* ou epm) : désigne un modèle de processus spécifique à un projet donné. Dans ce modèle, toutes les ressources nécessaires doivent être allouées.
Processus collaboratif (*Collaborative process*) : désigne un processus qui implique au moins une activité collaborative. Une activité collaborative est une activité réalisée par deux ou plusieurs participants.
Processus MDE Collaboratif (*Collaborative MDE Process*) : désigne un processus collaboratif qui tient compte des particularités de l'ingénierie dirigée par les modèles (*Model Driven Engineering* ou MDE).
Fragment de modèle (*Model fragment*) : désigne une partie d'un modèle qui peut être modifiée d'une manière indépendante.
Participant : désigne un développeur ou une institution qui participe à un projet MDE collaboratif en utilisant le framework Galaxy.
Exécution du processus (*Process enactment*) : désigne l'action d'exécuter un processus selon un modèle de processus. Le mot "enactment" est utilisé au lieu du mot "exécution" afin d'indiquer que les êtres humains sont aussi impliqués dans l'exécution du processus en plus des agents informatiques.
Unité collaborative (*Collaborative Unit*) : fournit un environnement de travail local pour chaque participant lui permettant d'éditer une copie d'un modèle. Chaque groupe de travail peut configurer leur propre unité collaborative, mais si plusieurs groupes de travail veulent travailler sur le même modèle, un repository central doit être configurée avec un mécanisme de verrous.

La Figure 5.1 présente les différentes phases du processus de développement dans chaque projet.

Lors de la première phase, le concepteur (*Process designer*) définit et décrit le modèle du processus MDE en spécifiant sa structure, et son comportement. Le résultat est un modèle de processus générique. Lors de la deuxième phase, le chef du projet (*Project manager*) définit un modèle de processus exécutable (enactable) qui représente une spécification du modèle de processus générique. Lors de la troisième phase, la plateforme Galaxy doit permettre aux développeurs d'exécuter des opérations et de collaborer selon le cycle de vie du processus défini dans le modèle de processus. Les développeurs peuvent également suivre l'état du projet MDE, ce qui a été fait et ce qui reste à faire. Le résultat peut être représenté par des modèles, du code, de la documentation, etc.

L'architecture, illustrée par la Figure 5.2, montre les interactions entre les différents services de la plateforme Galaxy.

La plateforme Galaxy utilise un serveur central (*Galaxy Server*) qui gère tous les projets de la plateforme Galaxy. Les outils de modélisation des différents projets doivent utiliser la même configuration du serveur. Afin d'assurer la compatibilité des différents outils avec la configuration du serveur,

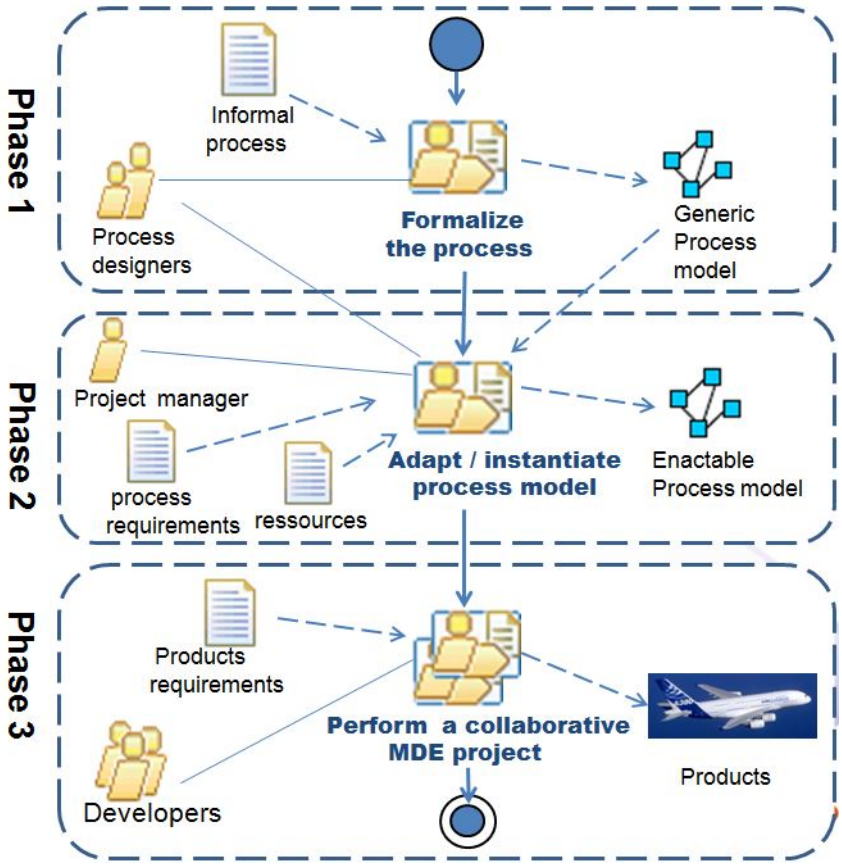

FIGURE 5.1 – *Les phases du processus de développement*

la plateforme Galaxy définit un agent, appelé *Galaxy Agent*, qui doit être déployé dans l'environnement de travail de chaque groupe. Cet agent permet d'homogénéiser la communication entre les outils de modélisation et le serveur Galaxy. Il représente l'extension (module, plugin, etc.) que les outils de modélisation (*MDE Tool*) doivent implémenter et intégrer afin qu'ils soient compatibles avec la plateforme Galaxy. Il permet à chaque groupe de travail d'utiliser le serveur Galaxy et donc de pouvoir collaborer avec les autres groupes de travail. En suivant cette approche, le serveur Galaxy ne communique qu'avec les agents Galaxy et il est indépendant des différents outils de modélisation utilisés.

Afin de pouvoir produire un *epm* en spécialisant le *gpm*, les chefs de projets (*Project Manager*) utilisent un outil de modélisation (PM Tool) qui doit être installé dans leurs dispositifs. Durant cette activité, chaque chef de projet définit et affecte des tâches aux utilisateurs impliqués dans le projet. Après chaque modification du modèle *epm*, le moteur de processus (*Process Engine*) se charge de son stockage dans la base de données (Galaxy DB). Les utilisateurs peuvent ensuite consulter leurs tâches à travers leur outil MDE (MDE Tool) en utilisant l'outil appelé *Process Enactment Client*. Ils sont également capables de notifier le *Process Engine* après chaque évènement lié à leurs tâches. Le PM Tool utilisé dans Galaxy est l'outil Topcased 5.1 [22] .

Le mécanisme de fragmentation permet de minimiser la quantité de don-

22. http://www.topcased.org/

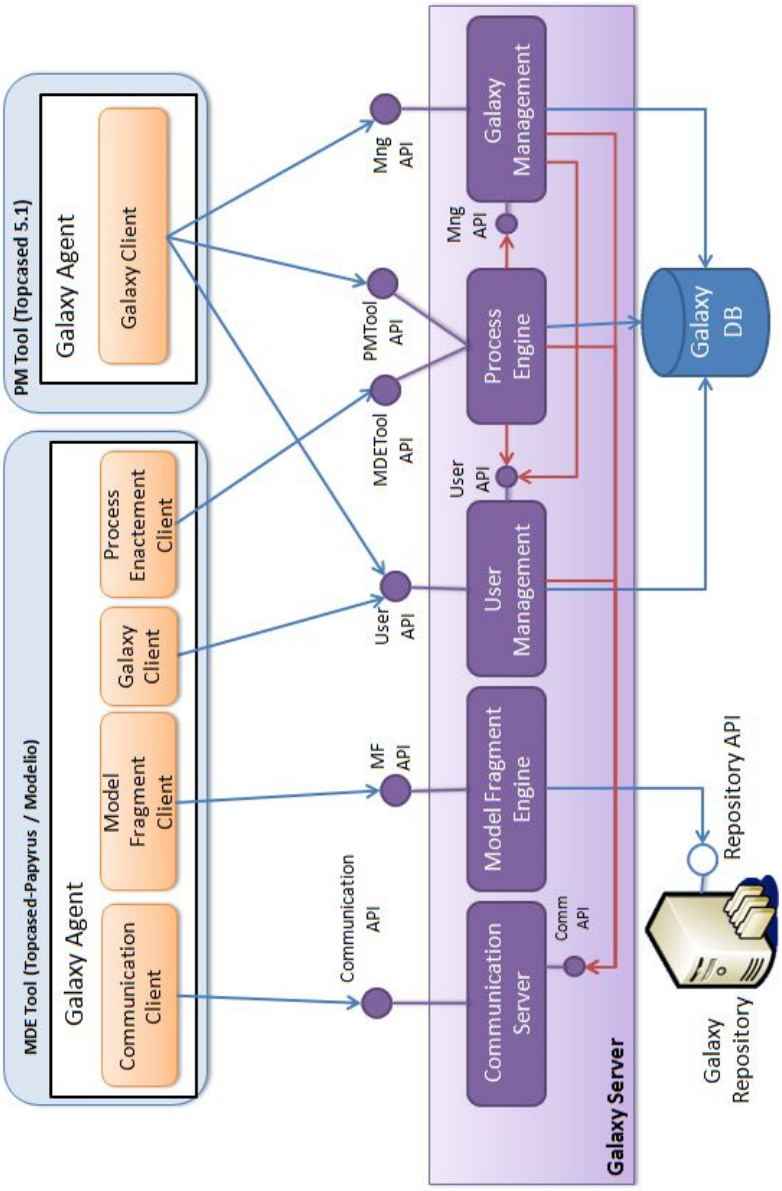

FIGURE 5.2 – *Architecture de la plateforme Galaxy*

nées à gérer quand des participants ont besoin de travailler sur une partie d'un modèle. Il adresse les problèmes de passage à l'échelle en permettant la synchronisation d'une partie d'un modèle, appelée fragment de modèle, au lieu de synchroniser tout le modèle. La consommation des ressources du mécanisme de fragmentation dépend de la fréquence de synchronisation et du temps nécessaire pour effectuer la synchronisation. Afin de pouvoir lire et modifier les éléments d'un fragment de modèle, les utilisateurs utilisent un client de fragmentation de modèle (*Model Fragment Client* ou MFC). Le moteur de fragmentation de modèle (*Model Fragment Engine*) du serveur Galaxy permet la gestion des fragment de modèles ainsi que leur stockage dans le registre (*Galaxy Repository*).

Le serveur Galaxy fournit :

– Une base de données qui stocke les entités Galaxy et les informations liées aux processus.

- Un repository (*Galaxy Repository*) qui stocke les versions des fragments de modèles et les rend accessibles aux développeurs.
- Deux composants d'administration (*Galaxy Management* et *User Management*) qui permettent de gérer les projets, les utilisateurs, les rôles et les groupes de travail. Ces deux composants sont accessibles à travers le client Galaxy (*Galaxy Client*).

Quand des groupes de développeurs travaillent sur le même projet, la communication est un facteur important pour l'efficacité des résultats. Elle permet un échange d'idées, d'informations et de ressources entre les différents développeurs. L'objectif est d'assurer la communication d'une manière simple et intelligente en se basant sur des sessions de communication qui regroupent des participants ayant les mêmes rôles et appartenant aux mêmes groupes. Notre équipe du LAAS-CNRS est responsable de cette tâche qui consiste à livrer tous les outils qui assurent la communication dans le projet Galaxy. Nous sommes responsables de la livraison des composants client et serveur de communication (Communication Client et Communication Server). Nous profitons également de la plateforme Galaxy pour illustrer l'utilisation du framework de contrôle d'accès.

5.3 Application prototype

Dans cette section, nous présentons le cas d'étude que nous avons utilisé afin d'illustrer l'utilisation du framework de contrôle d'accès et du framework de communication. Ce cas d'étude nous permettra, d'une part, de valider le fonctionnement du framework de contrôle d'accès ainsi que le processus et les techniques de déploiement que nous appliquerons pour les deux framework, et d'autre part, d'évaluer les performances de notre implantation. L'application prototype que nous avons utilisée dans le cadre du projet Galaxy est une application qui permet aux participants d'un projet donné de communiquer et de partager certains types de ressources tels que des documents de spécification et de conception. Dans cette application, un ensemble d'utilisateurs regroupés dans des groupes de travail, communiquent et partagent des ressources dans des sessions de collaboration.

5.3.1 Description des phases

Dans cette sous-section, nous décrivons les étapes qui doivent être effectuées dans les différentes phases (Figure 5.1) définies dans le projet Galaxy afin d'assurer la communication entre les utilisateurs et de sécuriser l'accès aux ressources.

Phase 1

Lors de la première phase, le concepteur des ontologies doit définir les ontologies génériques constituant le modèle de processus générique associé à la collaboration et au contrôle d'accès. Ces ontologies correspondent respectivement à GCO et GACO. Elles doivent être fournies sous la forme de deux fichiers OWL. Lors de la deuxième phase, ces deux ontologies doivent être étendues afin de définir les concepts spécifiques à l'application.

Phase 2

Dans cette phase, le concepteur de l'application définit la partie du modèle de processus dédiée à la communication et au contrôle d'accès. Cette phase correspond à réaliser toutes les étapes définies dans Figure 3.8. Elle consiste à définir la hiérarchie de rôles, les groupes, les sessions, les situations de collaboration et les URLs des ressources et des politiques de contrôle d'accès. Toutes ces informations spécifiques à l'application sont définies dans deux modèles de domaine qui étendent les deux ontologies GACO et GCO. En plus des ontologies de domaine, les règles SWRL spécifiques à l'application doivent ainsi être spécifiées afin d'exprimer les situations de collaboration.

Ontologie de domaine

Dans le cadre du projet Galaxy, nous avons proposé un ensemble de rôles qui peuvent être affectés aux différents participants d'un projet MDE. Ces rôles sont décrits comme suit :

- Concepteur (*Designer*) : il y a deux types de concepteurs dans l'application :
  - *Simple Designer* : fait la conception du système et définit les modèles de description de l'architecture,
  - *Designers Leader* : collabore avec les autres concepteurs afin de produire l'architecture détaillée et globale du système.
- Intégrateur (*Integration Manager*) : analyse les différentes fonctionnalités du système, décide si elles sont prêtes pour production et les intègre en s'assurant que le résultat fonctionne sans bugs.
- Développeur (*Developer*) : il y a deux types de développeurs dans l'application :
  - Développeur de codes (*Code Developer*) : écrit le code du système,
  - Développeur de tests (*Test Developer*) : écrit les tests d'intégration du système.
- Responsable du déploiement (*Deployment Manager*) : déploie le système pour production, supervise son exécution et retourne les erreurs d'exécution pour les développeurs.

Après avoir défini la hiérarchie de rôles, le concepteur de l'application doit définir les différents groupes, les sessions de collaboration et les URLs des ressources et des politiques de sécurité associés à chaque session. Il y a deux possibilités pour la définition des groupes et des sessions :

- les définir en tant que instances dans l'ontologie de domaine : cette méthode s'applique quand les groupes et les sessions sont connus à l'avance avant l'exécution de l'application. Cependant le concepteur de l'application a la possibilité d'ajouter et de supprimer des groupes et des sessions au cours de l'exécution de l'application ou bien
- les définir lors de la phase d'initialisation de l'application : cette méthode s'applique quand les groupes et les sessions ne sont connus que lors de l'initialisation de l'application.

Dans notre cas d'étude, nous définissons les groupes et les sessions dans l'ontologie de domaine qui étend GACO. L'ontologie de domaine est illustrée par la Figure 5.3. Dans cette figure, nous avons représenté en gris les concepts de GACO qui sont importés dans cette ontologie en utilisant le

préfixe *gaco*. Les concepts spécifiques à l'application sont représentés par des ellipses blanches avec des lignes hachurées. Tandis que les instances sont représentées par des ellipses en pointillés.

L'ensemble de rôles que nous avons définis sont représentés par les concepts suivants : *Designer*, *SimpleDesigner*, *DesignersLeader*, *IntegrationManager*, *Developer*, *CodeDeveloper* et *TestDeveloper*. Ces concepts héritent du concept *Role* de GACO. En conséquence, ils seront traités en tant que rôles dans le processus d'inférence et de raisonnement. Selon GACO, ces rôles peuvent être affectés à des nœuds à travers la relation *hasRole* et appartenir à un ou plusieurs groupes à travers la relation *belongsToGroup*. Dans Galaxy, les nœuds sont des acteurs qui participent aux projets MDE. Ils sont représentés par le concept *Actor* qui hérite du concept *Node* de GACO. Dans cette ontologie, nous définissons deux groupes de travail, appelés *workGroupA* et *workGroupB*, qui sont représentés par des instances du concept *Group* de GACO.

Dans cette ontologie de domaine, nous avons défini quatre sessions de collaboration :

- *Designers_Session* : cette session permet la communication entre les concepteurs (*SimpleDesigner* et *DesignersLeader*)
- *DesignerLead_Integrator_Session* : cette session permet la communication entre le *DesignersLeader* et les *IntegrationManager*.
- *Integrator_Developer_Session* : cette session permet la communication entre les intégrateurs (*IntegrationManager*) et les développeurs (*Developer*)
- *DesignerLead_Developer_Session* : cette session permet la communication entre les concepteurs (*DesignersLeader*) et les développeurs (*CodeDeveloper* et *TestDeveloper*)

Ces sessions sont représentées par des instances du concept *Session* de GACO. Comme c'est illustré par la Figure 5.3, les utilisateurs qui appartiennent au groupe de travail *workGroupA* sont organisés dans deux sessions *Designers_Session* et *DesignerLead_Integrator_Session*, tandis que les utilisateurs qui appartiennent au groupe de travail *workGroupB* sont organisés dans les deux sessions *Integrator_Developer_Session* et *DesignerLead_Developer_Session*. L'attribution des sessions aux groupes se fait à travers la relation *hasSession* de GACO. Toutes ces informations (i.e, les rôles, les groupes, les sessions) définies dans l'ontologie de domaine doivent être reproduites dans l'ontologie de domaine qui étend GCO.

Pour chaque session, nous devons définir la valeur de la propriété *hasResourcesRepository* qui permet de spécifier l'URL du répertoire qui héberge les ressources partagées dans la session, et la valeur de la propriété *hasPoliciesRepository* qui indique l'URL du répertoire qui héberge les politiques de contrôle d'accès associées à la session. Les URLs associés à chaque session sont représentés dans le Tableau 5.2.

Toutes les informations de l'ontologie de domaine qui étend GACO seront utilisés par le framework de contrôle d'accès afin d'assurer le contrôle d'accès aux ressources dans chaque session.

Règles SWRL de l'application

Après la définition des éléments qui étendent GACO et GCO, les règles SWRL associées à l'application doivent être spécifiées afin d'exprimer les

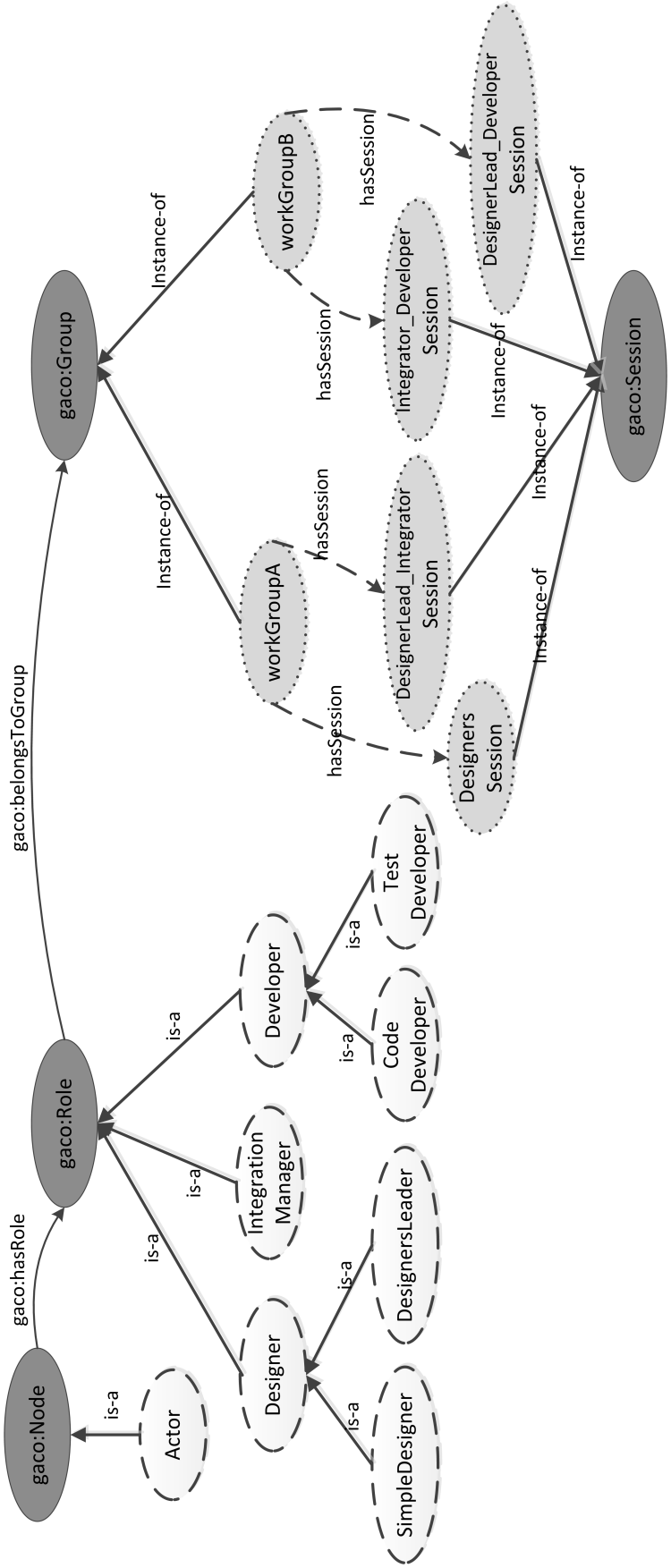

FIGURE 5.3 – *Ontologie de domaine*

Session	hasResourcesRepository	hasPoliciesRepository
Designers_Session	URL_resources_1	URL_policies_1
DesignerLead_ Integra- tor_Session	URL_resources_2	URL_policies_2
Integrator_ Deve- loper_Session	URL_resources_3	URL_policies_3
DesignerLead_ Develo- per_Session	URL_resources_4	URL_policies_4

Tab. 5.2 – *Les URLs des ressources et des politiques*

situations de collaboration. Nous intégrons ces règles dans le même fichier OWL de l'ontologie de domaine.

La règle *DesignersSession_rule*, présentée dans la Figure 5.4, permet d'indiquer au framework de contrôle d'accès que les concepteurs communiquent ensemble dans la session appelée *DesignersSession*. Ils peuvent également partager des ressources durant le projet MDE dans cette session. La règle trouve tous les utilisateurs (*Node*) qui jouent les rôles *SimpleDesigner* et *DesignersLeader* qui appartiennent au même groupe. Pour chaque individu trouvé, la règle l'associe à la session *DesignersSession* en utilisant la propriété *gaco* : *involvedIn*. Nous avons choisi d'utiliser une règle qui associe les deux rôles *SimpleDesigner* et *DesignersLeader* à la session *DesignersSession* car nous supposons qu'il y a au moins un *SimpleDesigner* et un *DesignersLeader* dans la session.

gaco :Node(?ndl) ∧ DesignersLeader(?rdl) ∧ gaco :hasRole(?ndl, ?rdl) ∧
gaco :Node(?nd) ∧ SimpleDesigner(?rd) ∧ gaco :hasRole(?nd, ?rd) ∧
differentFrom(?rd, ?rdl) ∧ gaco :belongsToGroup(?rd, ?g) ∧
gaco :belongsToGroup(?rdl, ?g) ∧ gaco :Session(?s) ∧
gaco :hasSession(?g, ?s) ∧ gaco :hasSessionName(?s, "DesignersSession")
→ gaco :involvedIn(?rd, ?s) ∧ gaco :involvedIn(?rdl, ?s)

FIGURE 5.4 – *Règle DesignersSession_rule*

De la même façon, nous définissons les trois autres règles : *DesignerDeveloperSession_rule* permet d'associer les concepteurs et les développeurs à la session *DesignerLead_Developer_Session*, *Integrator_Developer_Session_rule* permet d'associer les intégrateurs et les développeurs à la session *Integrator_Developer_Session* et la règle *DesignerLeaderIntegratorSession_rule* permet d'associer les Designer Leader et les intégrateurs à la session *DesignerLead_Integrator_Session*. Après chaque changement au niveau de l'application, le framework de contrôle d'accès commence par exécuter ces règles. Ensuite, il exécute les règles stockées dans GACO afin de déterminer les composants de contrôle d'accès qui doivent être mis en place afin de contrôler l'accès aux ressources.

L'ontologie de domaine qui étend GCO doit aussi contenir les règles SWRL qui décrivent les situations de collaboration dans chaque session. Ces règles permettent de créer flux de communication entre les différents parti-

cipants et de déterminer les composants qui doivent être mis en place afin d'assurer la communication dans chaque session.

Politiques de contrôle d'accès

Lors de la première phase, le concepteur de l'application doit définir les politiques de contrôle d'accès. Pour cela, il utilise l'éditeur de règles UMU-XACML-Editor afin de définir un PPS et un RPS pour chaque rôle dans chaque session de collaboration. Ici, nous définissons un ensemble de règles qui sont valides dans toutes les sessions afin de simplifier le cas d'étude. Nous considérons les deux fichiers *architecture.doc* et *rapport_tests.doc* que les utilisateurs peuvent lire et modifier.

Le Tableau 5.3 illustre les permissions associées à chaque rôle.

Rôle	Ressource	Permission
Designer	architecture.doc	Read / Write
	rapport_tests.doc	Read
Integration Manager	architecture.doc	Read
	rapport_tests.doc	Read
Developer	architecture.doc	
	rapport_tests.doc	Read
Deployment Manager	architecture.doc	Read / Write
	rapport_tests.doc	Read / Write

Tab. 5.3 – *Permissions associées aux rôles*

Ces permissions sont définies par le concepteur de l'application en utilisant l'éditeur UMU-XACML. La Figure 5.5 montre un exemple d'utilisation de l'éditeur pour la création du PPS associé au rôle développeur.

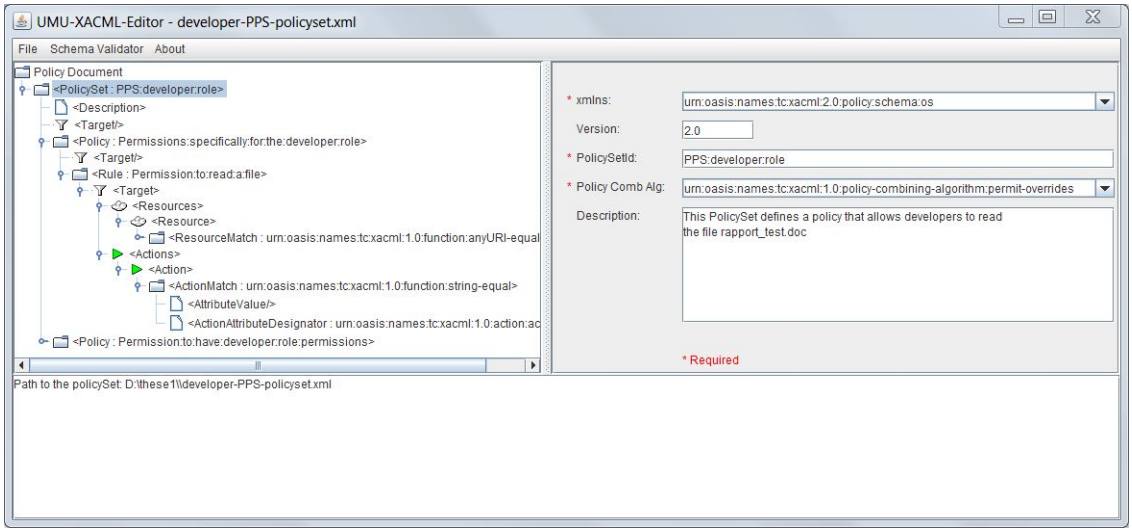

FIGURE 5.5 – *Création des politiques avec l'éditeur UMU-XACML*

Après avoir créé le PPS associé au rôle développeur, l'éditeur permet de générer un fichier XML, appelé $developer - PPS - policyset.xml$, qui contient toutes les permissions associées aux développeurs. Ce fichier est illustré par la Figure 5.6. Dans ce fichier, un ensemble de politiques (*PolicySet*) a été défini en utilisant la balise $< PolicySet >$. Cet ensemble de politiques est iden-

tifié par l'attribut *PolicySetId* qui a comme valeur *PPS* : *developer* : *role*. Cet identifiant sert pour associer le rôle développeur à ce *PolicySet*. Le *PolicySet* contient une description, une cible (*Target*) et une politique (*Policy*). La description est fournie en utilisant la balise $< Description >$. Elle est représentée par une description textuelle de l'ensemble des permissions définies dans le *PolicySet*. Nous avons choisi de définir la cible au niveau de la règle afin de pouvoir définir une permission pour chaque cible. En conséquence, la balise $< Target >$ du *PolicySet* est vide. Dans le *PolicySet*, nous définissons une politique, en utilisant la balise $< Policy >$, qui permet de créer une règle permettant aux développeurs de lire le fichier *rapport_tests.doc*. L'attribut *RuleCombiningAlgId* de la politique permet de définir l'algorithme qui permet de combiner les décisions de plusieurs règles. Dans notre cas, cet attribut a comme valeur $permit - overrides$. La règle qui affecte la permission de lecture aux développeurs est représentée par la balise $< Rule >$. Les deux attributs de la règle sont $Effect$ et *RuleId*. La valeur de l'attribut $Effect$ est "Permit". Cela veut dire que la règle autorise l'action sur la ressource seulement si la cible (ou *Target*) est satisfaite. La cible, représentée par la balise $< Target >$ définit deux conditions nécessaires pour que l'effet ($Effect$) de la règle soit applicable. La première condition concerne le nom de la ressource qui doit être $URL\_resources\_1/rapport\_tests.doc$. Tandis que la deuxième condition concerne le nom de l'action qui doit être *read*.

Afin d'associer l'ensemble de politiques définies dans le fichier $developer - PPS - policyset.xml$ au rôle développeur (*Developer*), nous avons besoin de créer le $Role < PolicySet >$ (ou RPS) pour le rôle développeur. Ce RPS, présenté dans la Figure 5.7, définit un *PolicySet* identifié par la valeur *RPS* : *developer* : *role*. Ce *PolicySet* contient deux éléments :

- Une cible (*Target*) qui identifie le rôle concerné à travers la balise $< Subjects >$.
- Une référence vers le PPS en utilisant la balise $< PolicySetIdReference >$. La valeur indiquée dans cette balise doit représenter l'identifiant du fichier PPS associé au rôle développeur. Dans notre cas, la valeur indiquée est *PPS* : *dveloper* : *role* qui est déclarée dans l'attribut *PolicySetId* du PPS associé au rôle développeur.

Les deux fichiers PPS et RPS que nous avons montrés permettent de définir les permissions associées au rôle développeur. De la même façon, nous pouvons définir les permissions associées aux autres rôles. Grâce au mécanisme de référencement de XACML, un rôle peut hériter les permissions d'un autre rôle. Ce mécanisme d'héritage de permissions est réalisable à travers l'utilisation de la balise $< PolicySetIdReference >$ qui doit être définie à la fin du *PolicySet* du PPS. Si un *PPS*1 référence un autre *PPS*2, alors *PPS*1 hérite toutes les permissions définies dans *PPS*2.

La Figure 5.8 montre les relations d'héritage entre les PPS de chaque rôle. Le rôle développeur est le rôle qui a le moins de permissions. En conséquence, le PPS associé au rôle développeur ne fait référence à aucun autre PPS. Le rôle *IntegrationManager* hérite les permissions associées au rôle développeur et il a une autre permission de lecture du fichier *architecture.doc*. En conséquence, le PPS associé au rôle *IntegrationManager* définit la permission *read* sur le fichier *architecture.doc* et utilise la balise $< PolicySetIdReference >$ afin de faire référence au PPS associé au rôle développeur. La Figure 5.9 montre un exemple d'utilisation de cette balise

```
<?xml version="1.0" encoding="UTF-8"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" PolicyCombiningAlgId=
"urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides" PolicySetId=
"PPS:developer:role" Version="2.0">
  <Description>This PolicySet defines a policy that allows developers to read the file
  rapport_test.doc</Description>
  <Target />
  <Policy PolicyId="Permissions:specifically:for:the:developer:role" RuleCombiningAlgId=
  "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
    <Target />
    <Rule Effect="Permit" RuleId="Permission:to:read:a:file">
      <Target>
        <Resources>
          <Resource>
            <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
              URL_resources_1/rapport_tests.doc</AttributeValue>
              <ResourceAttributeDesignator AttributeId=
              "urn:oasis:names:tc:xacml:1.0:resource:resource-id" DataType=
              "http://www.w3.org/2001/XMLSchema#anyURI" />
            </ResourceMatch>
          </Resource>
        </Resources>
        <Actions>
          <Action>
            <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read
              </AttributeValue>
              <ActionAttributeDesignator AttributeId=
              "urn:oasis:names:tc:xacml:1.0:action:action-id" DataType=
              "http://www.w3.org/2001/XMLSchema#string" />
            </ActionMatch>
          </Action>
        </Actions>
      </Target>
    </Rule>
  </Policy>
</PolicySet>
```

FIGURE 5.6 – *Politique PPS associée au rôle développeur*

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
PolicySetId="RPS:developer:role"
PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
    <Target>
        <Subjects>
            <Subject>
                <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
                <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#anyURI">developer</AttributeValue>
                <SubjectAttributeDesignator
                AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
                DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
                </SubjectMatch>
            </Subject>
        </Subjects>
    </Target>
<!-- Use permissions associated with the developer role -->
<PolicySetIdReference>PPS:developer:role</PolicySetIdReference>
</PolicySet>
```

FIGURE 5.7 – *Politique RPS associée au rôle développeur*

dans le fichier *integrationManager − PPS − PolicySet*. De la même façon le PPS associé au rôle concepteur, appelé *designer − PPS − PolicySet*, fait référence au PPS associé au rôle *IntegrationManager* et rajoute la permission d'écriture dans le fichier *architecture.doc*. Finalement, le PPS associé au rôle *DeploymentManager*, appelé *deploymentManager − PPS − PolicySet*, fait ré-

férence au PPS associé au rôle concepteur et rajoute la permission d'écriture dans le fichier *rapport_tests.doc*.

Après la création des PPS en utilisant l'éditeur UMU-XACML, un RPS doit être créé pour chaque rôle afin de l'associer avec son PPS. Les RPS ont une structure semblable à celle présentée dans la Figure 5.7.

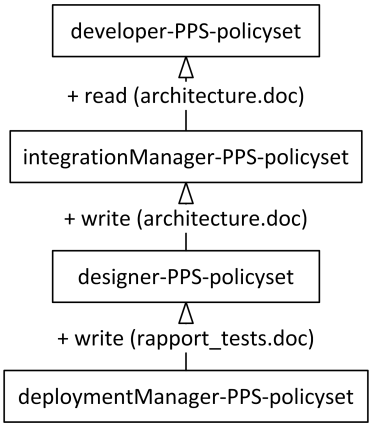

FIGURE 5.8 – *Référencement entre les PPS*

```
<PolicySetIdReference>PPS:developer:role</PolicySetIdReference>
```

FIGURE 5.9 – *Référencement du PPS associé au rôle développeur*

Phase 3

Durant cette phase, les utilisateurs, organisés dans des groupes de travail, collaborent et partagent des ressources dans les sessions de collaboration définies dans l'ontologie de domaine. Ils peuvent rejoindre des groupes et changer de rôle en fonction de leurs besoins durant l'activité collaborative. Après chaque action de l'utilisateur, le serveur de l'application notifie le framework de communication et le framework de contrôle d'accès afin d'exécuter les processus d'adaptation.

5.3.2 Architecture de l'application

L'architecture de l'application est illustrée par la Figure 5.10. En plus de l'outil MDE, chaque participant dispose d'un client de communication qui lui permet de se connecter au serveur Galaxy et de rejoindre des groupes de travail. Le serveur de communication gère les connexions des participants et leurs actions. Après chaque évènement, il notifie le framework de communication et le framework de contrôle d'accès afin de déployer les composants logiciels nécessaires.

Client de communication

Le diagramme de classe, présenté dans la Figure 5.11, montre les principales classes du client de communication de l'application. La classe principale est la classe *Client*. Elle permet l'initialisation de la partie client et fournit à la classe *ServerConnection* les informations de connexion telles que l'adresse IP

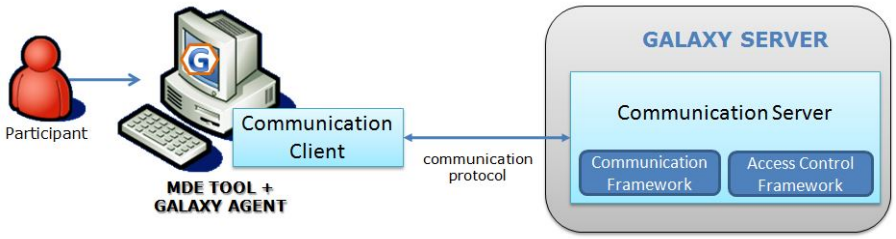

FIGURE 5.10 – *Architecture de l'application*

et le port d'écoute du serveur. La classe *ServerConnexion* permet d'établir la connexion avec le serveur et d'échanger de message à travers la classe *Comm_client_server*. Les donnés relatives à l'utilisateur telles que ses rôles et les groupes auxquels il appartient sont gérées par la classe *User*. La classe *GUI* fournit une interface graphique qui gère les connexions et les actions des utilisateurs. Nous avons utilisé la bibliothèque Swing qui fait partie de la bibliothèque JFC (*Java Foundation Classes* [23]) pour la gestion des éléments graphiques de la partie client. La Figure 5.12 montre l'interface graphique du client de communication. Elle contient les informations de connexions telles que les rôles qui peuvent être choisis par l'utilisateur (à gauche) et les groupes auxquels il peut appartenir (à droite).

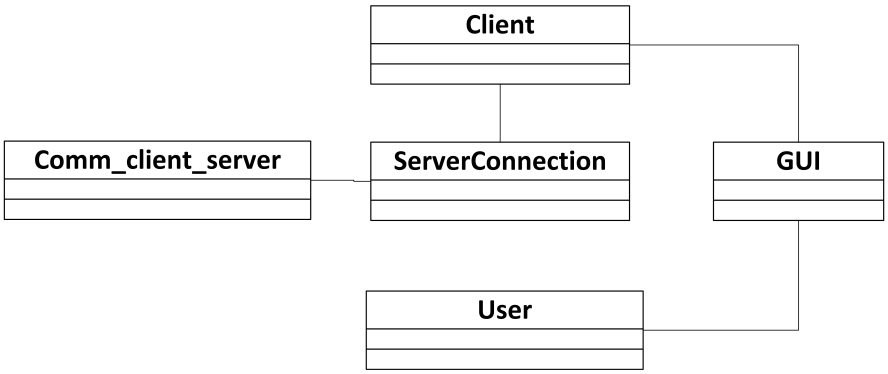

FIGURE 5.11 – *Classes principales du client de communication*

Chaque participant utilise cette interface pour se connecter au serveur de communication. Une fois connecté, il peut activer un ou plusieurs rôles et choisir un ou deux groupes de travail (*WorkGroupA* et *WorkGroupB*). Après la validation du serveur, le participant peut désactiver un rôle en utilisant le bouton *removerole*, activer un autre rôle en utilisant le bouton *addrole*, changer un rôle par un autre rôle en utilisant le bouton *changerole*, rejoindre un groupe en utilisant le bouton *joingroup*, ou bien quitter un groupe en utilisant le bouton *quitgroup*. Il peut également quitter l'application en utilisant le bouton *Quit*.

Serveur de communication

Les principales classes du serveur de communication sont illustrées par la Figure 5.13. La classe *Server* permet de gérer les connexions des utilisateurs à travers la classe *ClientConnection* qui utilise la classe *Comm_client_server*

23. http://docs.oracle.com/javase/tutorial/uiswing/start/about.html

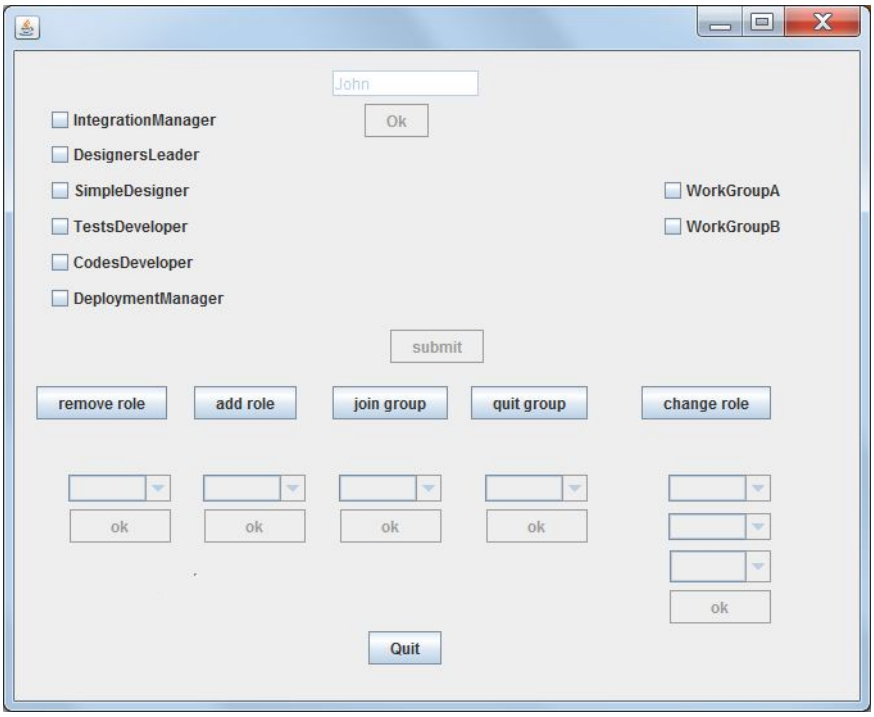

FIGURE 5.12 – *Interface graphique du client de l'application*

pour gérer les échanges de messages. La classe *ClientConnection* utilise également une classe *ClientAuthentication* afin gérer les authentifications des utilisateurs. La classe *ModelManager* permet de mettre à jour les instances de l'ontologie de domaine après chaque action d'un utilisateur. Elle représente l'API que nous avons décrite dans l'étape 2 de l'utilisation du framework de la section 3.3.4. La classe *ModelManager* contient également une fonction *initialize*() qui permet de charger les ontologies de domaine en utilisant l'API Protégé-OWL.

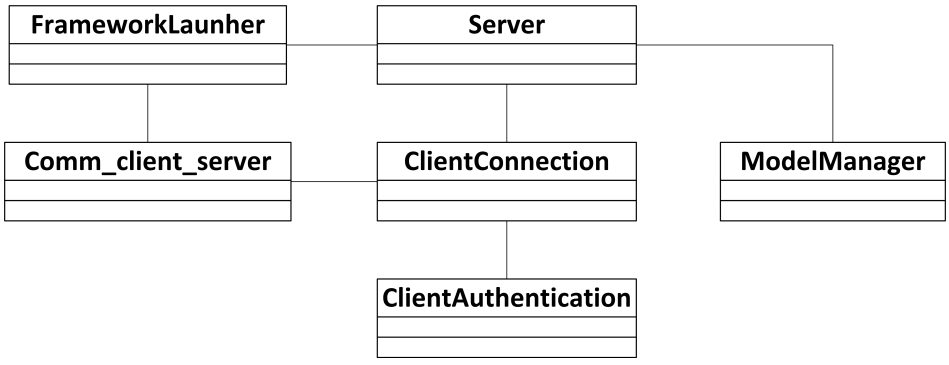

FIGURE 5.13 – *Classes principales du serveur de communication*

Après chaque réception d'un message de la part d'un utilisateur, une instance de la classe *ModelManager* se charge de mettre à jour l'instance de l'ontologie de domaine. Ensuite, une instance de la classe *FrameworkLauncher* est appelée afin d'exécuter le processus d'adaptation. Chaque processus d'adaptation finit par le déploiement des composants de communication et de contrôle d'accès dans les dispositifs des utilisateurs.

Protocole de communication
La communication entre le client et le serveur est basée sur les sockets qui permettent d'établir des canaux de communication suivant le protocole TCP ou UDP. Les sockets permettent l'échange de messages personnalisés sous la forme d'objets Java. Ce qui fait que le protocole de communication peut être représenté par un ensemble de classes où chaque classe correspond à un message échangé entre le client et le serveur. Ces classes sont représentées dans la Figure 5.14. Tous les messages héritent de la classe *Message* qui définit un attribut *id* représentant l'identifiant de chaque message. Chaque action de l'utilisateur est transformée en un objet de la classe correspondante qui sera envoyé au serveur de communication. Ceci est possible grâce au mécanisme de sérialisation en Java. Les objets de types *Acknowledgement* sont envoyés par le serveur au client afin de lui notifier du résultat de sa requête.

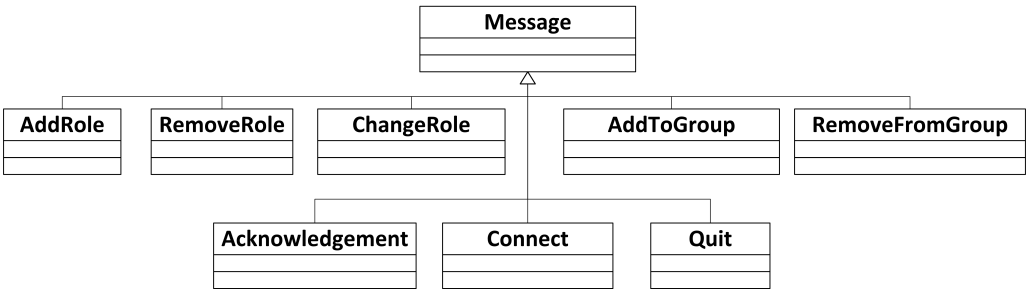

FIGURE 5.14 – *Diagramme de classes des messages échangés entre le client et le serveur*

5.4 ÉVALUATION FONCTIONNELLE

Dans cette section, nous présentons une évaluation fonctionnelle du framework de contrôle d'accès et des techniques de déploiement que nous avons appliquées aux deux frameworks. En premier lieu, nous présentons le scénario qui consiste en un ensemble d'évènements. Ensuite, nous décrivons l'interaction du serveur de l'application avec le framework de communication et le framework de contrôle d'accès après chaque évènement. Nous examinons également les résultats produits par les deux frameworks durant le déroulement du scénario. Les expérimentations que nous présentons dans cette sous-section nous permettent de valider le fonctionnement des processus d'adaptation dans plusieurs situations. Dans chaque situation, nous évaluons le comportement du framework de communication et du framework de contrôle d'accès depuis le moment où le changement se produit jusqu'au déploiement ou reconfiguration physique des composants logiciels dans les dispositifs des utilisateurs. Cette évaluation nous permet de vérifier si les processus d'adaptation produisent les résultats attendus pour chaque situation.

5.4.1 Scénario

Afin d'évaluer les processus d'adaptation, nous présentons un scénario qui illustre plusieurs situations pouvant avoir lieu au cours de l'exécution de

l'application. Ce scénario est illustré dans le Tableau 5.4. Il permet de suivre l'évolution de deux groupes de travail dans le temps. Durant le scénario, nous considérons les deux groupes de travail (*workGroupA* et *workGroupB*), les rôles, les ressources et les politiques que nous avons introduits dans la section 5.3. Dans le Tableau 5.4, la lettre A désigne le groupe *workGroupA*, et la lettre B désigne le groupe *workGroupB*. Dans le Tableau 5.5, nous rappelons les sessions associées à chaque groupe de travail ainsi que les rôles qui participent dans chaque session. Dans cette évaluation fonctionnelle, nous introduisons seulement trois utilisateurs dans le scénario afin de le rendre simple et clair. Cependant, nous considérerons plusieurs utilisateurs dans l'évaluation des performances.

Étape N	**Action**	**Situation dans les groupes**
1	John_Connect (Designers-Leader, A,B)	A :John
		B :John
2	Bob_Connect (SimpleDesigner, A)	A :John + Bob
		B :John
3	Tom_Connect (SimpleDesigner+TestDeveloper, B)	A :John + Bob
		B :John + Tom
4	Bob_AddRole (DeploymentManager)	A :John + Bob
		B :John + Tom
5	Bob_RemoveRole (SimpleDesigner)	A :John + Bob
		B :John + Tom
6	Bob_ChangeRole (DeploymentManager, IntegrationManager)	A :John + Bob
		B :John + Tom
7	Bob_JoinGroup(B)	A :John + Bob
		B :John + Tom + Bob
8	Bob_Quit(B)	A :John + Bob
		B :John + Tom

Tab. 5.4 – *Scénario*

Dans ce scénario, nous mettons en œuvre toutes les actions qui peuvent être effectuées par les utilisateurs durant l'activité collaborative. Le Tableau 5.4 décrit huit étapes où chaque étape représente une action (ou évènement). Pour chaque action, nous montrons la situation des utilisateurs dans chaque groupe. Les actions ont la forme suivante : nomUtilisateur_Action(rôles, groupes) où *nomUtilisateur* représente le nom de l'utilisateur qui effectue l'action et $Action(rles, groupes)$ représente l'action effectuée et les rôles et les groupes en question.

Groupe	Sessions	Rôles participants
workGroupA	Designers_Session	Simple Designer, Designers Leader
	DesignerLead_Integrator_Session	Designers Leader, Integration Manager
workGroupB	Integrator_Developer_Session	Integration Manager, Code Developer, Test Developer
	DesignerLead_Developer_Session	Designers Leader, Code Developer, Test Developer

Tab. 5.5 – *Sessions associées aux groupes*

5.4.2 Architecture matérielle

L'architecture matérielle que nous avons utilisée dans le scénario est illustrée dans la Figure 5.15. Nous avons utilisé un réseau sans fil de quatre machines identifiée chacune par une adresse IP :

- Machine serveur : cette machine est identifiée par l'adresse IP 140.93.1.10 ; elle héberge le serveur de communication qui intègre les deux frameworks sous forme de deux fichiers *jar*. Le registre (*Components Repository*) qui fournit les composants de communication et de contrôle d'accès est aussi démarré dans cette machine.
- Machine utilisée par le participant *John* : cette machine est identifiée par l'adresse IP 140.93.3.24 ; chaque machine utilisée par un utilisateur héberge le client de communication et l'agent de déploiement qui effectue le déploiement physique des composants dans la machine.
- Machine utilisée par le participant *Tom* : cette machine est identifiée par l'adresse IP 140.93.2.140
- Machine utilisée par le participant *Bob* : cette machine est identifiée par l'adresse IP 140.93.65.6.

Chaque participant utilise le client de communication pour se connecter au serveur de communication et effectuer des actions. Après la réception de chaque action, le serveur de communication effectue sa validation. Par exemple, il vérifie si l'utilisateur est permis d'activer un rôle donné. Ensuite, il crée ou met à jour l'instance de l'ontologie de domaine et appelle le framework de communication et de framework de contrôle d'accès pour générer le fichier de déploiement. En se basant sur ce fichier, le gestionnaire de déploiement assure le déploiement, la reconfiguration ou la désinstallation de certains composants.

5.4.3 Résultats

Dans cette sous-section, nous présentons les résultats des processus d'adaptation. Dans chaque étape du scénario, le résultat consiste à un dé-

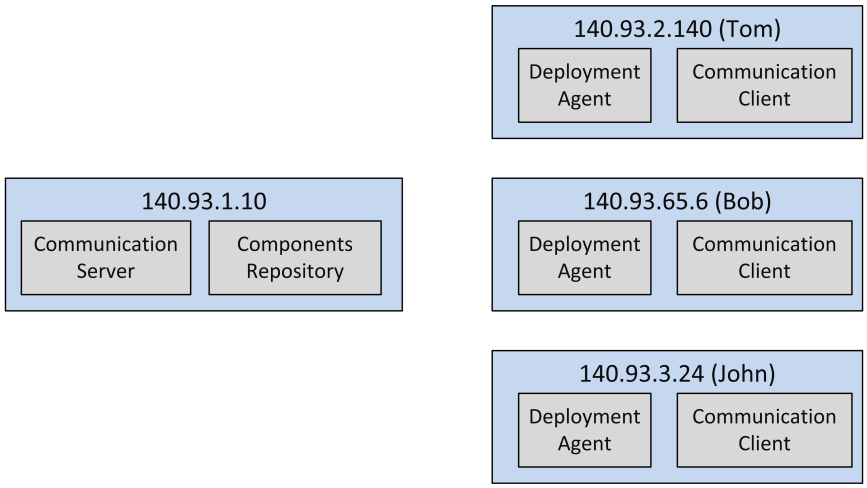

FIGURE 5.15 – *Architecture matérielle utilisée*

ploiement physique des composants de communication et de contrôle d'accès. Afin de détailler ces résultats, nous montrerons une représentation graphique des graphes générés pendant les processus d'adaptation en utilisant la bibliothèque Java Universal Network Graph [24] (JUNG). Dans cette évaluation, nous ne montrerons que les graphes générés par le framework de contrôle d'accès. Cependant, nous montrerons tous les composants déployés dans les machines des utilisateurs.

Étape 1

Cette étape correspond à la connexion du participant John aux deux groupes de travail A et B en tant que $DesignersLeader$. Avant cette étape, l'instance de l'ontologie de domaine contient les deux groupes de travail $workGroupA$ et $workGroupB$ et les quatre sessions associées aux deux groupes. Dans cette étape, l'instance de l'ontologie de domaine (Figure 5.16) contient en plus des groupes et des sessions, les individus qui correspondent au participant John qui a rejoint les deux groupes. Les individus de la Figure 5.16 sont représentés par des cercles et leurs noms sont précédés par le nom du concept auquel ils appartiennent. Les flèches qui relient les individus représentent les instances des relations. Les individus qui ont été créés dans cette étape sont le nœud qui représente le participant John, le point d'accès qu'il utilise ($JohnAP$) et son rôle ($DL1$). Le nœud est relié au rôle avec la relation $hasRole$, et au point d'accès avec la relation $uses$. La relation $hasMember$ relie les deux groupes au rôle.

Puisque les deux groupes ne contiennent qu'un seul participant, il n'y a pas de collaboration et de partage de ressources. En effet, les deux processus d'adaptation des deux framework ne génèrent pas des graphes en appliquant les règles SWRL, et donc ne génèrent aucun fichier de déploiement. En conséquence, aucun déploiement n'a eu lieu.

Étape 2

Cette étape correspond à la connexion du participant Bob au groupe $workGroupA$ en tant que $SimpleDesigner$. L'instance de l'ontologie de do-

24. http://jung.sourceforge.net/

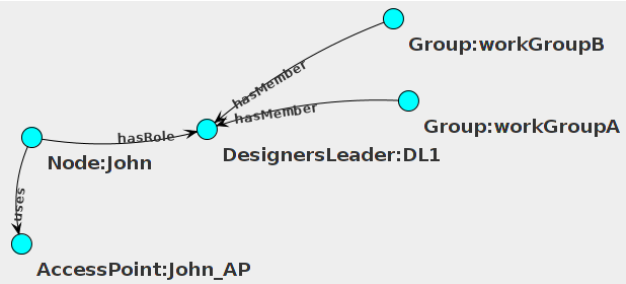

FIGURE 5.16 – *Étape 1 - instance de l'ontologie de domaine*

maine, représentée dans la Figure 5.17, ajoute à l'instance de l'étape 1 le nœud Bob, son rôle ($SD1$) et son point d'accès (Bob_AP).

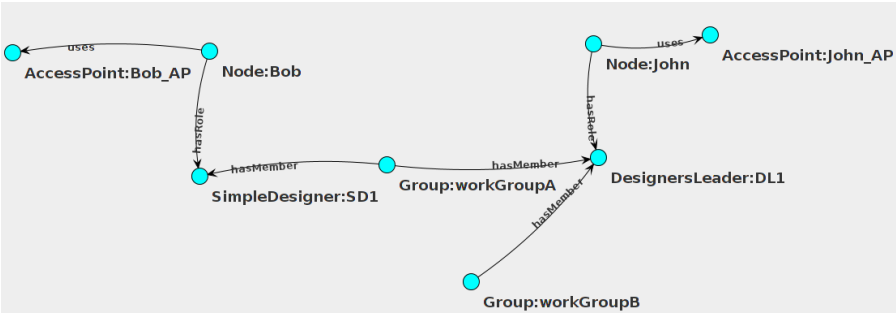

FIGURE 5.17 – *Étape 2 - instance de l'ontologie de domaine*

Le processus d'adaptation du framework de contrôle d'accès applique les règles SWRL à cette instance et produit une nouvelle instance de GACO qui crée des individus du concept *AccessControlComponent*. Ces individus, illustrés dans la Figure 5.18, représentent les composants qui gèrent l'accès aux ressources dans les dispositifs des participants. Les règles génèrent des individus qui ont la forme *SWRLInjected_N*. Le premier individu, appelé *SWRLInjected_1*, permet de gérer les accès de John (relation *managesAccessOf*) dans la session *designers_s* (relation *managesAccessInSession*). Le deuxième individu, appelé *SWRLInjected_2*, permet de gérer les accès de Bob dans la même session.

Les individus de type *AccessControlComponent* sont ensuite extraits de cette instance pour former le graphe de contrôle d'accès qui sera le graphe d'entrée pour le GMTE. Les nœuds de ce graphe sont représentés dans la Figure 5.19. Le premier nœud a comme identifiant *id_1* et gère les accès du dispositif de John (140.93.3.24) dans la session *designers_s* [25] . Tandis que le deuxième nœud a comme identifiant *id_2* et gère les accès du dispositif de Bob (140.93.65.6). Dans ce scénario, nous utilisons la grammaire de graphe GG_{simple} qui permet au GMTE de produire un seul graphe à partir du graphe de contrôle d'accès. Afin de pouvoir utiliser cette grammaire, le cinquième attribut doit avoir comme valeur 0 ou 1 afin d'indiquer le dispositif où doit être déployé le PDP. Selon le graphe de la Figure 5.19, le PDP de la session *designers_s* doit être déployé dans le dispositif identifié par l'adresse IP 140.93.65.6.

25. designers_s est la valeur de la propriété de type de donnée hasSessionName de l'individu Designers_Session

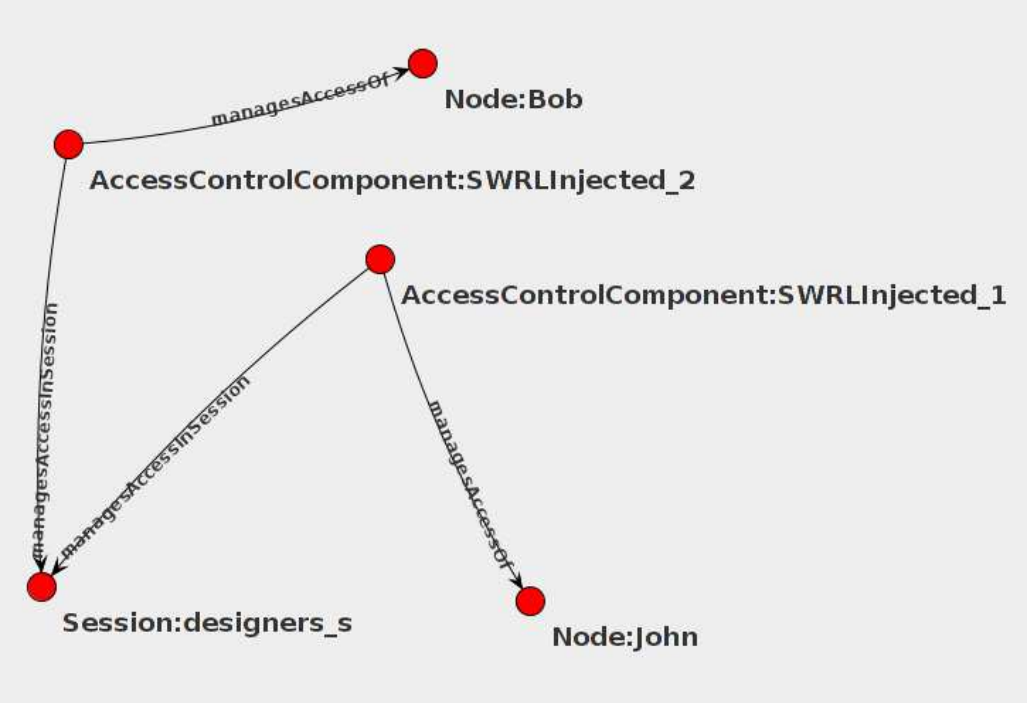

FIGURE 5.18 – *Étape 2 - nouveaux individus créés*

FIGURE 5.19 – *Étape 2 - graphe de contrôle d'accès*

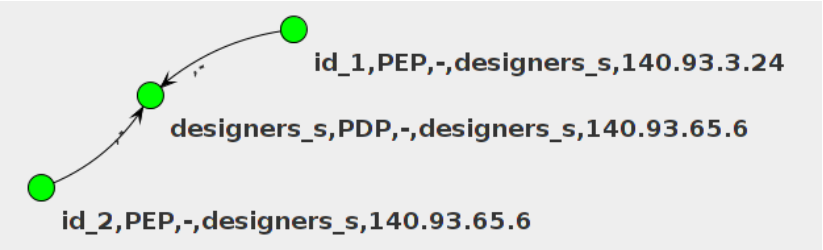

FIGURE 5.20 – *Étape 2 - graphe de déploiement initial*

La Figure 5.20 montre le graphe de déploiement initial généré par le GMTE. D'après le graphe, un PEP doit être déployé dans le dispositif de John (140.93.3.24) et deux composants (un PEP et un PDP) doivent être déployés dans le dispositif de Bob (140.93.65.6). Afin de pouvoir utiliser le gestionnaire de déploiement, un fichier de déploiement GraphML est généré en suivant les étapes décrites dans le diagramme d'activités de la Figure 4.15. La Figure 5.21 montre une partie du fichier de déploiement qui représente les composants déployés dans le dispositif de Bob (140.93.65.6). Comme le montre la figure, le PDP est en écoute des PEPs sur le port 6002 (attribut

configContent). L'attribut *resources_policies* du PDP indique le répertoire qui héberge les politiques de contrôle d'accès. Le PEP déployé dans le même dispositif utilise l'attribut *configContent* afin de pouvoir se connecter au PDP et lui envoyer les requêtes d'accès aux ressources. Il utilise l'attribut *resources_policies* afin de pouvoir accéder à la liste des ressources associées à la session *designers_s*.

```
<node id="140.93.65.6">
  <data key="ip">140.93.65.6</data>
  <data key="port">5000</data>
  <data key="device_type">linux</data>
  <graph id="140.93.65.6:" edgedefault="directed">
    <node id="140.93.65.6::PEP_140.93.65.6_designers_s">
      <data key="bundleID">PEP_140.93.65.6_designers_s</data>
      <data key="action">deployAndStart</data>
      <data key="tech">SOCKET</data>
      <data key="comm">XACML</data>
      <data key="type">PEP</data>
      <data key="configContent">client.session=designers_s;client.ip=140.93.65.6;client.port=6002</data>
      <data key="resources_policies">URL_resources_1</data>
      <data key="configFile">./config/config</data>
    </node>
    <node id="140.93.65.6::PDP_140.93.65.6_designers_s">
      <data key="bundleID">PDP_140.93.65.6_designers_s</data>
      <data key="action">deployAndStart</data>
      <data key="tech">SOCKET</data>
      <data key="comm">XACML</data>
      <data key="type">PDP</data>
      <data key="configContent">server.session=designers_s;server.port=6002</data>
      <data key="resources_policies">URL_policies_1</data>
      <data key="configFile">./config/config</data>
    </node>
  </graph>
</node>
```

FIGURE 5.21 – *Étape 2 - composants de contrôle d'accès déployés dans le dispositif de Bob*

Le déploiement d'un PDP est transparent aux participants. Tandis que le déploiement d'un PEP fait apparaitre une interface utilisateur. Les figures Figure 5.22 et Figure 5.23 montre les interfaces utilisateur déployées respectivement dans les dispositifs de *John* et de *Bob*.

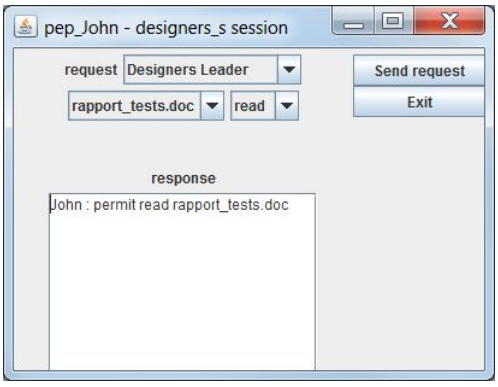

FIGURE 5.22 – *Étape 2 - PEP déployé dans le dispositif de John*

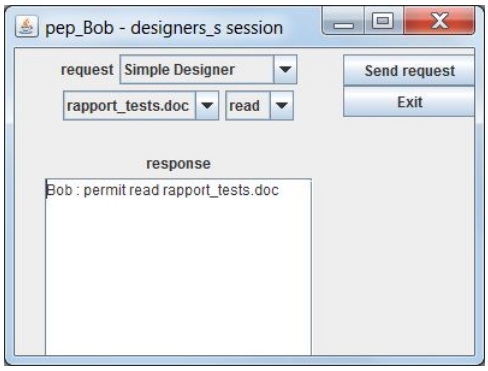

FIGURE 5.23 – *Étape 2 - PEP déployé dans le dispositif de Bob*

Une fois les deux PEPs sont déployés, John et Bob peuvent accéder aux ressources associées à la session *designers_s* en envoyant des requêtes au PDP. Dans chaque requête, ils peuvent choisir la ressource et l'action à effectuer et ensuite cliquer sur le bouton *Sendrequest*. La réponse est ensuite retournée par le PDP et affichée dans le champ de texte appelé *response*.

Afin d'assurer la communication entre John et Bob dans la session *designers_s*, le processus d'adaptation du framework de communication permet de déployer les composants de type EP, EC et CM. Dans cette étape, un EP et un EC sont déployés dans le dispositif de John (140.93.3.24), et un EP et un EC sont déployés dans le dispositif de Bob (140.93.65.6). Ces composants échangent les messages à travers un CM qui gère l'échange de messages dans la session. Les figures Figure 5.24 et Figure 5.25 montrent les résultats de déploiement des composants de communication dans les deux dispositifs. Une fois le déploiement est effectué, John et Bob peuvent échanger des messages et communiquer dans la session *designers_s*.

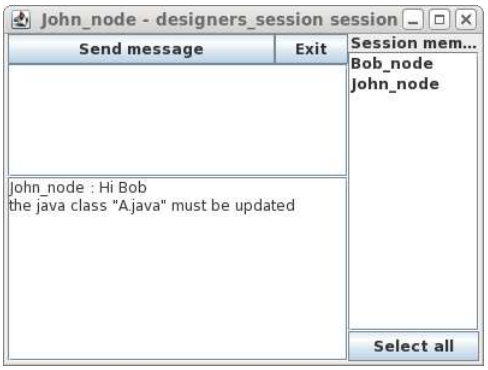

FIGURE 5.24 – *Étape 2 - interface de communication dans le dispositif de John*

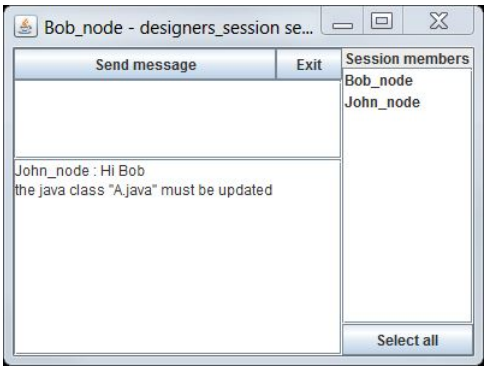

FIGURE 5.25 – *Étape 2 - interface de communication dans le dispositif de Bob*

Étape 3

Cette étape correspond à la connexion du participant Tom au groupe *workGroupB* en tant que *SimpleDesigner* et *TestDeveloper*. L'instance de l'ontologie de domaine, représentée dans la Figure 5.26, ajoute à l'instance de l'étape 2 le nœud (*Tom*), ses deux rôles (*SD*2 et *TD*1) et son point d'accès (*Tom_AP*).

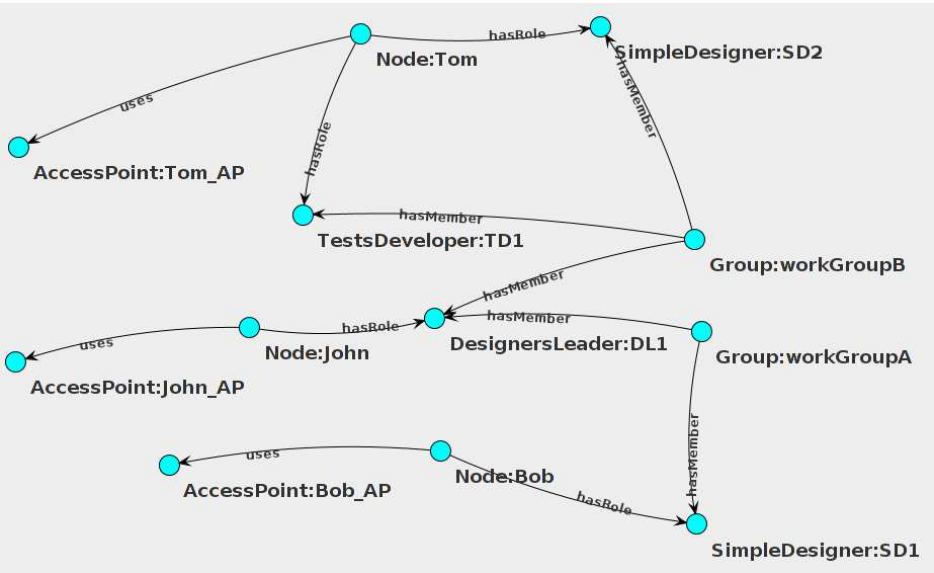

FIGURE 5.26 – *Étape 3 - instance de l'ontologie de domaine*

Dans cette étape, la session *DesignerLead_Developer_Session* doit être établie afin de permettre la communication entre John (en tant que *DesignersLeader*) et Tom en tant que *TestDeveloper*. Le Tableau 5.6 montre les participants dans chaque session des deux groupes.

Groupe	**Sessions**	Participants
workGroupA	Designers_Session	John (Designers Leader) + Bob (Simple Designer)
	DesignerLead_Integrator_Session	
workGroupB	Integrator_Developer_Session	
	DesignerLead_Developer_Session	John (Designers Leader) + Tom (Test Developer)

Tab. 5.6 – *Étape 3 - état des sessions dans chaque groupe*

Le processus d'adaptation du framework de contrôle d'accès applique les règles SWRL à cette instance et produit une nouvelle instance de GACO qui contient de nouveaux individus de la classe *AccessControlComponent*. Ces individus, illustrés dans la Figure 5.27, représentent les composants qui gèrent les accès aux ressources dans les dispositifs des participants. Dans cette figure, on trouve les individus qui ont été créés dans l'étape 2 (Figure 5.18) et deux nouveaux individus *SWRLInjected_1* et *SWRLInjected_4* qui gèrent respectivement les accès de John et Tom dans la session *DesignerLead_Developer_Session*.

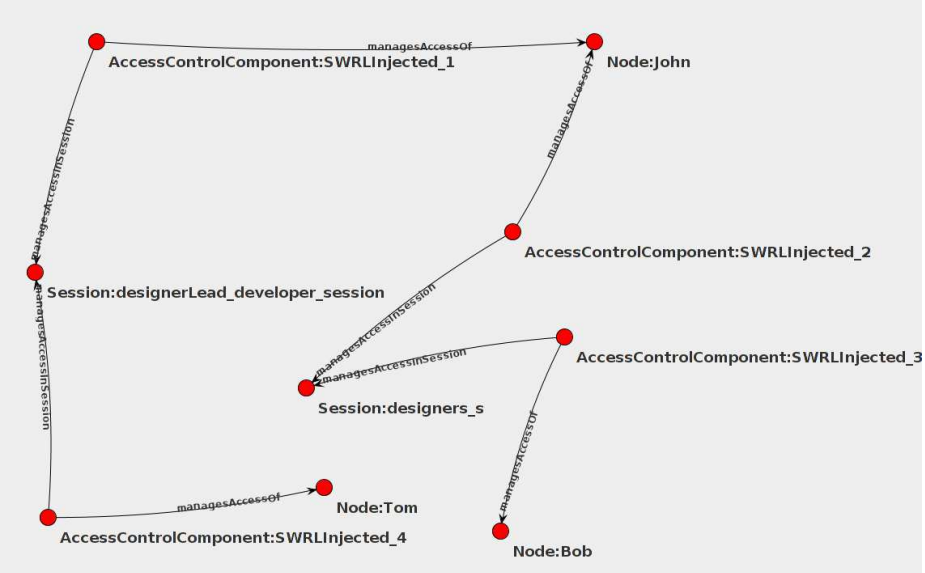

FIGURE 5.27 – *Étape 3 - nouveaux individus créés*

Tout comme dans l'étape 2, les individus de type *AccessControlComponent* sont extraits de cette instance pour former le graphe de contrôle d'accès qui sera le graphe d'entrée pour le GMTE. Les nœuds de ce graphe sont représentés dans la Figure 5.28. Par rapport à l'étape 2, deux nouveaux nœuds

ont été créés (*id*_1 et *id*_4) afin de gérer l'accès des dispositifs de John et de Tom dans la nouvelle session.

FIGURE 5.28 – *Étape 3 - graphe de contrôle d'accès*

La Figure 5.29 montre le graphe de déploiement initial généré par le GMTE. En plus des composants déployés dans l'étape 2, trois nouveaux composants doivent ainsi être déployés afin de contrôler l'accès aux ressources dans la nouvelle session *DesignerLead_Developer_Session*. Selon le graphe, un PEP doit être déployé dans le dispositif de John (140.93.3.24) et deux composants (un PEP et un PDP) doivent être déployés dans le dispositif de *Tom* (140.93.2.140). Afin de pouvoir utiliser le gestionnaire de déploiement, un fichier de déploiement GraphML est généré en suivant les étapes décrites dans le diagramme d'activités de la Figure 4.15. Dans cette étape, les fonctions $ModifiedSessions()$, $ApplyAndConcatenate()$ et $Delta()$ sont utilisées puisqu'il ne s'agit pas du premier déploiement à effectuer. Ces fonctions permettent de tenir compte des déploiements qui sont déjà effectués dans l'étape 2. En conséquence, le fichier GraphML de déploiement généré par le processus d'adaptation ne contient que la description des nouveaux composants qui doivent être déployés.

FIGURE 5.29 – *Étape 3 - graphe de déploiement initial*

Le résultat de déploiement des composants de contrôle d'accès est illustré par les figures Figure 5.30 et Figure 5.31.

La communication entre John et Tom est assuré par le processus d'adaptation du framework de communication. Les figures Figure 5.32 Figure 5.33 montrent les résultats de déploiement des composants de communication dans les dispositifs de John et Tom. Une fois le déploiement est effectué, *John* et *Tom* peuvent échanger des messages dans la session *DesignerLead_Developer_Session*.

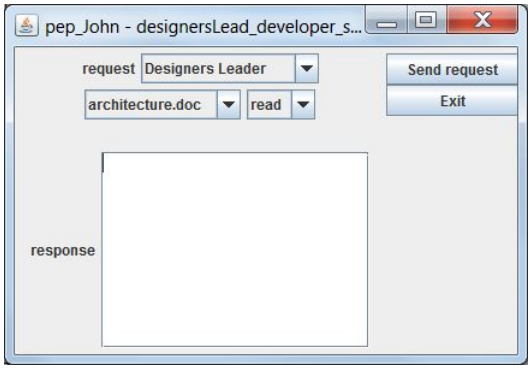

FIGURE 5.30 – *Étape 3 - PEP déployé dans le dispositif de John*

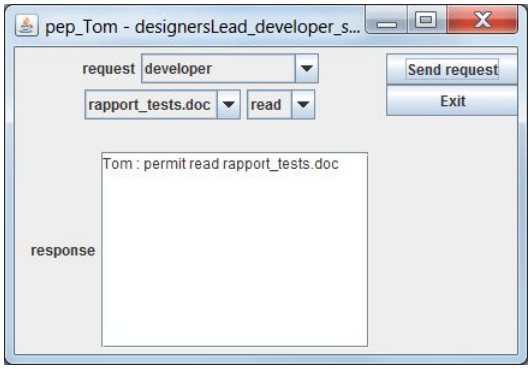

FIGURE 5.31 – *Étape 3 - PEP déployé dans le dispositif de Tom*

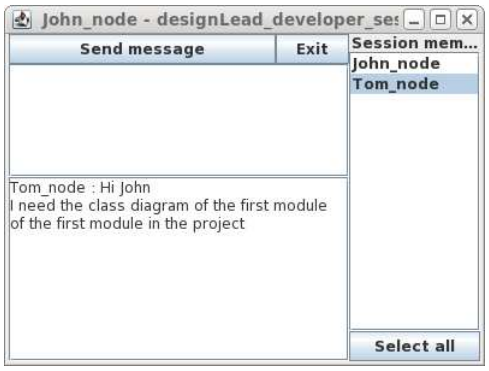

FIGURE 5.32 – *Étape 3 - interface de communication dans le dispositif de John*

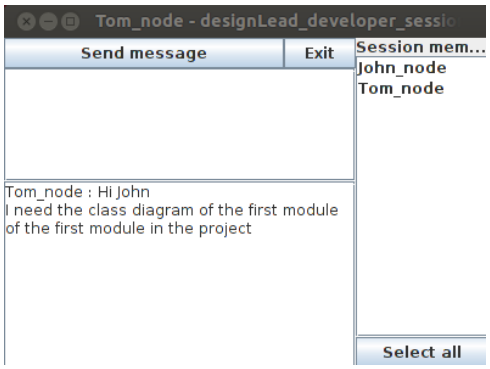

FIGURE 5.33 – *Étape 3 - interface de communication dans le dispositif de Tom*

Étape 4

Dans cette étape, Bob active le rôle *DeploymentManager*. L'instance de l'ontologie de domaine, représentée dans la Figure 5.34, ajoute à l'instance de l'étape 3 le rôle $DM1$ au nœud Bob. Dans cette étape, les graphes de contrôle d'accès et le graphe de déploiement sont les mêmes qui ont été générés lors de l'étape 3 puisqu'il n y a pas une session qui implique le rôle *DeploymentManager* dans le groupe $workGroupA$. En conséquence, il n'y a pas d'actions de déploiement à effectuer dans cette étape et l'état des composants qui ont été déjà déployés reste le même.

Étape 5

Dans cette étape, Bob désactive son rôle *SimpleDesigner*. L'instance de l'ontologie de domaine, représentée dans la Figure 5.35, supprime de l'instance de l'étape 4 le rôle $SD1$ du nœud Bob.

Dans cette étape, il n'y a que John dans la session des concepteurs (*Designers_Session*). Cette session doit être donc supprimée. En conséquence, tous les éléments qui correspondent à cette session doivent être supprimés. Le Tableau 5.7 montre les participants de chaque session dans les deux groupes de travail dans cette étape.

L'instance de GACO produite suite à l'exécution des règles infère les individus représentés dans la Figure 5.36. Ils représentent les composants qui gèrent les accès aux ressources dans les dispositifs de John et Tom dans la session *DesignerLead_Developer_Session*.

Le graphe de déploiement généré par le GMTE à partir du graphe de

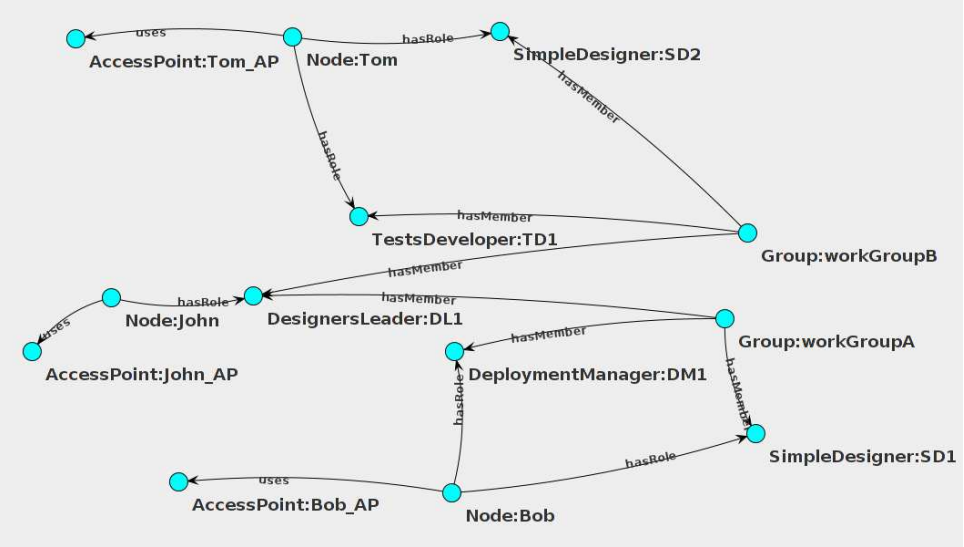

FIGURE 5.34 – *Étape 4 - instance de l'ontologie de domaine*

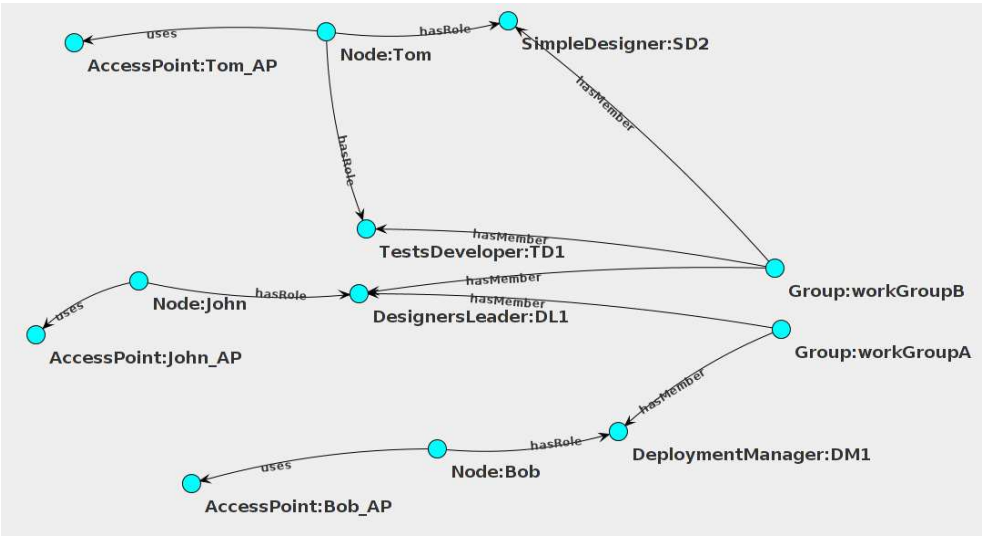

FIGURE 5.35 – *Étape 5 - instance de l'ontologie de domaine*

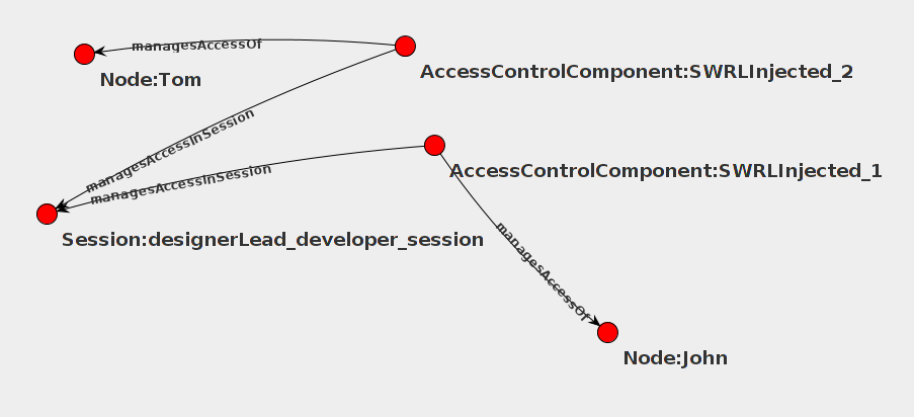

FIGURE 5.36 – *Étape 5 - nouveaux individus créés*

Groupe	Sessions	Participants
workGroupA	Designers_Session	
	DesignerLead_Integrator_Session	
workGroupB	Integrator_Developer_Session	
	DesignerLead_Developer_Session	John (Designers Leader) + Tom (Test Developer)

Tab. 5.7 – *Étape 5 - état des sessions dans chaque groupe*

contrôle d'accès est illustré par la Figure 5.37. Les trois composants décrits dans ce graphe ont été déjà déployés dans l'étape 3. Dans cette étape, le gestionnaire de déploiement effectue des actions de désinstallation des composants de contrôle d'accès et de communication qui correspondent à la session des concepteurs (*Designers_Session*). Les composants qui doivent être désinstallés sont les composants représentés dans les figures Figure 5.22, Figure 5.23, Figure 5.24 et Figure 5.25. La Figure 5.38 montre la partie du fichier GraphML de déploiement qui correspond au dispositif de Bob (140.93.65.6). Nous remarquons que les composants PEP et PDP qui ont été installés et démarrés lors de la phase 2 sont désinstallés dans cette phase grâce à la valeur de l'attribut *action*.

FIGURE 5.37 – *Étape 5 - graphe de déploiement initial*

Étape 6

Dans cette étape, Bob change son rôle *DeploymentManager* par le rôle *IntegrationManager*. Dans l'instance de l'ontologie de domaine (Figure 5.39), le rôle de Bob ($DM1$) est remplacé par le rôle $IM1$. Dans cette étape, le nouveau rôle $IM1$ de Bob lui permet de communiquer avec John dans la session *DesignerLead_Integrator_Session* du *workGroupA*. Le Tableau 5.8 montre les participants de chaque session dans les deux groupes dans cette étape.

L'exécution des règles sur la nouvelle instance produit des individus (Figure 5.40) de la classe *AccessControlComponent* qui gèrent l'accès des nœuds dans les deux sessions. Les individus *SWRLInjected_3* et *SWRLInjected_1* gèrent respectivement les accès de John et de Bob dans la nouvelle session *DesignerLead_Integrator_Session*.

Le graphe de déploiement généré par le GMTE à partir du graphe de contrôle d'accès est illustré par la Figure 5.41. Les composants de contrôle d'accès de la session *DesignerLead_Developer_Session* sont déjà déployés dans les dispositifs de John (140.93.32.24) et Tom (140.93.2.140 lors de l'étape 3. Les nouveaux composants qui doivent être déployés sont : un PEP et un PDP dans le dispositif de John (140.93.32.24) et un PEP dans le dispositif de Bob (140.93.65.6). Le fichier graphML de déploiement qui correspond

```
<node id="140.93.65.6">
  <data key="ip">140.93.65.6</data>
  <data key="port">5000</data>
  <data key="device">linux</data>
  <graph id="140.93.65.6:" edgedefault="directed">
    <node id="140.93.65.6::PEP_140.93.65.6_designers_s">
      <data key="bundleID">PEP_140.93.65.6_designers_s</data>
      <data key="action">uninstall</data>
      <data key="tech">SOCKET</data>
      <data key="comm">XACML</data>
      <data key="type">PEP</data>
      <data key="configContent">-;140.93.65.6;6002;designers_s</data>
      <data key="resources_policies">URL_resources_1</data>
      <data key="configFile">./config/config</data>
    </node>
    <node id="140.93.65.6::PDP_140.93.65.6_designers_s">
      <data key="bundleID">PDP_140.93.65.6_designers_s</data>
      <data key="action">uninstall</data>
      <data key="tech">SOCKET</data>
      <data key="comm">XACML</data>
      <data key="type">PDP</data>
      <data key="configContent">designers_s;6002</data>
      <data key="resources_policies">URL_policies_1</data>
      <data key="configFile">./config/config</data>
    </node>
  </graph>
</node>
```

FIGURE 5.38 – *Étape 5 - partie du fichier de déploiement*

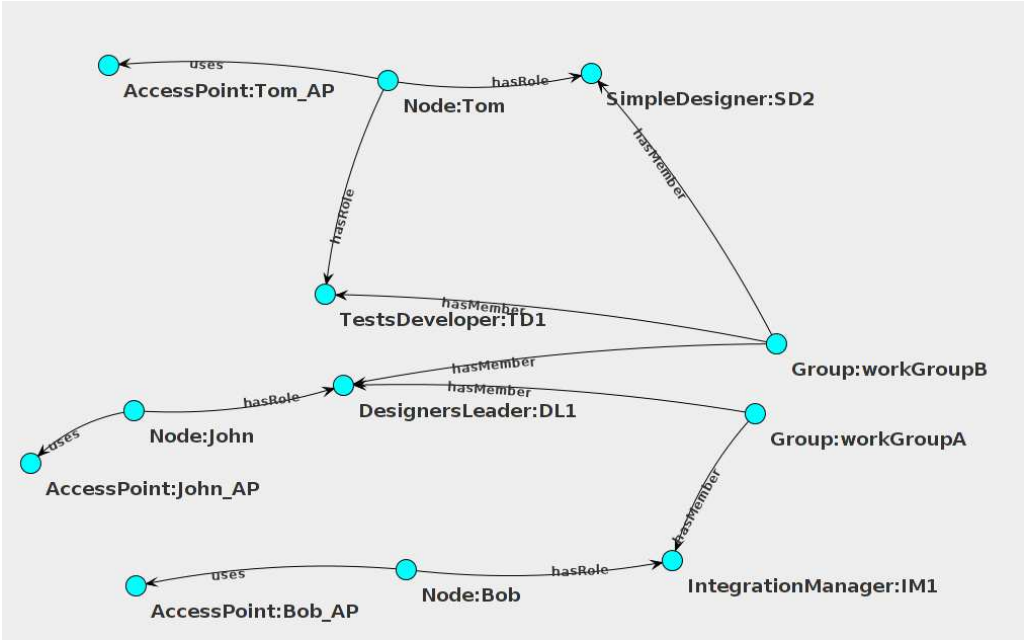

FIGURE 5.39 – *Étape 6 - instance de l'ontologie de domaine*

à cette étape décrit des actions *deployAndStart* pour les trois composants à déployer.

Tout comme dans les étapes 2 et 3, le déploiement des composants de contrôle d'accès fait apparaître les interfaces utilisateur (Figure 5.42 et Figure

Groupe	Sessions	Participants
workGroupA	Designers_Session	
	DesignerLead_Integrator_Session	John (Designers Leader) + Bob (Integration Manager)
workGroupB	Integrator_Developer_Session	
	DesignerLead_Developer_Session	John (Designers Leader) + Tom (Test Developer)

Tab. 5.8 – *Étape 6 - état des sessions dans chaque groupe*

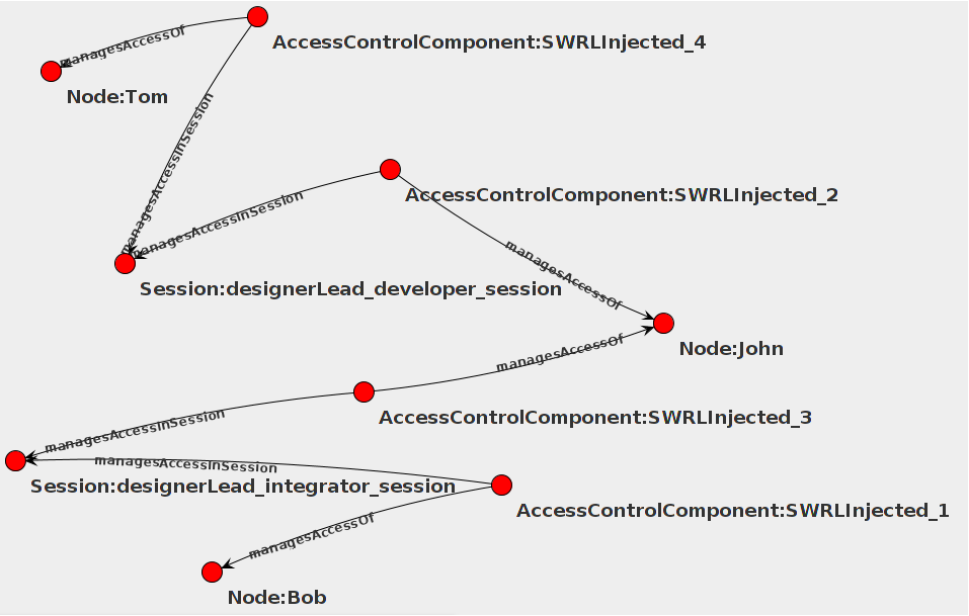

FIGURE 5.40 – *Étape 6 - nouveaux individus créés*

FIGURE 5.41 – *Étape 6 - graphe de déploiement initial*

5.43) qui permettent à John et Bob d'envoyer des requêtes d'accès à des ressources dans la session *DesignerLead_Integrator_Session*.

Le résultat de déploiement des composants de communication est illustré par les figures Figure 5.44 et Figure 5.45. Ils permettent à John et Bob de communiquer dans la nouvelle session *DesignerLead_Integrator_Session*.

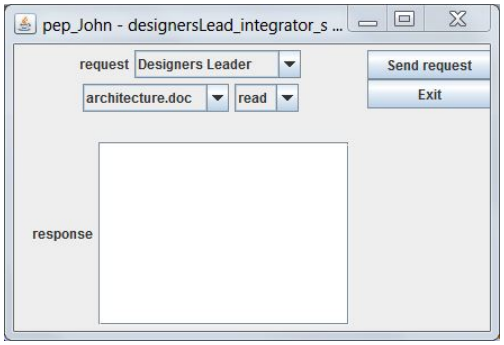

FIGURE 5.42 – *Étape 6 - PEP déployé dans le dispositif de John*

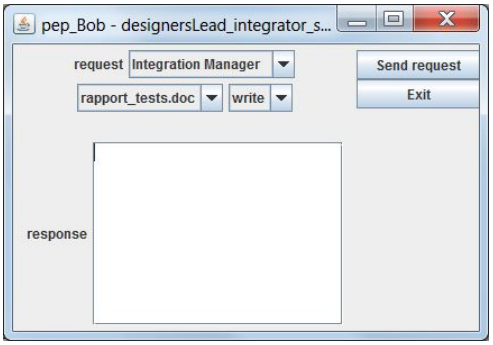

FIGURE 5.43 – *Étape 6 - PEP déployé dans le dispositif de Bob*

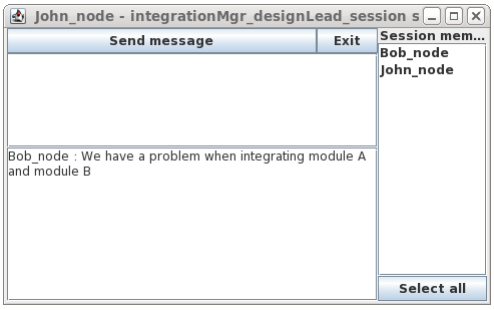

FIGURE 5.44 – *Étape6 - interface de communication dans le dispositif de John*

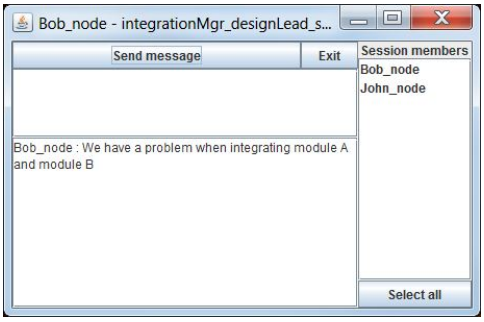

FIGURE 5.45 – *Étape6 - interface de communication dans le dispositif de Bob*

Étape 7

Dans cette étape, Bob décide de rejoindre le groupe *workGroupB*. L'instance de l'ontologie de domaine, représentée dans la Figure 5.46, ajoute à l'instance de l'étape 6 une instance de la relation *hasMember* entre le rôle *IM*1 de Bob et le groupe *workGroupB*. Le fait que le groupe *workGroupB* possède une session appelée *Integrator_Developer_Session*, permet d'établir une session de communication entre Bob (en tant que *IntegrationManager*) et Tom (en tant que *TestDeveloper*).

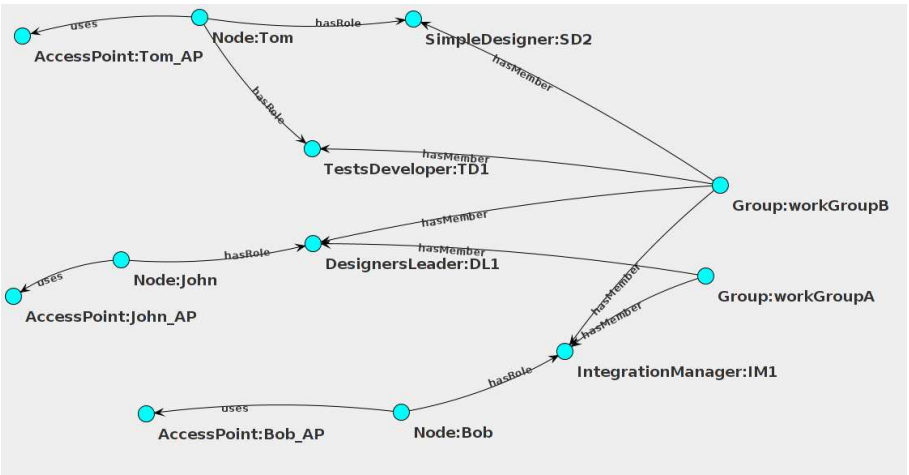

FIGURE 5.46 – *Étape 7 - instance de l'ontologie de domaine*

Le Tableau 5.9 montre les participants de chaque session dans les deux

groupes dans cette étape. Nous remarquons que trois sessions sont établies dans cette étape.

Groupe	**Sessions**	Participants
workGroupA	Designers_Session	
	DesignerLead_Integrator_Session	John (Designers Leader) + Bob (Integration Manager)
workGroupB	Integrator_Developer_Session	Bob (Integration Manager) + Tom (Test Developer)
	DesignerLead_Developer_Session	John (Designers Leader) + Tom (Test Developer)

Tab. 5.9 – *Étape 7 - état des sessions dans chaque groupe*

L'exécution des règles SWRL produit six individus de la classe *AccessControlComponent* (Figure 5.47). Puisque chaque nœud appartient à deux sessions, deux individus sont créés pour chaque nœud.

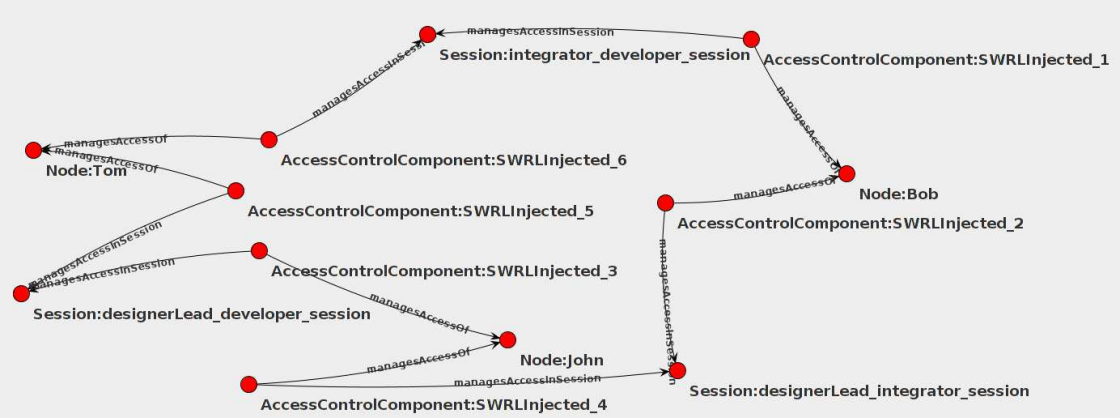

FIGURE 5.47 – *Étape 7 - nouveaux individus créés*

Le graphe de déploiement généré par le GMTE est illustré par la Figure 5.48. Les composants de contrôle d'accès qui doivent être déployés dans cette étape sont : un PEP et un PDP dans le dispositif de Tom (140.93.2.140) et un PEP dans le dispositif de Bob (140.93.65.6). Le résultat de déploiement de ces composants est illustré par les figures Figure 5.49 et Figure 5.50.

Le processus d'adaptation du framework de communication permet de déployer les composants de type EP, EC et CM dans les dispositifs de Bob et Tom. Le résultat de déploiement est illustré par les figures Figure 5.51 et Figure 5.52.

Étape 8

Dans cette dernière étape du scénario, Bob quitte le groupe *workGroupB* qu'il a rejoint dans l'étape 7. L'instance de l'ontologie de domaine produite dans cette étape est similaire à celle qui a été produite lors de

FIGURE 5.48 – *Étape 7 - graphe de déploiement initial*

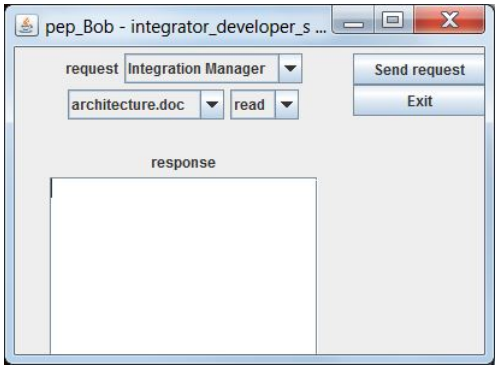

FIGURE 5.49 – *Étape 7 - PEP déployé dans le dispositif de Bob*

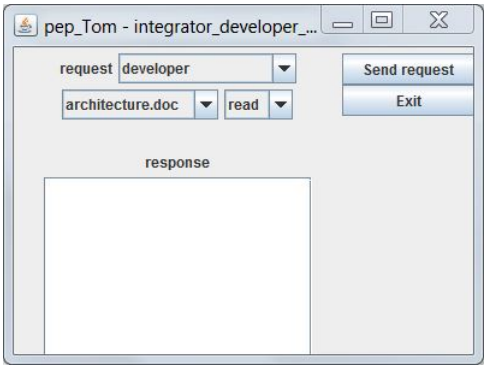

FIGURE 5.50 – *Étape 7 - PEP déployé dans le dispositif de Tom*

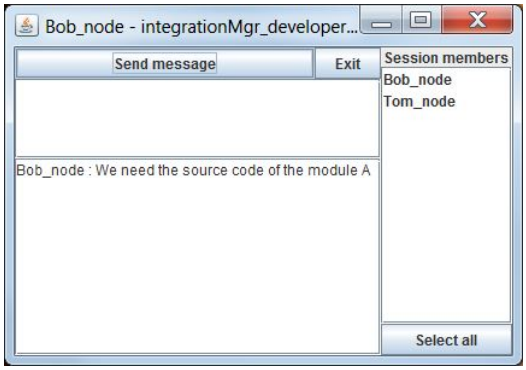

FIGURE 5.51 – *Étape7 - interface de communication dans le dispositif de Bob*

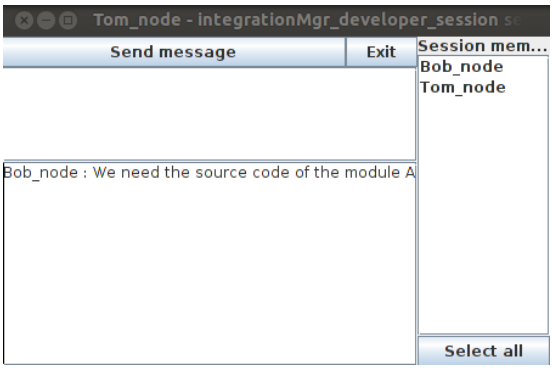

FIGURE 5.52 – *Étape7 - interface de communication dans le dispositif de Tom*

l'étape 6 (Figure 5.39). A la fin des processus d'adaptation, les composants de communication et de contrôle d'accès qui correspondent à la session *Integrator_Developer_Session* sont arrêtés et désinstallés des dispositifs de Bob et de Tom.

5.5 Évaluation des performances

Dans cette sous section, nous présentons une évaluation des performances du famework de contrôle d'accès et des composants de contrôle d'accès.

Dans la première partie, nous nous concentrons sur le temps de réponse du processus d'adaptation du framework de contrôle d'accès suite aux changements qui ont lieu durant l'exécution de l'application.

Dans la deuxième partie de cette sous-section, nous effectuons une évaluation du temps de génération des décisions d'autorisation d'accès. Nous montrerons l'avantage de décentralisation des entités de contrôle d'accès de type PDP sur le temps de réponse.

5.5.1 Temps d'exécution du processus d'adaptation

Afin d'évaluer les performances du framework de contrôle d'accès, nous utilisons un scénario illustré par le Tableau 5.10. Dans ce scénario, nous considérons un ensemble de participants répartis sur deux groupes. Deux sessions de collaboration sont associées pour chaque groupe. Dans le Tableau 5.10, nous indiquons le nombre de participants dans chaque phase du scénario. Chaque phase correspond à un ensemble de changements qui nécessite l'exécution du processus d'adaptation. Nous avons définis des changements qui affectent toutes les sessions dans chaque phase du scénario. Ceci permet de rendre l'évaluation plus significative puisque le processus d'adaptation traite les sessions d'une façon indépendante. En effet, si les changements n'affectent qu'une seule session $s1$, alors le processus d'adaptation effectue des adaptations seulement pour cette session $s1$. En conséquence, le temps de réponse ne dépendra que du nombre de participants dans la session concernée.

Numéro de phase	1	2	3	4	5	6
Nombre de participants	5	8	17	26	35	38

Tab. 5.10 – *Scénario pour l'évaluation des performances*

Afin de pouvoir mesurer les temps de réponse, nous avons créé les instances qui correspondent à chacune des six phases. Dans chaque instance, nous avons simulé l'existence des dispositifs des participants en attribuant une adresse IP à chaque individu de la classe *Access Point*. Les instances sont ensuite fournies à une classe de test qui lance un processus d'adaptation à partir de chaque instance créée. Durant l'évaluation, nous avons lancé 5 fois chaque processus d'adaptation d'une façon séquentielle dans des threads séparés. Nous avons ensuite calculé la moyenne du temps de réponse de chaque processus pour chaque phase du scénario.

Dans chacune des phases du scénario, nous avons mesuré trois temps :

- t_1 : le temps de génération de l'instance de l'ontologie de contrôle d'accès par le générateur des besoins de contrôle d'accès (*Access Control Requirements Generator*). Ce temps représente le temps de chargement de l'instance et le temps d'application des règles SWRL.
- t_2 : le temps de génération du graphe de contrôle d'accès.

– t_3 : le temps de génération du fichier de déploiement. Ce temps représente le temps de génération du graphe de déploiement initial par le GMTE avec le temps de génération du fichier de déploiement utilisé par le gestionnaire de déploiement.

Dans cette évaluation, nous avons utilisé une machine *hp* 64-bit, dotée d'un processeur *Intel® Xeon(R)* 3.20GHz, avec une mémoire vive de 8 Go. Le système d'exploitation utilisé est *Ubuntu 12.04 LTS*.

Résultats

La Figure 5.53 montre les temps t_1, t_2 et t_3 de chaque processus d'adaptation pour chacune des phases du scénario. Nous représentons sur l'axe horizontal le nombre de participants qui correspond à chaque phase. Comme le montre la figure, le temps t_1 est le temps prédominant dans les quatre premières phases. En effet, dans ces quatre phases, le graphe de contrôle d'accès contient moins de 26 nœuds représentant les individus (de type *AccessControlComponent*) associés aux participants. En conséquence, la génération du graphe de déploiement par le moteur de transformation de graphe est rapide et t_3 reste toujours petit par rapport à t_1.

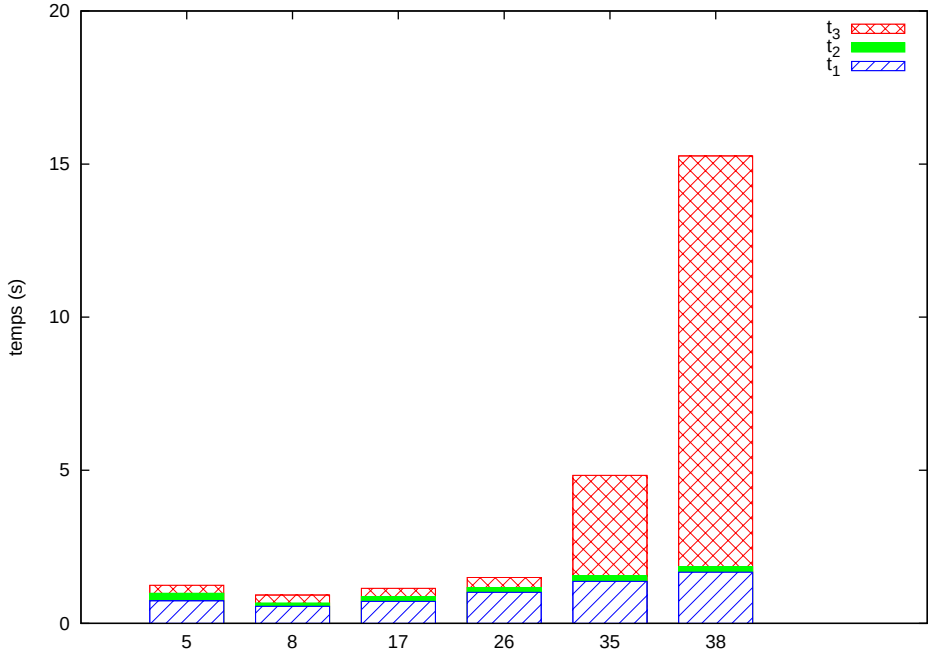

FIGURE 5.53 – *Temps de réponse total*

Dans les phases 5 et 6, la taille du graphe de contrôle d'accès est plus importante (>35), ce qui fait que le moteur de transformation de graphe met beaucoup plus de temps pour la génération du graphe de déploiement. Ceci est causé par l'application récursive des règles définies dans la grammaire de graphe GG_{simple} sur le graphe de contrôle d'accès. Nous remarquons qu'à partir de 35 participants, le moteur de transformation de graphe devient lent, ce qui ralentit le déploiement physique des composants de contrôle d'accès dans les dispositifs des participants.

La Figure 5.54 montre les temps t_1 et t_2 pour chacune des phases. La somme de ces deux temps représente le temps total de génération du

graphe de contrôle d'accès. Nous remarquons que le temps t_2 reste toujours inférieur à 500 milli secondes dans toutes les phases indépendamment du nombre d'individus. Cependant, le temps t_1 augmente en fonction du nombre d'individus mais reste toujours inférieur à 2 secondes dans toutes les phases.

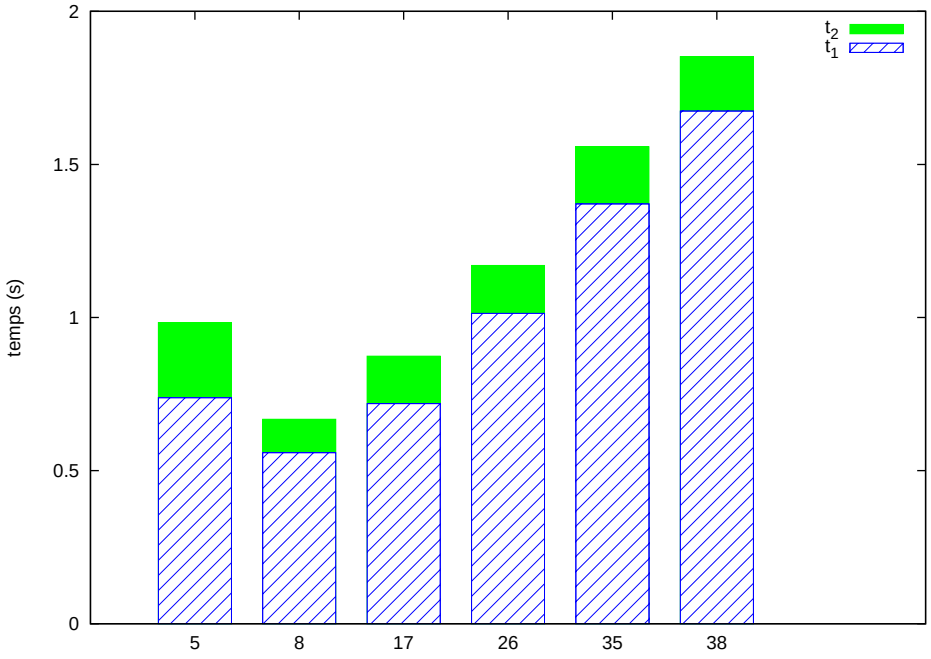

FIGURE 5.54 – *Temps de génération du graphe de contrôle d'accès*

5.5.2 Temps de génération des décisions d'autorisation

Afin d'évaluer les performances des entités de contrôle d'accès, nous mesurons le temps de génération des décisions d'autorisation par l'ensemble des points de décision de politiques (ou PDP) déployés dans les dispositifs des participants. Le but de cette évaluation est de comparer notre approche de décentralisation du contrôle d'accès avec l'approche classique centralisée.

Pour mesurer les temps de réponse des PDPs, nous avons créé une classe qui instancie deux PDPs et lance les processus de prise de décision dans des threads séparés. Nous avons simulé l'existence de plusieurs dispositifs, répartis dans deux sessions, qui envoient des requêtes d'accès à une ressource simultanément. Dans les applications réelles, les PDPs sont généralement répartis dans des dispositifs différents et le fait de les démarrer dans un seul dispositif rend le temps de réponse plus lent puisque les deux PDP partagent le processeur du même dispositif. Ce qui fait que les résultats réels sont meilleurs que ceux que nous présentons ici.

Dans cette évaluation, nous avons exécuté les processus de prise de décision 10 fois et nous avons ensuite calculé les valeurs moyennes des temps mesurés. Nous avons commencé par une première expérimentation qui calcule le temps moyen de prise de décision d'un seul PDP, ensuite nous l'avons fait pour deux PDPs. Pour l'exécution de ces tests, nous avons utilisé une

machine SONY VAIO 64 bit, dotée d'un processeur *Intel(R) Core(TM) i7* 2.20 GHz, avec une mémoire 8 Go.

La Figure 5.55 montre les résultats de notre expérimentation. Elle montre les temps de génération des décisions d'autorisation en fonction du nombre de requêtes. L'axe horizontal de la figure représente le nombre de requête que nous font augmenter à partir de 100. Tandis que l'axe vertical représente le temps génération des décisions en milli secondes. Les deux courbes de la figure représentent les temps t4 et t5 décrits comme suit :

- t_4 : le temps de génération des décisions dans le cas où un seul PDP est utilisé dans deux sessions. Ce temps est représenté par la courbe bleue.
- t_5 : le temps de génération des décisions dans le cas où deux PDPs différents sont utilisés dans chacune des deux sessions. Ce temps est représenté par la courbe rouge.

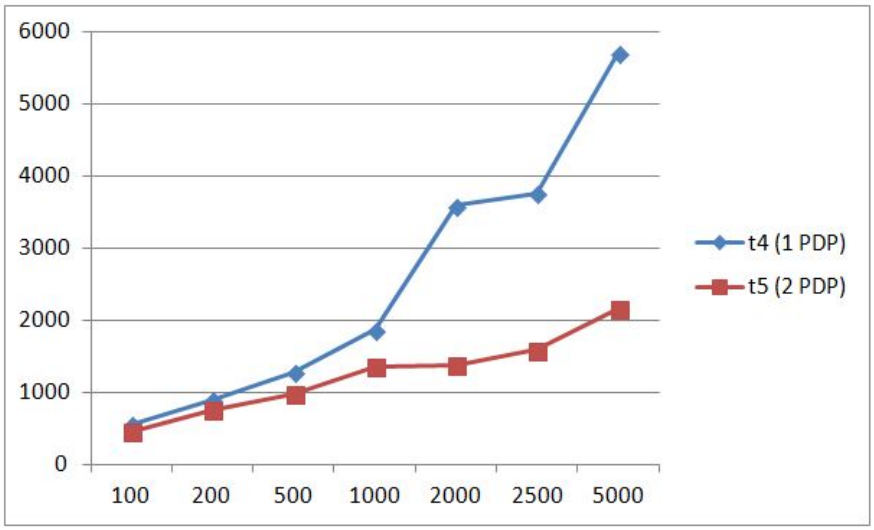

FIGURE 5.55 – *Temps de génération des décisions d'autorisation*

Nous remarquons que les deux temps t_4 et t_5 sont quasiment les mêmes lors du traitement d'un petit nombre de requêtes d'accès. En effet, les deux courbes restent adjacentes pour un nombre de requêtes inférieur à 1000. A partir de 1000 requêtes, la courbe rouge augmente d'une façon quasi régulière et ralentie, alors que la courbe bleue augmente très rapidement. En effet pour 5000 requêtes, la valeur de t_4 est 5700 ms ; tandis que la valeur de t_5 est 2100 ms.

Ces résultats montrent que le temps de génération des décisions d'autorisation est toujours meilleur dans une architecture décentralisée. Ceci est justifié par le fait que les requêtes sont traitées par deux entités de contrôle. Comme le montre la figure, cette répartition de charge est de plus en plus avantageuse quand le nombre de requêtes augmente.

L'architecture décentralisée de contrôle d'accès ne permet pas seulement la décentralisation des PDPs mais aussi la décentralisation des politiques. Ceci rend les PDPs beaucoup plus performant puisque chaque PDP ne parcourt qu'une partie des politiques qui lui sont associées.

Conclusion du chapitre

Dans ce chapitre, nous avons présenté une application d'illustration dans le cadre du projet Galaxy. En utilisant cette application, nous avons décrit comment le framework de contrôle d'accès peut être utilisé afin de sécuriser l'accès aux ressources à travers le déploiement adaptatif des composants de contrôle d'accès dans les dispositifs des utilisateurs. Nous avons également montré comment l'ontologie GACO peut être étendue afin d'exprimer les concepts et les relations spécifiques à l'application.

Après la description de l'application, nous avons effectué une évaluation fonctionnelle qui nous a permis de valider le fonctionnement du framework de contrôle d'accès et les techniques de déploiement. Lors de cette évaluation, nous avons détaillé tout le processus d'adaptation effectué après chaque changement dans l'application. Nous avons également validé le processus de déploiement des composants de contrôle d'accès et des composants de communication. Dans cette évaluation, nous avons remarqué que les processus d'adaptation produisent les résultats attendus après chaque évènement.

Dans l'évaluation des performances du processus d'adaptation, nous avons remarqué que le temps de réponse reste raisonnable dans la plupart des cas. Cependant, il dépend fortement du nombre de participants dans le groupe. En effet, l'utilisation du moteur de transformation de graphe pour la génération du graphe de déploiement dégrade le temps de réponse du processus d'adaptation si le nombre de participants est supérieur à 35. Nous avons remarqué que le temps de traitement de l'ontologie reste toujours raisonnable (entre 0,5 et 2 secondes). En conséquence, si le groupe de collaboration de l'application est constitué d'un nombre de participants inférieur à 35, nous pouvons considérer qu'un temps de réponse inférieur ou égal à 5 secondes est acceptable pour que le système puisse s'adapter aux changements.

Quant aux performances des composants de contrôle d'accès, le temps de génération des décisions d'autorisation reste toujours raisonnable. Ce temps dépend du nombre de participants et de l'architecture de contrôle d'accès. Les résultats ont montré que la décentralisation de composants de contrôle d'accès rend le processus de prise de décision beaucoup plus performant (surtout lors de la réception d'un grand nombre de requêtes d'accès. Les composants de contrôle sont répartis en fonction des situations de collaboration. Une bonne conception de l'application et de ses règles de collaboration assure une meilleure répartition des composants et donc de meilleures performances.

Conclusion générale

Dans ce travail, nous avons étudié le contrôle d'accès dans les environnements collaboratifs ubiquitaires qui ont introduit de nouveaux types d'interaction entre les utilisateurs et leurs dispositifs. Nous avons appliqué notre étude dans les environnements de travail collaboratif dont l'un des objectifs est de faciliter la communication et le partage des ressources durant les processus de production. Bien que ce type d'environnement ne tire pas pleinement profit des opportunités offertes par l'informatique ubiquitaire, plusieurs travaux sont en cours d'explorer les techniques qui pourraient être mises à disposition afin de faire évoluer la façon de communiquer et sécuriser l'accès ressources.

Le contrôle d'accès aux ressources est l'un des défis de recherche dans les environnements collaboratifs ubiquitaires, caractérisés par une grande dynamique. Afin que les nouveaux systèmes émergents puissent contrôler l'accès aux ressources, de nouvelles architectures et mécanismes de contrôle sont nécessaires. Ces mécanismes doivent être autonomes et auto-adaptatifs afin d'assurer une adaptation dynamique aux changements du contexte. Dans ce travail, nous nous sommes basés sur les concepts de l'informatique autonomique afin d'assurer une adaptation architecturale des logiciels de contrôle d'accès. Nous avons également utilisé des techniques de représentation des connaissances afin de capturer les exigences de l'application et d'en tenir compte dans le processus d'adaptation.

Contributions apportées

Dans le chapitre 2 de ce manuscrit, nous avons introduit les thématiques sur lesquelles nous nous sommes focalisés dans nos travaux : les environnements collaboratifs ubiquitaires, le contrôle d'accès et la représentation des connaissances. Dans une première partie, nous avons mis l'accent sur la notion de contexte et d'adaptation dans les environnements collaboratifs ubiquitaires. Ensuite, nous avons présenté les modèles de contrôle d'accès et les exigences des mécanismes de contrôle d'accès dans l'informatique ubiquitaire. Nous avons également présenté les concepts de représentation des connaissances et des ontologies OWL. Finalement, nous avons analysé les travaux existants qui traitent le contrôle d'accès dans les environnements collaboratifs ubiquitaires et nous avons positionné nos contributions par rapport à ces travaux. A l'issu de cette analyse, nous avons constaté que peu de travaux proposent une adaptation architecturale des composants décentralisés de contrôle d'accès dans les environnements collaboratifs ubiquitaires. La décentralisation du contrôle est largement étudiée dans les systèmes statiques à grande échelle. Cependant, sa mise en place est statique et indépendante de la dynamique et des besoins des applications qui évoluent dans le temps.

Nous avons présenté nos contributions dans le chapitre 3 et le chapitre 4. Nos contributions visent à satisfaire les exigences de contrôle d'accès suivantes : (1) décentralisation du contrôle, (2) capacité à s'adapter dynamiquement aux changements et (3) facilité d'utilisation dans les environnements de travail collaboratif.

Le chapitre 3 est divisé en trois parties principales : l'architecture de contrôle d'accès, l'ontologie GACO et le framework de contrôle d'accès. Dans la première partie, nous avons expliqué le besoin de décentralisation des mécanismes de contrôle d'accès et son effet dans les systèmes à grande échelle qui traitent un énorme nombre de ressources, d'utilisateurs et de politiques de contrôle d'accès. Nous avons également mis l'accent sur l'effet de décentralisation des politiques dans les environnements collaboratifs. Ensuite, nous nous sommes basés sur les éléments du standard XACML pour définir l'architecture de décentralisation de contrôle d'accès. Nous avons également justifié nos choix de répartition des différentes entités du standard XACML. L'utilisation de ce standard permet une meilleure interopérabilité avec les systèmes collaboratifs. Nous avons bien remarqué l'avantage de cette architecture décentralisée à travers les résultats présentés dans le chapitre 5. Ces résultats montrent que le temps de génération des décisions d'autorisation reste toujours raisonnable même avec l'augmentation du nombre de requêtes traitées par les composants logiciels de contrôle d'accès.

Dans la deuxième partie du Chapitre 3, nous avons présenté le modèle *Generic Access Control Ontology* (GACO). Ce modèle est une ontologie décrite en OWL (*Web Ontology Language*). L'objectif principal de ce modèle est d'exprimer les situations de collaboration et les besoins de contrôle d'accès qui y sont associés. L'avantage de l'utilisation des ontologies OWL pour représenter GACO est la possibilité de raisonnement qui permet d'inférer, à partir d'une instance de GACO, une nouvelle instance qui exprime les besoins de contrôle d'accès en fonction des situations de collaboration. Ceci nous permet de facilité l'application de notre approche de contrôle d'accès dans les environnements de travail collaboratif, et donc satisfaire la troisième exigence que nous avons mentionnée. En effet, les concepteurs des applications collaboratives doivent juste étendre GACO et exprimer les situations de collaboration en utilisant des règles, et c'est le mécanisme de raisonnement qui se charge d'inférer de nouveaux individus qui expriment les besoins de contrôle d'accès.

Dans la troisième partie du Chapitre 3, nous avons présenté notre framework pour l'adaptation de l'architecture logicielle de contrôle d'accès dans les environnements collaboratifs ubiquitaires. Notre framework assure le déploiement et la reconfiguration dynamique des composants logiciels de contrôle d'accès en fonction des exigences haut niveau de l'application. Afin d'effectuer cette adaptation architecturale, nous avons utilisé une boucle de contrôle autonomique qui se base sur GACO et des mécanismes de transformation de graphes pour générer un plan de déploiement des composants de contrôle d'accès. Nous avons détaillé tous les modèles générés au cours du processus d'adaptation, ainsi que les transformations qui assurent les transitions entre les modèles. Après chaque génération d'un plan de déploiement, un processus de déploiement est lancé afin de mettre en œuvre toutes les actions qui doivent être effectuées sur les composants. La mise en place de notre framework permet de satisfaire la deuxième exigence qui représente

la capacité des applications collaboratives de s'adapter dynamiquement aux changements.

Le chapitre 4 a été consacré à la présentation de notre apport pour le déploiement physique des composants logiciels. Ce déploiement est effectué lors de la phase d'exécution de la boucle de contrôle autonomique. Il consiste à démarrer, arrêter ou reconfigurer des composants logiciels en cours d'exécution. Nous avons détaillé les différentes étapes du processus de déploiement, ainsi que la structure des fichiers de déploiement qui doivent être fournis au gestionnaire de déploiement. Dans ce chapitre, nous avons également proposé des algorithmes afin de résoudre les problèmes d'incohérence causés par la nature monotone du langage OWL. Nous avons appliqué le processus et les techniques de déploiement dans deux framework : le framework de contrôle d'accès et le framework de communication. L'utilisation du déploiement dans le framework de contrôle d'accès nous a permis de mettre en place les composants physiques de contrôle d'accès. Tandis que son utilisation dans les framework de communication nous a permis de réaliser physiquement l'adaptation architecturale des composants de communication. Le framework de communication nous a servi, d'une part, comme un cadre d'application du déploiement, et d'autre part, pour assurer la communication dans les environnements collaboratifs ubiquitaires.

Dans le chapitre 5, nous avons évalué le framework de contrôle d'accès et le mécanisme de déploiement des composants. Pour cela, nous avons introduit une application, développée dans le cadre du projet Galaxy, qui utilise le framework de contrôle d'accès et le framework de communication. Nous avons défini un scénario qui nous a servi pour évaluer le fonctionnement des processus d'adaptation après chaque changement au niveau de l'application. Les résultats de l'évaluation fonctionnelle ont montré que les modèles produits sont ceux qui sont attendus dans chaque étape du scénario. Dans la deuxième partie du chapitre, nous avons évalué les performances du framework de contrôle d'accès. Les résultats ont montré que le temps d'exécution du processus d'adaptation reste raisonnable si le nombre de participants est inférieur à 35. L'étape du processus qui dégrade ce temps est la transformation du graphe de collaboration en graphe de déploiement en utilisant le moteur de transformation de graphes. Malgré le coût élevé de cette transformation, l'utilisation du moteur de transformation de graphe nous apporte l'avantage de pouvoir choisir et mettre en place l'architecture de contrôle d'accès désirée en fonction des caractéristiques de l'environnement collaboratif. En effet, l'utilisateur du framework a la possibilité de spécifier la façon dont les composants de contrôle d'accès sont répartis à travers des règles de transformation indépendamment de l'implantation du framework. Quant à l'évaluation des performances des composants de contrôle d'accès, les résultats ont montré que, dans une architecture décentralisée, le temps de génération des décisions d'autorisation reste toujours acceptable (2 secondes) pour un nombre de requêtes égal à 5000.

Perspectives

Notre travail de thèse nous a permis d'aborder les problématiques de contrôle d'accès et de déploiement dans les environnements collaboratifs ubiquitaires. Bien que nous avons présenté une évaluation fonctionnelle et

une évaluation des performances, plusieurs améliorations peuvent être effectuées tant au niveau fonctionnel qu'au niveau de performance. Nous listons ci-dessous quelques perspectives de recherche.

Gestion dynamique des politiques Dans notre travail de thèse, nous nous sommes concentrés sur la reconfiguration dynamique des architectures logicielles pour assurer l'adaptation aux changements du contexte dans les environnements ubiquitaires. Mais, les politiques de contrôle d'accès peuvent aussi changer durant l'activité collaborative. Ceci est géré au niveau des entités de contrôle qui récupèrent périodiquement les politiques de contrôle d'accès d'un serveur FTP (*File Transfer Protocol*) et génère les décisions d'autorisation en se basant sur ces politiques. Cependant la gestion dynamique des politiques n'a pas été traitée puisque l'éditeur UMU-XACML utilisé pour l'édition des politiques ne permet pas de modifier des politiques stockées dans un serveur FTP. L'autre inconvénient de l'utilisation de l'éditeur UMU-XACML est que la définition des politiques n'est pas assez simple et exige une maîtrise préalable des éléments de l'éditeur et du langage XACML. Il serait intéressant de développer un éditeur plus simplifié qui héberge un client FTP permettant de télécharger les politiques, les modifier et les charger dans le serveur FTP.

Vérification de la cohérence des politiques Pour la définition des politiques de contrôle d'accès, nous avons utilisé le langage standard XACML, plus particulièrement le profil RBAC proposé par OASIS. Afin de résoudre l'incohérence des politiques de contrôle d'accès, le langage XACML définit un ensemble d'algorithmes de combinaison qui combinent le résultat de plusieurs politiques et règles pour un même rôle. Bien que ces algorithmes permettent de résoudre les incohérences dans la plupart des cas, ils ne sont pas toutefois suffisants dans des situations complexes où un utilisateur joue plusieurs rôles auxquels sont associés des droits d'accès contradictoires. Par exemple, si un utilisateur joue deux rôles $r1$ et $r2$ tels que $r1$ lui permet de lire un fichier $F1$ et $r2$ l'interdit de le lire. Ce cas n'est pas traité par le langage XACML. Une première idée consisterait à utiliser un seul rôle dans chaque requête et accepter la décision indépendamment des autres droits attribués aux autres rôles de l'utilisateur. Une deuxième possibilité serait de vérifier la cohérence des politiques en utilisant des techniques de raisonnement. Ceci est possible en définissant un modèle sémantique des permissions et en déterminant des ensembles de rôles auxquels sont associés des droits contradictoires à travers des règles. Ensuite deux cas sont possibles : (1) interdire l'attribution des rôles de chaque ensemble à un même utilisateur, (2) ou bien informer l'administrateur des politiques et le notifier des incohérences possibles.

Amélioration des performances du framework de contrôle d'accès Les résultats présentés dans le chapitre 5 montre que le temps de génération des décisions d'autorisation reste raisonnable grâce à la décentralisation des entités de contrôle d'accès. Cependant, le temps d'exécution du processus d'adaptation se dégrade quand la taille du groupe augmente. Ceci est causé par l'utilisation du moteur de transformation de graphe qui applique les règles de transformation d'une manière récursive. Nous avons

optimisé l'utilisation du moteur de transformation de graphes en traitant chaque session d'une manière indépendante et en appliquant les transformations seulement sur les sessions qui ont été modifiées. Toutefois, ceci n'est pas suffisant quand le nombre d'utilisateurs dans la même session augmente. Une première amélioration consisterai à établir de nouvelles sessions de collaboration à partir du moment où le temps de réponse du moteur de transformation de graphe n'est plus raisonnable pour le traitement d'une session donnée. Ceci doit s'effectuer avec l'accord de l'application qui autorise ou non l'établissement de nouvelles sessions. Une autre idée serait de décentraliser l'architecture du framework de contrôle d'accès pour la gestion répartie des architectures de contrôle d'accès. Ceci nous permettrait d'améliorer le passage à l'échelle et d'éviter la centralisation du traitement dans un seul point central.

Utilisation du framework dans d'autres applications Dans notre travail, nous avons utilisé le framework dans le cadre du projet Galaxy qui propose un environnement de travail collaboratif où les participants utilisent des dispositifs mobiles constituant un réseau ad-hoc sans fil. Il serait intéressant de l'utiliser dans d'autres applications qui se caractérisent par un aspect dynamique pour étudier son applicabilité et évaluer son fonctionnement. Nous avons commencé à utiliser nos apports dans le cadre du projet européen Imagine dont l'objectif est de fournir une méthodologie et une plateforme de collaboration pour la gestion des réseaux de production dynamiques (*Dynamic Manufacturing Networks* - DMN). La plateforme du projet Imagine se base sur une architecture SOA (*Service-oriented Architecture*) pour assurer la gestion d'un ensemble de producteurs et de consommateurs de services. Les changements sont dynamiques dans la plateforme. En effet, les producteurs de services peuvent être remplacés par d'autres producteurs de services, ils peuvent aussi participer dans plusieurs processus de production, etc. La gestion d'accès aux ressources par les différents acteurs de la plateforme est compliquée vu l'aspect dynamique de la plateforme. En plus, un niveau de performance élevé des entités de contrôle d'accès est aussi exigé par la plateforme vu le nombre important de ressources et de politiques de contrôle d'accès. Nous avons mis en place un ensemble de services de contrôle d'accès qui sont connectés à la plateforme afin de sécuriser l'accès aux ressources. Les services de contrôle d'accès que nous avons proposés sont répartis sur plusieurs machines afin de décentraliser le contrôle et assurer un passage à l'échelle. Nous avons également mis en place un processus d'orchestration qui se charge du routage des requêtes d'accès aux ressources vers le service de contrôle adéquat en utilisant la norme BPEL (*Business Process Execution Language*). Il serait intéressant d'utiliser le framework de contrôle d'accès pour assurer le déploiement dynamique des services de contrôle d'accès en fonction des besoins de la plateforme. Cela exige des interactions entre le framework et le processus d'orchestration pour s'adapter dynamiquement aux changements.

Liste des publications personnelles

Revues scientifiques

1. A. Kamoun, S. Tazi, and K. Drira. Fadyrcos, a semantic interoperability framework for collaborative model-based dynamic reconfiguration of networked services. *Computers in Industry*, 63(8) :756-765, 2012. Special Issue on Sustainable Interoperability : The Future of Internet Based Industrial Enterprises.
2. A. El Mougy, A. Kamoun, M. Ibnkahla, S. Tazi, and K. Drira. A context and application-aware framework for resource management in dynamic collaborative wireless M2M networks. *Journal of Network and Computer Applications*, 44(0) :30-45, 2014.
3. D.S. Allison, A. Kamoun, S. Capretz, M.A.M. Tazi, K. Drira, and H. ElYamany. An ontology driven privacy framework for collaborative working environments. *International Journal of Autonomous and Adaptive Communications Systems*, 2014.

Conférences internationales

1. A. Kamoun, S. Tazi, and K. Drira. A semantic adaptive framework for collaborative systems. In *European Concurrent Engineering Conference (ECEC)*, Bucharest (Romania), April 2012. (Primé meilleur article)
2. A. Kamoun and S. Tazi. A semantic role-based access control for intra and inter-organization collaboration. In *Proceedings of the 2012 IEEE 23rd International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises*, WETICE '14, 2014. IEEE Computer Society.
3. S. Ramanathan, A. Kamoun, and C. Chassot. Ontology-based collaborative framework for disaster recovery scenarios. In *Proceedings of the 2012 IEEE 21st International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises*, WETICE '12, pages 104-106, Washington, DC, USA, 2012. IEEE Computer Society.
4. A. Kamoun. Virtuoso : yPBL cookbook. In *International Collaborative Tutorial of Multimedia Ontology-driven Learning Collaborative Approaches (MOLCA)*, Toulouse (France), June 2012, 36 pages.
5. A. Kamoun. Access control in liferay. In *International Collaborative Tutorial of Multimedia Ontology-driven Learning Collaborative Approaches (MOLCA)*, Toulouse (France), April 2014, 53 pages.

Rapports de projet

1. A. Kamoun. Mechanisms for Model-driven Team communication. Galaxy deliverable D2.3, 36 pages, 2012.
2. A. Kamoun. Communication Framework -Use Case Definition. Galaxy deliverable D5.1.5, 30 pages, 2013.
3. A. Kamoun. Framework-Tests Reports. Galaxy deliverable D5.3, 12 pages, 2013.

Bibliographie

[AC07] V. Atluri and S.A. Chun. A geotemporal role&#45 ;based authorisation system. *Int. J. Inf. Comput. Secur.*, 1(1/2) :143–168, January 2007. (Cité page 33.)

[ADB+99] G.D. Abowd, A.K. Dey, P.J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, HUC '99, pages 304–307, London, UK, 1999. Springer-Verlag. (Cité pages 10 et 11.)

[AMMBX09] J. Arnedo-Moreno, K. Matsuo, L. Barolli, and F. Xhafa. A security framework for jxta-overlay. In *International Conference on Network-Based Information Systems, 2009. NBIS '09.*, pages 212–219, Aug 2009. (Cité page 34.)

[And05] A. Anderson. Core and Hierarchical Role Based Access Control (RBAC) Profile of XACML v2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf, 2005. [Online ; accessed 19-March-2014]. (Cité page 27.)

[BB03] Vijay Bharadwaj and J. Baras. Dynamic adaptation of access control policies. In *Military Communications Conference, 2003. MILCOM '03. 2003 IEEE*, volume 2, pages 761–766 Vol.2, Oct 2003. (Cité page 12.)

[BBC97] P.J. Brown, J.D. Bovey, and X. Chen. Context-aware applications : from the laboratory to the marketplace. *Personal Communications, IEEE*, 4(5) :58–64, Oct 1997. (Cité pages 10 et 11.)

[BCDP05] E. Bertino, B. Catania, M.L. Damiani, and P. Perlasca. Geo-rbac : A spatially aware rbac. In *Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies*, SACMAT '05, pages 29–37, New York, NY, USA, 2005. ACM. (Cité page 33.)

[BCM+03] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook : Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003. (Cité page 47.)

[BDCdVS02] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Trans. Inf. Syst. Secur.*, 5(1) :1–35, February 2002. (Cité page 36.)

[BDEKK08] A. Baina, Y. Deswarte, A.A. El Kalam, and M. Kaaniche. Access control for cooperative systems : A comparative analysis. In *Third International Conference on Risks and Security of Internet and Systems, 2008. CRiSIS '08.*, pages 19–26, Oct 2008. (Cité pages 14 et 16.)

[BGBJ05] R. Bhatti, A. Ghafoor, E. Bertino, and J.B.D. Joshi. X-gtrbac : An xml-based policy specification framework and architecture for enterprise-wide access control. *ACM Trans. Inf. Syst. Secur.*, 8(2) :187–227, May 2005. (Cité page 19.)

[BL04] R. Brachman and H. Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. (Cité pages 28 et 29.)

[BRSV+09] I. Bouassida Rodriguez, G. Sancho, T. Villemur, S. Tazi, and K. Drira. A model-driven adaptive approach for collaborative ubiquitous systems. In *Proceedings of the 3rd Workshop on Agent-oriented Software Engineering Challenges for Ubiquitous and Pervasive Computing*, AUPC 09, pages 15–20, New York, NY, USA, 2009. ACM. (Cité page 73.)

[Bul99] A. Bullock. *SPACE : SPatial Access Control for Collaborative Virtual Environments*. PhD thesis, University of Nottingham, 1999. (Cité page 18.)

[BX11] L. Barolli and F. Xhafa. Jxta-overlay : A p2p platform for distributed, collaborative, and ubiquitous computing. *IEEE Transactions on Industrial Electronics,*, 58(6) :2163–2172, June 2011. (Cité page 34.)

[CCT07] L. Cirio, I. Cruz, and R. Tamassia. A role and attribute based access control system using semantic web technologies. In *Proceedings of the 2007 OTM Confederated International Conference on On the Move to Meaningful Internet Systems - Volume Part II*, OTM'07, pages 1256–1266, Berlin, Heidelberg, 2007. Springer-Verlag. (Cité page 30.)

[CE03] M. Coetzee and J.H.P. Eloff. Virtual enterprise access control requirements. In *Proceedings of the 2003 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement Through Technology*, SAICSIT '03, pages 285–294, Republic of South Africa, 2003. South African Institute for Computer Scientists and Information Technologists. (Cité page 36.)

[Cha07] T. Chaari. *Adaptation d'applications pervasives dans des environnements multi-contextes*. PhD thesis, Institut National des Sciences Appliquées de LYON, Sep 2007. (Cité page 10.)

[Cho56] N. Chomsky. Three models for the description of language. *Information Theory, IRE Transactions on*, 2(3) :113–124, September 1956. (Cité page 62.)

[CJ05] S. Chandran and J.B.D. Joshi. Lot-rbac : A location and time-based rbac model. In AnneH.H. Ngu, Masaru Kitsuregawa, ErichJ. Neuhold, Jen-Yao Chung, and QuanZ. Sheng, editors,

Web Information Systems Engineering, volume 3806 of *Lecture Notes in Computer Science*, pages 361–375. Springer Berlin Heidelberg, 2005. (Cité page 33.)

[CKK+04] B. Chang, D. Kim, H. Kim, J. Na, and T. Chung. Active security management based on secure zone cooperation. *Future Gener. Comput. Syst.*, 20(2) :283–293, February 2004. (Cité page 28.)

[CLS+01] M.J. Covington, W. Long, S. Srinivasan, A.K. Dev, M. Ahamad, and G.D. Abowd. Securing context-aware applications using environment roles. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, SACMAT '01, pages 10–20, New York, NY, USA, 2001. ACM. (Cité page 21.)

[CMT04] A. Corradi, R. Montanari, and D. Tibaldi. Context-based access control for ubiquitous service provisioning. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, pages 444–451 vol.1, Sept 2004. (Cité page 31.)

[Cra05] J. Crampton. Understanding and developing role-based administrative models. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, pages 158–167, New York, NY, USA, 2005. ACM. (Cité page 14.)

[CS99] Peter H. Carstensen and Kjeld Schmidt. Computer supported cooperative work : New challenges to systems design. In *In K. Itoh (Ed.), Handbook of Human Factors*, pages 619–636, 1999. (Cité page 8.)

[CS07] W. Claycomb and Dongwan Shin. Enabling delegation for impromptu collaboration in pervasive computing. In *Computer and information sciences, 2007. iscis 2007. 22nd international symposium on*, pages 1–6, Nov 2007. (Cité page 34.)

[DBR10] K. Drira and I. Bouassida Rodriguez. A demonstration of an efficient tool for graph matching and transformation. In *10ème Conférence Internationale Francophone sur l'Extraction et la Gestion des Connaissances (EGC 2010)*, pages 71–73, Hammamet (Tunisie), 2010. (Cité page 65.)

[DCME01] N. Davies, K. Cheverst, K. Mitchell, and A. Efrat. Using and determining location in a context-sensitive tour guide. *Computer*, 34(8) :35–41, Aug 2001. (Cité page 11.)

[Dey98] Anind K. Dey. Context-aware computing : The cyberdesk project. In *AAAI 1998 Spring Symposium on Intelligent Environments*, pages 51–54, Palo Alto, 1998. AAAI Press. (Cité page 10.)

[DGLA99] H.-P. Dommel and J. Garcia-Luna-Aceves. Group coordination support for synchronous internet collaboration. *IEEE Internet Computing*, 3(2) :74–80, March 1999. (Cité pages 1 et 8.)

[Dino3] G. Dini. A secure and available electronic voting service for a large-scale distributed system. *Future Generation Computer Systems*, 19(1) :69 – 85, 2003. Selected papers of the 29th {SPEEDUP} workshop on distributed computing and high-speed networks, 22-23 March 2001, Bern, Switzerland. (Cité page 28.)

[DJYM05] W. Di, L. Jian, D. Yabo, and Z. Miaoliang. Using semantic web technologies to specify constraints of rbac. In *Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies*, PDCAT '05, pages 543–545, Washington, DC, USA, 2005. IEEE Computer Society. (Cité page 30.)

[DLJ08] A. Dersingh, R. Liscano, and A. Jost. Utilizing semantic knowledge for access control in pervasive and ubiquitous systems. In *Networking and Communications, 2008. WIMOB '08. IEEE International Conference on Wireless and Mobile Computing,*, pages 435–441, Oct 2008. (Cité pages 32 et 45.)

[Edw96] W. Keith Edwards. Policies and roles in collaborative applications. In *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work*, CSCW '96, pages 11–20, New York, NY, USA, 1996. ACM. (Cité page 14.)

[EMKI+14] A. El Mougy, A. Kamoun, M. Ibnkahla, S. Tazi, and K. Drira. A context and application-aware framework for resource management in dynamic collaborative wireless {M2M} networks. *Journal of Network and Computer Applications*, 44(0) :30 – 45, 2014. (Cité page 73.)

[FB97] D. Ferraiolo and J. Barkley. Specifying and managing role-based access control within a corporate intranet. In *Proceedings of the Second ACM Workshop on Role-based Access Control*, RBAC '97, pages 77–82, New York, NY, USA, 1997. ACM. (Cité page 14.)

[FB09] R. Ferrini and E. Bertino. Supporting rbac with xacml+owl. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, SACMAT '09, pages 145–154, New York, NY, USA, 2009. ACM. (Cité page 30.)

[FCAG03] D.F. Ferraiolo, R. Chandramouli, G. Ahn, and Serban I. Gavrila. The role control center : Features and case studies. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, SACMAT '03, pages 12–20, New York, NY, USA, 2003. ACM. (Cité page 19.)

[FCK95] D. F. Ferraiolo, J. A. Cugini, and D. Richard Kuhn. Role-Based Access Control (RBAC) : Features and Motivations. In *11th Annual Computer Security Applications Proceedings*, 1995. (Cité page 19.)

[FJK+08] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. Winsborough, and B. Thuraisingham. Rowlbac : Representing role based access control in owl. In *Proceedings of the 13th ACM*

Symposium on Access Control Models and Technologies, SACMAT '08, pages 73–82, New York, NY, USA, 2008. ACM. (Cité page 30.)

[FK92] D. Ferraiolo and R. Kuhn. Role-based access control. In *In 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992. (Cité page 19.)

[FM09] J.B. Filho and H. Martin. A generalized context-based access control model for pervasive environments. In *Proceedings of the 2Nd SIGSPATIAL ACM GIS 2009 International Workshop on Security and Privacy in GIS and LBS*, SPRINGL '09, pages 12–21, New York, NY, USA, 2009. ACM. (Cité page 31.)

[For90] Charles L. Forgy. Expert systems. chapter Rete : A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, pages 324–341. IEEE Computer Society Press, Los Alamitos, CA, USA, 1990. (Cité page 101.)

[FPP+02] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti. drbac : distributed role-based access control for dynamic coalition environments. In *22nd International Conference on Distributed Computing Systems, 2002. Proceedings.*, pages 411–420, 2002. (Cité page 34.)

[FSG+01] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3) :224–274, August 2001. (Cité page 19.)

[GCH+04] D. Garlan, S-W. Cheng, A-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow : architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10) :46–54, Oct 2004. (Cité page 12.)

[GGS07] C. Groba, S. Grob, and T. Springer. Context-dependent access control for contextual information. In *The Second International Conference on Availability, Reliability and Security, 2007. ARES 2007.*, pages 155–161, April 2007. (Cité page 31.)

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns : Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. (Cité page 67.)

[GMPT01] C.K. Georgiadis, I. Mavridis, G. Pangalos, and R.K. Thomas. Flexible team-based access control using contexts. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, SACMAT '01, pages 21–27, New York, NY, USA, 2001. ACM. (Cité page 21.)

[GP99] A. Gomez-Perez. Ontological engineering : A state of the art. *Expert Update*, 2(3) :33 – 43, 1999. (Cité pages 29 et 44.)

[Gru93] T.R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2) :199–220, June 1993. (Cité page 29.)

[Gru94] Jonathan Grudin. Computer-supported cooperative work : History and focus. *Computer*, 27(5) :19–26, May 1994. (Cité page 8.)

[Gua97] N. Guarino. Some organizing principles for a unified top-level ontology. In *AAAI 1997 SPRING SYMPOSIUM ON ONTOLOGICAL ENGINEERING (LADSEB-CNR INT. REP. 02/97, V3.0*. AAAI Press, 1997. (Cité page 29.)

[Gue06] K. Guennoun. *Architectures dynamiques dans le contexte des applications Ã base de composants et orientÃ©es services*. PhD thesis, Université Paul Sabatier, Toulouse, 2006. (Cité page 63.)

[HBRDPH13] M.A. Hannachi, I. Bouassida Rodriguez, K. Drira, and S. Pomares Hernandez. Gmte : A tool for graph transformation and exact/inexact graph matching. In WalterG. Kropatsch, NicoleM. Artner, Yll Haxhimusa, and Xiaoyi Jiang, editors, *Graph-Based Representations in Pattern Recognition*, volume 7877 of *Lecture Notes in Computer Science*, pages 71–80. Springer Berlin Heidelberg, 2013. (Cité page 65.)

[HCZ+06] Nuermaimaiti Heilili, Yang Chen, Chen Zhao, Zhenxing Luo, and Zuoquan Lin. An owl-based approach for rbac with negative authorization. In *Proceedings of the First International Conference on Knowledge Science, Engineering and Management*, KSEM'06, pages 164–175, Berlin, Heidelberg, 2006. Springer-Verlag. (Cité page 30.)

[HI04] K. Henricksen and J. Indulska. A software engineering framework for context-aware pervasive computing. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications, 2004. PerCom 2004.*, pages 77–86, March 2004. (Cité page 11.)

[HL09] K. Hamadache and L. Lancieri. Role-based collaboration extended to pervasive computing. In *International Conference on Intelligent Networking and Collaborative Systems, 2009. INCOS '09.*, pages 9–15, Nov 2009. (Cité page 8.)

[Hor51] A. Horn. On sentences which are true of direct unions of algebras. *J. Symb. Log.*, 16(1) :14–21, 1951. (Cité page 48.)

[Hor01] P. Horn. Autonomic computing : Ibm's perspective on the state of information technology. Technical report, 2001. (Cité page 12.)

[HPSB+04] I Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. Swrl : A semantic web rule language combining owl and ruleml. W3c member submission, World Wide Web Consortium, 2004. (Cité pages 30 et 48.)

[HWCL06] X. Huang, H. Wang, Y. Chen, and J. Lin. A context, rule and role-based access control model in enterprise pervasive computing environment. In *1st International Symposium on Pervasive Computing and Applications, 2006.*, pages 497–502, Aug 2006. (Cité page 33.)

[JBLG05] J.B.D Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role-based access control model. *IEEE Transactions on Knowledge and Data Engineering,*, 17(1) :4–23, Jan 2005. (Cité page 33.)

[JHJ06] S Javanmardi, H Hemmati, and R Jalili. An access control framework for pervasive computing environments. In Hamid R. Arabnia, editor, *PSC*, pages 97–103. CSREA Press, 2006. (Cité page 32.)

[JKS12] X. Jin, R. Krishnan, and R. Sandhu. A unified attribute-based access control model covering dac, mac and rbac. In Nora Cuppens-Boulahia, FrÃ©dÃ©ric Cuppens, and Joaquin Garcia-Alfaro, editors, *Data and Applications Security and Privacy XXVI*, volume 7371 of *Lecture Notes in Computer Science*, pages 41–55. Springer Berlin Heidelberg, 2012. (Cité page 23.)

[JP96] T. Jaeger and A. Prakash. Requirements of role-based access control for collaborative systems. In *Proceedings of the First ACM Workshop on Role-based Access Control*, RBAC '95, New York, NY, USA, 1996. ACM. (Cité pages 14 et 16.)

[KAT12] D. Kulkarni, T. Ahmed, and A. Tripathi. A generative programming framework for context-aware cscw applications. *ACM Trans. Softw. Eng. Methodol.*, 21(2) :11 :1–11 :35, March 2012. (Cité pages 10 et 11.)

[KB10] M.S. Kirkpatrick and E. Bertino. Enforcing spatial constraints for mobile rbac systems. In *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies*, SACMAT '10, pages 99–108, New York, NY, USA, 2010. ACM. (Cité page 33.)

[KC03] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1) :41–50, Jan 2003. (Cité pages 11, 13 et 56.)

[KHD08] M. Knechtel, J. Hladik, and F. Dau. Using OWL DL Reasoning to decide about authorization in RBAC. In *OWLED'08 : Proceedings of the OWLED 2008 Workshop on OWL : Experiences and Directions*, 2008. (Cité page 30.)

[KS07] Angelos D. Keromytis and Jonathan M. Smith. Requirements for scalable access control and security management architectures. *ACM Trans. Internet Technol.*, 7(2), May 2007. (Cité pages 2 et 14.)

[KT08] D. Kulkarni and A. Tripathi. Context-aware role-based access control in pervasive computing systems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, SACMAT '08, pages 113–122, New York, NY, USA, 2008. ACM. (Cité page 33.)

[KT14] A. Kamoun and S. Tazi. A semantic role-based access control for intra and inter-organization collaboration. In *Proceedings*

of the 2012 IEEE 23rd International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises, WETICE '14. IEEE Computer Society, 2014. (Cité page 33.)

[KTD12a] A. Kamoun, S. Tazi, and K. Drira. Fadyrcos, a semantic interoperability framework for collaborative model-based dynamic reconfiguration of networked services. *Computers in Industry*, 63(8) :756 – 765, 2012. Special Issue on Sustainable Interoperability : The Future of Internet Based Industrial Enterprises. (Cité page 73.)

[KTD12b] A. Kamoun, S. Tazi, and K. Drira. A semantic adaptive framework for collaborative systems. In *European Concurrent Engineering Conference (ECEC)*, Bucharest (Romania), April 2012. (Cité page 73.)

[Lad00] R. Laddaga. Active software. In *Proceedings of the First International Workshop on Self-adaptive Software*, IWSAS' 2000, pages 11–26, Secaucus, NJ, USA, 2000. Springer-Verlag New York, Inc. (Cité page 11.)

[Lam74] Butler W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8(1) :18–24, January 1974. (Cité page 17.)

[LS07] T.H. Lim and S.U. Shin. Intelligent access control mechanism for ubiquitous applications. In *6th IEEE/ACIS International Conference on Computer and Information Science, 2007. ICIS 2007.*, pages 955–960, July 2007. (Cité page 33.)

[LW05] R. Liscano and K. Wang. A sip-based architecture model for contextual coalition access control for ubiquitous computing. In *The Second Annual International Conference on Mobile and Ubiquitous Systems : Networking and Services, 2005. MobiQuitous 2005.*, pages 384–392, July 2005. (Cité page 34.)

[LW07] R. Liscano and K. Wang. A context-based delegation access control model for pervasive computing. In *21st International Conference on Advanced Information Networking and Applications Workshops, 2007, AINAW '07.*, volume 2, pages 44–51, May 2007. (Cité page 34.)

[LWG+06] S. Lachmund, T. Walter, L. Gomez, L. Bussard, and E. Olk. Context-aware access control ; making access control decisions based on context information. In *Third Annual International Conference on Mobile and Ubiquitous Systems : Networking Services, 2006.*, pages 1–8, July 2006. (Cité page 31.)

[MC02] R. Meier and V. Cahill. Taxonomy of distributed event-based programming systems. In *22nd International Conference on Distributed Computing Systems Workshops, 2002. Proceedings.*, pages 585–586, 2002. (Cité page 75.)

[MGvdHW10] I. MistrÃk, J. Grundy, A. van der Hoek, and J. Whitehead. Collaborative software engineering : Challenges and prospects. In Ivan MistrÃk, John Grundy, AndrÃ© Hoek, and Jim Whitehead, editors, *Collaborative Software Engineering*, pages 389–403. Springer Berlin Heidelberg, 2010. (Cité page 10.)

[MKSK00] R. Mizoguchi, K. Kozaki, T. Sano, and Y. Kitamura. Construction and deployment of a plant ontology. In Rose Dieng and Olivier Corby, editors, *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*, volume 1937 of *Lecture Notes in Computer Science*, pages 113–128. Springer Berlin Heidelberg, 2000. (Cité page 29.)

[Mos05] T. Moses. eXtensible Access Control Markup Language TC v2.0 (XACML), February 2005. (Cité pages 21 et 23.)

[MSKC04] P.K. McKinley, S.M. Sadjadi, E.P. Kasten, and B. H C Cheng. Composing adaptive software. *Computer*, 37(7) :56–64, July 2004. (Cité pages 11 et 12.)

[MTPP13] P.N. Mahalle, P.A. Thakre, N.R. Prasad, and R. Prasad. A fuzzy approach to trust based access control in internet of things. In *3rd International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace Electronic Systems (VITAE), 2013.*, pages 1–5, June 2013. (Cité page 34.)

[Oas05] Oasis extensible access control markup language (xacml) version 2.0, 2005. (Cité pages vi, 24 et 26.)

[OGT+99] P. Oreizy, M.M. Gorlick, R.N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D.S. Rosenblum, and A.L. Wolf. An architecture-based approach to self-adaptive software. *Intelligent Systems and their Applications, IEEE*, 14(3) :54–62, May 1999. (Cité page 11.)

[Pas98] J. Pascoe. Adding generic contextual capabilities to wearable computers. In *Second International Symposium on Wearable Computers, 1998. Digest of Papers.*, pages 92–99, Oct 1998. (Cité page 10.)

[PDB06] W. Prinz, S. Dustdar, and Isidro L. Ballesteros. New Collaborative Working Environments 2020, 2006. (Cité page 9.)

[PFMP04] T. Priebe, E.B. Fernandez, J.I. Mehlau, and G. Pernul. A pattern system for access control. In *IN RESEARCH DIRECTIONS IN DATA AND APPLICATIONS SECURITY XVIII, C. FARKAS AND P. SAMARATI (EDS.), PROCS OF THE 18TH. ANNUAL IFIP WG 11.3 WORKING CONFERENCE ON DATA AND APPLICATIONS SECURITY*, pages 25–28. Kluwer, 2004. (Cité page 21.)

[QKF+06] K. Quinn, A. Kenny, K. Feeney, D. Lewis, D. O'Sullivan, and V.P. Wade. A framework for the decentralisation and management of collaborative applications in ubiquitous computing environments. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 1–4, April 2006. (Cité page 34.)

[RDCJ09] I.B. Rodriguez, K. Drira, C. Chassot, and M. Jmaiel. A model-based multi-level architectural reconfiguration applied to adaptability management in context-aware cooperative communication support systems. In *Software Architecture, 2009*

European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on, pages 353–356, Sept 2009. (Cité page 65.)

[RDF+12] S. Raje, C. Davuluri, M. Freitas, R. Ramnath, and J. Ramanathan. Using ontology-based methods for implementing role-based access control in cooperative systems. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 763–764, New York, NY, USA, 2012. ACM. (Cité page 30.)

[RGD+08] I.B. Rodriguez, K. Guennoun, K. Drira, C. Chassot, and M. Jmaiel. Implementing a rule-driven approach for architectural self configuration in collaborative activities using a graph rewriting formalism. In *Proceedings of the 5th International Conference on Soft Computing As Transdisciplinary Science and Technology*, CSTST '08, pages 484–491, New York, NY, USA, 2008. ACM. (Cité page 65.)

[RKY06] Indrakshi Ray, Mahendra Kumar, and Lijun Yu. Lrbac : A location-aware role-based access control model. In Aditya Bagchi and Vijayalakshmi Atluri, editors, *Information Systems Security*, volume 4332 of *Lecture Notes in Computer Science*, pages 147–161. Springer Berlin Heidelberg, 2006. (Cité page 33.)

[RL05] P. Robertson and R. Laddaga. Model based diagnosis and contexts in self adaptive software. In Ozalp Babaoglu, MÃ¡rk Jelasity, Alberto Montresor, Christof Fetzer, Stefano Leonardi, Aad van Moorsel, and Maarten van Steen, editors, *Self-star Properties in Complex Information Systems*, volume 3460 of *Lecture Notes in Computer Science*, pages 112–127. Springer Berlin Heidelberg, 2005. (Cité page 13.)

[RM10] S. Rashwand and J. Mišić. A novel access control framework for secure pervasive computing. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, IWCMC '10, pages 829–833, New York, NY, USA, 2010. ACM. (Cité page 32.)

[RPM98] N. S. Ryan, J. Pascoe, and D. R. Morse. Enhanced reality fieldwork : the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications in Archaeology 1997*, British Archaeological Reports, pages 182–196, Oxford, October 1998. Tempus Reparatum. (Cité pages 10 et 11.)

[RR00] Pierre N. Robillard and Martin P. Robillard. Types of collaborative work in software engineering. *Journal of Systems and Software*, 53(3) :219 – 224, 2000. (Cité page 10.)

[Sat01] M. Satyanarayanan. Pervasive computing : vision and challenges. *Personal Communications, IEEE*, 8(4) :10–17, Aug 2001. (Cité page 7.)

[SCFY96] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-based access control models. *Computer*, 29(2) :38–47, Feb 1996. (Cité page 19.)

[Sch90] Michael Schrage. *Shared Minds : The New Technologies of Collaboration*. Random House Inc., New York, NY, USA, 1990. (Cité page 8.)

[SFK00] R. Sandhu, D. Ferraiolo, and R. Kuhn. The nist model for role-based access control : Towards a unified standard. In *Proceedings of the Fifth ACM Workshop on Role-based Access Control*, RBAC '00, pages 47–63, New York, NY, USA, 2000. ACM. (Cité page 19.)

[SGB08] A. Samuel, A. Ghafoor, and E. Bertino. Context-aware adaptation of access-control policies. *Internet Computing, IEEE*, 12(1) :51–54, Jan 2008. (Cité page 12.)

[SN04] M. Strembeck and G. Neumann. An integrated approach to engineer and enforce context constraints in rbac environments. *ACM Trans. Inf. Syst. Secur.*, 7(3) :392–427, August 2004. (Cité page 32.)

[SS96] R. Sandhu and P. Samarati. Authentication, access control, and audit. *ACM Comput. Surv.*, 28(1) :241–243, March 1996. (Cité page 14.)

[ST94] B.N. Schilit and M.M. Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5) :22–32, Sept 1994. (Cité pages 10 et 11.)

[ST09] M. Salehie and L. Tahvildari. Self-adaptive software : Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2) :14 :1–14 :42, May 2009. (Cité pages 11 et 12.)

[STV10] G. Sancho, S. Tazi, and T. Villemur. Gco : a generic collaboration ontology. In *Proceedings of the Fourth International Conference on Advances in Semantic Processing (SEMPARO 2010)*, Florence, Italy, 2010. (Cité pages 55 et 75.)

[SWM04] M.K. Smith, C. Welty, and D.L McGuiness. Owl web ontology language guide. w3c recommendation, 2004. [Online ; accessed 27-March-2014]. (Cité pages 29 et 45.)

[TAK+04a] A. Tripathi, T. Ahmed, D. Kulkarni, R. Kumar, and K. Kashiramka. Context-based secure resource access in pervasive computing environments. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004.*, pages 159–163, March 2004. (Cité page 22.)

[TAK+04b] A. Tripathi, T. Ahmed, D. Kulkarni, R. Kumar, and K. Kashiramka. Context-based secure resource access in pervasive computing environments. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004.*, pages 159–163, March 2004. (Cité page 32.)

[TAPH05] W. Tolone, G-J. Ahn, T. Pai, and S. Hong. Access control in collaborative systems. *ACM Comput. Surv.*, 37(1) :29–41, March 2005. (Cité pages 14, 16 et 22.)

[The03] O. Thereaux. Common http implementation problems. W3C note, World Wide Web Consortium. http://www.w3.org/TR/chips/, 2003. [Online ; accessed 10-June-2013]. (Cité page 49.)

[Theo7] The OSGi Alliance. OSGi service platform core specification, release 4.1. http://www.osgi.org/Specifications, 2007. (Cité page 73.)

[Tho97] R.K. Thomas. Team-based access control (tmac) : A primitive for applying role-based access controls in collaborative environments. In *Proceedings of the Second ACM Workshop on Role-based Access Control*, RBAC '97, pages 13–19, New York, NY, USA, 1997. ACM. (Cité page 21.)

[TMKL06] A. Toninelli, R. Montanari, L. Kagal, and O. Lassila. A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In *Proceedings of the 5th International Conference on The Semantic Web*, ISWC'06, pages 473–486, Berlin, Heidelberg, 2006. Springer-Verlag. (Cité pages 12, 30 et 32.)

[TS98] Roshan K. Thomas and Ravi S. Sandhu. Task-based authorization controls (tbac) : A family of models for active and enterprise-oriented autorization management. In *Proceedings of the IFIP TC11 WG11.3 Eleventh International Conference on Database Securty XI : Status and Prospects*, pages 166–181, London, UK, UK, 1998. Chapman & Hall, Ltd. (Cité page 18.)

[TYW04] R. Tamassia, D. Yao, and W.H. Winsborough. Role-based cascaded delegation. In *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies*, SACMAT '04, pages 146–155, New York, NY, USA, 2004. ACM. (Cité page 34.)

[Vil06] T. Villemur. *Modèles et services logiciels pour le travail collaboratif*. Hdr, Université Paul Sabatier - Toulouse III, September 2006. 06716 06716. (Cité page 9.)

[Wei91] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3) :66–75, September 1991. (Cité pages 1 et 7.)

[Win01] T. Winograd. Architectures for context. *Hum.-Comput. Interact.*, 16(2) :401–419, December 2001. (Cité page 10.)

[WL01] H.F. Wedde and M. Lischka. Modular authorization. In *Proceedings of the sixth ACM Symposium on Access Control Models and Technologies (SACMAT*, pages 3–4. ACM, ACM Press, 2001. (Cité page 36.)

[WL04] H.F. Wedde and M. Lischka. Role-based access control in ambient and remote space. In *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies*, SACMAT '04, pages 21–30, New York, NY, USA, 2004. ACM. (Cité page 33.)

[WZC08] H. Wang, Y. Zhang, and J. Cao. Access control management for ubiquitous computing. *Future Generation Computer Systems*, 24(8) :870 – 878, 2008. (Cité pages 28 et 31.)

[WZCV03] H. Wang, Y. Zhang, J. Cao, and V. Varadharajan. Achieving secure and flexible m-services through tickets. *Systems, Man and Cybernetics, Part A : Systems and Humans, IEEE Transactions on*, 33(6) :697–708, Nov 2003. (Cité page 28.)

[YT05] E. Yuan and J. Tong. Attributed based access control (abac) for web services. In *IEEE International Conference on Web Services, 2005. ICWS 2005. Proceedings. 2005.*, pages –569, July 2005. (Cité page 21.)

[YTP05] D. Yao, R. Tamassia, and S. Proctor. On improving the performance of role-based cascaded delegation in ubiquitous computing. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005.*, pages 157–170, Sept 2005. (Cité page 34.)

[ZMR05] W. Zhou, C. Meinel, and V.H. Raja. A framework for supporting distributed access control policies. In *10th IEEE Symposium on Computers and Communications, 2005. ISCC 2005. Proceedings.*, pages 442–447, June 2005. (Cité page 33.)

Ce document a été préparé à l'aide de l'éditeur de texte GNU Emacs et du logiciel de composition typographique LaTeX 2ε.

Titre Adaptation d'architectures logicielles de contrôle d'accès dans les environnements collaboratifs ubiquitaires

Résumé L'objectif de ce travail de thèse est d'explorer la conception de mécanismes de contrôle d'accès dans les environnements collaboratifs ubiquitaires. Les changements dynamiques et l'augmentation du nombre de ressources et d'utilisateurs dans ce type d'environnements rendent le contrôle d'accès plus complexe. Dans ce travail, nous proposons une approche d'adaptation qui permet de déployer et de reconfigurer dynamiquement les composants logiciels en cours d'exécution en fonction des fluctuations de la structure de collaboration et des changements des besoins de contrôle d'accès. En premier lieu, un modèle sémantique de contrôle d'accès est proposé afin de permettre l'expression des besoins de contrôle d'accès selon les situations de collaboration. Des règles d'adaptation qui assurent la décentralisation des mécanismes de contrôle d'accès ont été proposées dans le but d'assurer un passage à l'échelle. En second lieu, nous proposons un framework qui permet d'effectuer l'adaptation et d'exécuter les actions de déploiement des composants logiciels en fonction des besoins de contrôle d'accès à tout moment de la collaboration. Nous avons validé nos contributions à travers un scénario mettant en œuvre des services de collaboration et de partage de ressources dans un environnement de travail collaboratif.

Mots-clés Adaptation, modèles sémantiques, contrôle d'accès, collaboration, environnements ubiquitaires

Title Adaptation of access control software architectures in collaborative ubiquitous environments

Abstract The objective of this work is to explore the design of access control mechanisms in collaborative ubiquitous environments. The dynamic changes and the increasing number of resources and users in this type of environments make the access control more complex. In this work, we propose an adaptation approach that allows deploying and reconfiguring the software components at runtime according to the changes of the collaboration structure and access control requirements. Firstly, a semantic model of access control is proposed to allow the expression of access control requirements. Adaptation rules that ensure the decentralization of access control mechanisms are also proposed for scalability reasons. Secondly, we propose a framework that makes adaptation and implements deployment actions during the collaboration process. We validate our contributions through a scenario that implements collaboration services and resources sharing in a collaborative working environment.

Keywords Adaptation, semantic models, access control, collaboration, ubiquitous environments