Université Abdelhamid Ibn Badis Mostaganem
Faculté des Sciences Exactes et Informatique
Département d'Informatique

FORMALISATION DES PROPRIETES DE SECURITE DES OBSERVATEURS POUR LA VALIDATION DES TRANSFORMATIONS

Thèse de Doctorat
de l'Université de Mostaganem

Spécialité : Informatique
Option : Apprentissage automatique et Web intelligence

Présentée par

Mohammed Walid KRAKALLAH

Soutenue le ../../2022 devant un jury composé de :

Président du jury	**Pr. SEHABA Karim**	(Université de Mostaganem)
Examinateurs	**Dr. ROUBA Baroudi**	(Université de Mostaganem)
	Dr. MEROUFEL Bakhta	(Université de Mostaganem)
	Pr. CHOURAQUI Samira	(USTO Mohamed-Boudiaf)
Directeur de thèse	**Pr. NAIT-BAHLOUL Safia**	(Université d'Oran1)

RESUME

La modélisation des politiques de contrôle d'accès et les contraintes de sécurité ont comme objectif de décrire les aspects des différentes exigences de sécurité à un niveau d'abstraction plus élevé. Elle offre un cadre logique qui permet une meilleure caractérisation des propriétés qu'elles doivent satisfaire et donne un langage d'expression commun sans ambiguïtés. Pour réunir ces conditions, plusieurs méthodes ont été proposées, afin d'enrichir les formalismes des outils de modélisation existants et faciliter également la description et l'analyse de toutes les contraintes du contrôle d'accès et les contraintes spatio-temporelles. Une fois que la politique de contrôle d'accès est modélisée, la difficulté se situe dans l'expression et la vérification formelle des propriétés de cette politique. La formalisation permet également d'utiliser des outils mathématiques pour raisonner sur les modèles et vérifier les propriétés souhaitées. Cette vérification doit être effectuée quelle que soit la politique déterminée. Dans cette thèse, nous proposons d'élaborer un cadre formel et pratique pour la spécification et la validation de politiques de contrôle d'accès hybrides. Dans les situations d'urgence, les utilisateurs doivent parfois accéder à des ressources non autorisées dans des situations normales. Pour augmenter la flexibilité du contrôle d'accès, une extension du modèle UACML au modèle RBAC d'urgence (E-RBAC) est proposée. Nous commençons par la spécification semi-formelle des règles de contrôle d'accès à l'aide d'un méta-modèle que nous nommons E-UACML accompagnées de contraintes de contrôle d'accès et des contraintes spatio-temporelles. Un ensemble d'outils (E-UACML, Fiacre, OBP/CDL) pour exprimer et vérifier les propriétés de contrôles d'accès est introduit. Les modèles d'activité associés à E-UACML sont ensuite traduits en une spécification formelle exprimée dans le langage Fiacre et les exigences en automate observateur à l'aide du langage CDL. Ce dernier est utilisé pour vérifier formellement les propriétés avec l'outil OBP Explorer exploitant une technique de vérification formelle de propriétés de sécurité par Model-checking. Nous évaluons notre approche avec une étude de cas. Ainsi, nous concevons un modèle de contrôle d'accès générique, et de vérifier que les propriétés de sécurité sont respectées dans ce modèle.

Mots-clés. Contrôle d'accès, Automate observateur, UACML, E-UACML, Fiacre, OBP/CDL, Méthodes formelles, Model-checking.

ملخص

تهدف نمذجة سياسات التحكم في الوصول والقيود الأمنية إلى وصف جوانب متطلبات الأمان المختلفة بمستوى أعلى من التجريد. إنها توفر إطارًا منطقيًا يسمح بتوصيف أفضل للخصائص التي يجب أن تفي بها ويعطي لغة تعبير مشتركة بدون غموض. لنمذجة هذه الشروط، تم اقتراح عدة طرق من أجل إثراء الشكليات لأدوات النمذجة الحالية بالإضافة إلى تسهيل وصف وتحليل جميع قيود التحكم في الوصول والقيود الزمنية. بمجرد صياغة سياسة التحكم في الوصول، تكمن الصعوبة في التعبير عن خصائص هذه السياسة والتحقق منها رسميًا. كما أن الصياغة الرسمية تجعل من الممكن استخدام الأدوات الرياضية للتفكير في النماذج والتحقق من الخصائص المرغوبة. يجب إجراء هذا التحقق بغض النظر عن السياسة التي يحددها المسؤولون. في هذا العمل، نقترح تطوير إطار عمل رسمي وعملي لمواصفات والتحقق من سياسات التحكم في الوصول الهجين. في حالات الطوارئ، يحتاج المستخدمون أحيانًا إلى الوصول إلى الموارد غير المسموح بها في المواقف العادية. من أجل زيادة مرونة التحكم في الوصول، تم اقتراح امتداد نموذج UACML لنموذج الطوارئ E-RBAC. نبدأ بالمواصفات شبه الرسمية لقواعد التحكم في الوصول باستخدام النموذج المرجعي نسميه E-UACML مصحوبًا بقيود التحكم في الوصول والقيود المكانية والزمانية. تم تقديم مجموعة من الأدوات (E-UACML، OBP / CDL،Fiacre) للتعبير عن خصائص التحكم في الوصول والتحقق منها. يتم بعد ذلك ترجمة نماذج النشاط المرتبطة بـ E-UACMLإلى مواصفة رسمية معبر عنها بلغة Fiacre والمتطلبات في مراقب آلي باستخدام لغة CDL. يتم استخدام هذا الأخير للتحقق رسميًا من الخصائص باستخدام أداة OBP Explorer التي تستغل تقنية للتحقق الرسمي من خصائص الأمان عن طريق فحص النموذج. نقوم بتقييم نهجنا مع دراسة الحالة. لذلك، نصمم نموذجًا عامًا للتحكم في الوصول، ونتحقق من احترام خصائص الأمان في هذا النموذج.

الكلمات المفتاحية. التحكم في الوصول، المراقب الآلي، OBP / CDL، UACML، E-UACML Fiacre، الطرق الرسمية، فحص النماذج.

ABSTRACT

The modeling of access control policies and security constraints aims to describe aspects of the different security requirements at a higher level of abstraction. it offers a logical framework that allows a better characterization of the properties they must fulfill and gives a common language of expression without ambiguities. To model these conditions, several methods have been proposed in order to enrich the formalisms of existing modeling tools as well as facilitating the description and analysis of all access control constraints and time constraints. Once the access control policy is modeled, the difficulty lies in expressing and formally verifying the properties of this policy. The formalization also makes it possible to use mathematical tools to reason about the models and verify the desired properties. This verification must be carried out regardless of the policy determined by the administrators. In this work, we suggest to develop a formal and practical framework for the specification and validation of hybrid access control policies. In emergency situations, users sometimes need to access resources that are not allowed in normal situations. In order to increase the flexibility of access control, an extension of the UACML model to the Emergency RBAC (E-RBAC) model is proposed. We start with the semi-formal specification of access control rules using a meta-model that we call E-UACML accompanied by access control constraints and spatio-temporal constraints. A set of tools (E-UACML, Fiacre, OBP/CDL) for expressing and verifying access control properties is introduced. The activity models associated with E-UACML are then translated into a formal specification expressed in the Fiacre language and the requirements in an observer automaton using the CDL language. This latter is used in order to formally verify the properties with the OBP Explorer tool exploiting a technique for formal verification of security properties by Model-checking. We evaluate our approach with a case study. Therefore, we design a generic access control model, and verify that the security properties are respected in this model.

Keywords. Access control, Observer Automata, UACML, Fiacre, OBP/CDL, formal methods, model-checking.

Remerciements

Je veux tout d'abord remercier vivement mon directeur de thèse, Mme Safia NAIT-BAHLOUL qui m'a donné l'opportunité de commencer cette thèse et par qui j'ai appris tant de choses et de m'avoir accompagné et encouragé dans ce travail.

J'exprime aussi tous mes sincères remerciements aux membres du jury qui m'ont fait l'honneur de juger ce travail.

J'adresse mes remerciements à Pr. SEHABA Karim (Université de Mostaganem), Dr. ROUBA Baroudi (Université de Mostaganem), Dr. MEROUFEL Bakhta (Université de Mostaganem) et au Pr. CHOURAQUI Samira (Université d'Oran Mohamed-Boudiaf) de m'avoir fait l'honneur d'examiner ce travail de thèse et d'avoir accepté de faire partie du jury. J'apprécie l'intérêt qu'ils ont porté à mes travaux.

Je tiens également à remercier le staff de l'Université de Mostaganem. Je pense notamment au Dr. Omar BELHAMITI pour ses précieux conseils, son soutien, sa disponibilité. Je remercie également Dr. Fouad HENNI pour son soutien et ses orientations, il a toujours fait de son mieux pour faciliter mon travail.

Je remercie l'ensemble de mes professeurs qui m'ont inculqués leur savoir et leur expérience et tous ceux qui de près ou de loin ont contribué à la réalisation de ce travail.

Je remercie également les membres du laboratoire LITIO de l'Université d'Oran 1 que j'ai eu l'honneur de côtoyer pour leur amitié et leurs mots de soutien.

À tous mes précieux amis de l'Université de Mostaganem, en particulier : Amin, Djalil, Mohamed, Amine, Amira, Saliha, Djamila, Maghnia et Houria avec lesquels j'ai partagé des moments inoubliables.

Dédicace

Je dédie ce travail à toutes les personnes qui me sont très chères :

Mes parents pour les quels je suis reconnaissant pour m'avoir constamment soutenu, leurs conseils et leur soutien tout au long de mes études, pour avoir cru en moi pour mener à bien ce travail, pour avoir mis à ma disposition tous les moyens dont ils disposaient, mais surtout pour leur amour. Merci infiniment !

Mes sœurs Amina, Sihem et Nesrine pour leur soutien indéfectible et leurs encouragements.

TABLE DES MATIERES

Table des matières

Liste des Figures

Liste des tableaux

PARTIE I: INTRODUCTION

Introduction générale

Les systèmes de contrôle d'accès sont des mécanismes qui autorisent ou interdisent à un sujet, entité capable d'initier des requêtes comme personne, logiciel, etc… l'accès à des objets, entités à protéger comme fichiers, dossiers, etc. Ces systèmes sont créés sur la base de modèles de contrôle d'accès qui permettent de présenter des politiques de sécurité de façon à atteindre des objectifs particuliers. Plusieurs types de modèles de contrôle d'accès existent dans la littérature. Les modèle de base sont : les modèles MAC (Mandatory Access Control) [1], les modèles DAC (Discretionary Access Control) [2], et le modèle RBAC (Role-Based Access Control) [3].

Cependant, les développeurs en général éprouvent des difficultés à spécifier correctement des exigences de contrôle d'accès adéquates pendant les phases initiales de développement. Pour modéliser ces exigences, les développeurs tendent vers l'Ingénierie Dirigée par les Modèles (IDM) [4] qui sont des systèmes modélisés à l'aide d'une notation semi-formelle et sont par la suite validés puis implantés. Plusieurs méthodes ont été proposés, comme securUML [5], UMLsec [6], UACML [7], etc… afin de faciliter la description et l'analyse de toutes les contraintes du contrôle d'accès.

Parmi ces méthodes nous nous intéressons particulièrement à la méthode semi-formelle UACML (Unified Access Control Modeling Language) [7]. Cette méthode permet grâce à des concepts assez généraux de produire des modèles de contrôle d'accès hybrides basés sur UML.

Les besoins en sécurité sont diversifiés aujourd'hui. En effet, on rencontre de plus en plus de situations dans lesquelles, il ne suffit plus d'un modèle de contrôle d'accès pour répondre aux exigences de sécurité d'un système. Dans ce genre de situation, Il est nécessaire de combiner deux, voire plusieurs concepts (rôles, groupes, niveaux de sécurité, etc.) pour renforcer le contrôle d'accès. Par exemple, avec l'utilisation du Cloud Computing, on peut avoir besoin de combiner les trois modèles de contrôle d'accès de base DAC, MAC et RBAC (contrôle d'accès hybrides).

Dans les situations d'urgence, les utilisateurs doivent parfois accéder à des ressources non autorisées dans des situations normales. Parmi les principales contributions de ce travail, la proposition d'une extension du modèle UACML par le modéle E-RBAC [8] qui permettra de contrôler le système dans les situations d'urgence. Dans le cadre de notre étude, nous proposons le méta-modèle E-UACML (Emergency Unified Access Control Modeling Language) [9] qui étend les diagrammes d'activité UACML pour représenter la politique RBAC d'urgence (E-RABC).

Une fois que la politique de contrôle d'accès est modélisée, la difficulté se situe dans l'expression des propriétés et leur vérification formelle. E-UACML à lui seul ne peut garantir que l'implémentation des modèles créés à partir de notations UML puisse satisfaire les exigences de sécurité d'un système. Il est alors nécessaire d'introduire de nouvelles techniques pour faire en sorte que les implémentations des modèles E-UACML soient correctes par rapport aux propriétés de sécurité. Notons qu'une démarche IDM (Ingénierie Dirigée par les Modèles) reste insuffisante dans le sens où elle n'indique pas comment utiliser les modèles pour appliquer une analyse formelle, pour cela, des techniques formelles de transformation doivent être introduites.

Ce travail de thése introduit une méthodologie (architecture globale) et une chaine de transformation (tool-chain) pour formaliser les propriétés de sécurité. Nous utilisons dans cette architecture notre modèle E-UACML qui est un méta-modèle de contrôle d'accès qui offre au moins deux avantages : (i) simplifier la syntaxe de la modélisation et (ii) fournir aux auteurs des politiques un cadre général pour représenter une variété de modèles pour modéliser plusieurs types combinés de politiques de contrôle d'accès.

Comme E-UACML est un langage semi-formel, une traduction vers une méthode formelle est nécessaire. Nous commençons par la spécification semi-formelle des règles de contrôle d'accès à l'aide des diagrammes d'activité E-UACML accompagnés de contraintes de contrôle d'accès et des contraintes spatio-temporelles. Les modèles d'activité sont ensuite traduits en une spécification formelle exprimée dans le langage Fiacre [10]. Ce dernier est utilisé pour vérifier formellement les propriétés avec l'outil OBP Explorer basé sur le model-checking.

Organisation de la thèse

Dans le premier chapitre, nous introduisons les concepts de base du contrôle d'accès et l'évolution de leur gestion. Nous présentons les concepts de base du contrôle d'accès et soulignons les principaux objectifs de recherche de la thèse.

Après un aperçu général des principaux modèles de contrôle d'accès proposés dans la littérature, nous nous concentrons sur le nouveau modèle de contrôle d'accès basé sur les rôles d'urgence « Emergency role-based access control» (E-RBAC) [8].

Le deuxième chapitre présente des politiques de sécurité dans des cadres bien définis, plusieurs méthodes de modélisation semi-formelles de règles de contrôle d'accès sont proposées. Ces méthodes ont essentiellement pour objectif d'aider les intervenants dans des projets à comprendre les grandes lignes des exigences de ces projets. Elles facilitent également l'implémentation des systèmes. Ainsi les méthodes que nous présentons dans ce chapitre, permettent d'exprimer à l'aide de langages semi-formels comme UML (Unified Modeling Language), des politiques de sécurité de façon à respecter les exigences des modèles de contrôle d'accès.

Dans le troisième chapitre, nous nous intéressons aux techniques de vérification par model-cheking. Nous décrivons l'outillage que nous utilisons dans notre approche de vérification formelle de propriétés de sécurité. L'outillage de vérification exploite des scenarios de contextes afin de réduire la complexité des modèles durant les vérifications.

Dans le quatrième chapitre, nous présentons l'architecture dirigée par les modèles (MDA) qui est une approche « centrée sur le modèle » bien connue pour soutenir le processus de développement logiciel [4]. L'objectif de ce chapitre est de développer un langage de modélisation visuel basé sur UML qui prend en charge un large éventail de modèles de contrôle d'accès de manière « générique » tout en restant aussi simple que possible. L'idée clé de notre travail est d'utiliser un « méta-modèle de contrôle d'accès ». Nous proposons le méta-modèle E-UACML [9] (Emergency Unified Access Control Modeling Language) qui étend les diagrammes d'activité UACML pour représenter la politique RBAC d'urgence (E-RABC) et le modèle STRBAC.

Le cinquième chapitre présente notre approche [11][9] pour la formalisation des politiques de contrôle d'accès ou nous présentons les principes de transformation et de vérification ainsi que les expérimentation sur un cas d'étude. Dans ce chapitre nous introduisons une méthodologie (architecture globale) et une chaine de transformation (tool-chain) pour formaliser les propriétés de sécurité. Les modèles E-UACML [9], Fiacre [10] , OBP/CDL2 sont utilisés pour exprimer et vérifier des contraintes de

contrôles d’accès. Nous utilisons dans cette architecture notre modèle E-UACML qui est un méta-modèle de contrôle d'accès. Une conclusion et des perspectives termine cette thèse.

PARTIE II: MODELES DE CONTROLE D'ACCES ET UML

Chapitre 1: Modèles de contrôles d’accès

SOMMAIRE

1 Modèles de contrôles d'accès

La sécurité est un aspect important dans le développement et la gestion des systèmes informatiques modernes, et plus spécifiquement le contrôle d'accès. Ce dernier est devenu un probléme majeur pour les développeurs.

Plusieurs types de contrôle d'accès ont été proposés : le DAC (Discretionary Access Control) [2] , le MAC (Mandatory Access Control) [1], et enfin le RBAC (Role Based Access Control) [3]. Dans ce chapitre, nous introduisons les concepts de base du contrôle d'accès et leur gestion. Nous présentons les concepts de base du contrôle d'accès et nous soulignaons les principaux objectifs de recherche de la thèse.

Après un aperçu général des principaux modèles de contrôle d'accès proposés dans la littérature, nous nous concentrons sur le nouveau modèle de contrôle d'accès basé sur les rôles d'urgence « Emergency role-based access control» (E-RBAC) [8].

1.1 Contrôle d'accès

Les systèmes de contrôle d'accès sont la première ligne de défense pour la protection des systèmes informatiques. Ils sont définis par des règles qui établissent dans quelles conditions la demande d'un sujet visant à accéder à une ressource doit être autorisée ou refusée. En pratique, cela revient à restreindre les droits d'accès physiques et logiques des sujets aux ressources système.

[12] a défini le contrôle d'accès comme « centre de gravité traditionnel de la sécurité informatique », où l'ingénierie de la sécurité rencontre l'informatique, dont la fonction est de contrôler quels principaux (personnes, processus, machines, etc.) ont accès aux ressources. Le système, les fichiers qu'ils peuvent lire, les programmes qu'ils peuvent exécuter, comment ils partagent des données avec d'autres entités, etc.

Ainsi, le contrôle d'accès fournit des mécanismes pour déterminer qui ou quoi peut visualiser ou utiliser des ressources dans un environnement informatique.

Au fil des années, différents modèles de définition des contrôles d'accès ont été proposés et exploités. Les modèles traditionnels attribuent des droits d'accès sur la base de l'identité des sujets et des ressources, soit directement, par exemple : Matrice de contrôle d'accès (ACM) [13] ou via des fonctionnalités prédéfinies, telles que des rôles ou des groupes par exemple : le contrôle d'accès basé sur les rôles [3] . Cependant, Les besoins en sécurité sont diversifiés aujourd'hui. En effet, on rencontre de plus en plus de situations dans lesquelles, un seul modèle de contrôle ne suffit plus pour répondre aux exigences de sécurité d'un système. Dans ce genre de situation, Il est nécessaire de

combiner deux, voire plusieurs concepts (rôles, groupes, niveaux de sécurité, etc.) pour renforcer le contrôle d’accès. Par exemple, avec l’utilisation du Cloud Computing, on peut avoir besoin de combiner les trois modèles de contrôle d’accès de base DAC, MAC et RBAC (contrôle d’accès hybrides).

Comme notre travail vise à formaliser les proprieté de sécurité, nous avons choisi le contrôle d’accés comme base, car il fournit un moyen de protéger les données dans les environnements industriels et parce qu'il est souvent utilisé et connu par les entreprises.

1.1.1 Concepts de base: Cycle de vie de l'accès

Le cycle de vie du contrôle d'accès commence par une demande d'accès (demande) émise par un sujet qui demande au système d'effectuer une certaine action sur un certain objet, voir Figure 1.1. Selon les caractéristiques et le contexte de l'utilisateur, le système étudiera les autorisations accordées à cette entité et répondra à sa demande par des réponses binaires : un « Oui » permettra au sujet d'effectuer l'action souhaitée sur l'objet ou un « Non » qui annulera cette demande. Afin de bien comprendre ce processus, nous donnons par la suite les définitions des principales entités qui existent dans un système d'autorisation.

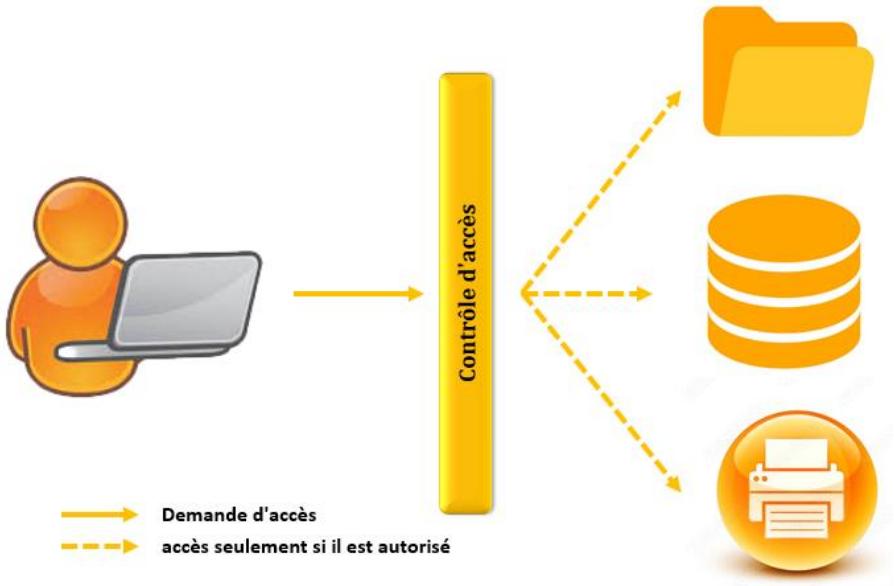

Figure 1.1 Cycle de vie du contrôle d'accès.

Ces termes sont définis comme suit [14]:

Sujet : un sujet représente une entité active, généralement sous la forme d'une personne, d'un processus ou d'un dispositif, qui fait circuler l'information parmi les objets ou modifie l'état du système.

Objet : un objet représente une entité passive qui contient ou reçoit des informations. L'accès à un objet implique potentiellement l'accès aux informations qu'il contient. Les exemples d'objets sont : les enregistrements, les blocs, les pages, les fichiers ou les répertoires.

Action : une action ou une opération représente un processus actif invoqué par un sujet sur des objets. Des exemples d'actions sont : Lire, Écrire, Exécuter, etc.

Autorisation (privilège) : Une autorisation attribuée à un sujet pour effectuer une action sur le système. Dans la plupart des publications sur la sécurité informatique, le terme autorisation fait référence à une combinaison d'objet et d'opération. Une opération particulière utilisée sur deux objets différents représente deux autorisations distinctes, et de même, deux opérations différentes appliquées à un même objet représentent deux autorisations distinctes [15].

1.2 Propriétés de sécurité

Historiquement parlant, la sécurité informatique a commencé à être mise en œuvre pour protéger le matériel physique contre plusieurs attaques. Ces attaques étaient principalement des vols et des dommages de matériel et d'informations dans le but de provoquer une interruption des services et des coûts financiers. Pour éviter de tels problèmes, des modèles de sécurité ont été proposés. Ces modèles intègrent généralement les principes suivants: confidentialité, intégrité, disponibilité, authentification, non-répudiation, et la vie privée. Les sous-sections suivantes décrivent ces propriétés.

1.2.1 Confidentialité

La confidentialité vise à dissimuler l'information des ressources en veillant à ce que les personnes non autorisées n'y aient pas accès. La confidentialité est basée sur le cryptage des données, un mécanisme garantissant que seules les bonnes personnes (les personnes qui connaissent la clé) peuvent lire l'information. Pour ce faire, des fonctions mathématiques sont utilisées pour transformer une information voulue (c'est-à-dire un texte clair) en une information chiffrée (c'est-à-dire un texte chiffré). Une fois chiffrée, l'information peut être déchiffrée et lisible. Ainsi, seules les personnes qui connaissent

la clé peuvent accéder à l'information. La cryptographie moderne peut être divisée en deux familles : la cryptographie symétrique et asymétrique.

La cryptographie symétrique utilise la même clé pour le cryptage et le décryptage, ce qui signifie qu'une clé commune est partagée entre l'expéditeur et le destinataire. Le chiffrement par clé symétrique le plus couramment utilisé est AES [16], une norme proposée par le NIST5 en 2001.

La cryptographie asymétrique (également connue sous le nom de cryptographie à clé publique) utilise deux clés différentes, mais liées mathématiquement. La première clé est publique, ce qui signifie qu'elle peut être partagée et transmise à tout le monde. La deuxième clé est privée, ce qui signifie qu'elle doit être gardée secrète. Les algorithmes les plus couramment utilisés pour fournir une cryptographie asymétrique sont RSA[17] ou l'échange de clés Diffie-Hellman [18].

La figure 1.2 donne un aperçu de la façon dont fonctionnent les cryptages symétriques et asymétriques.

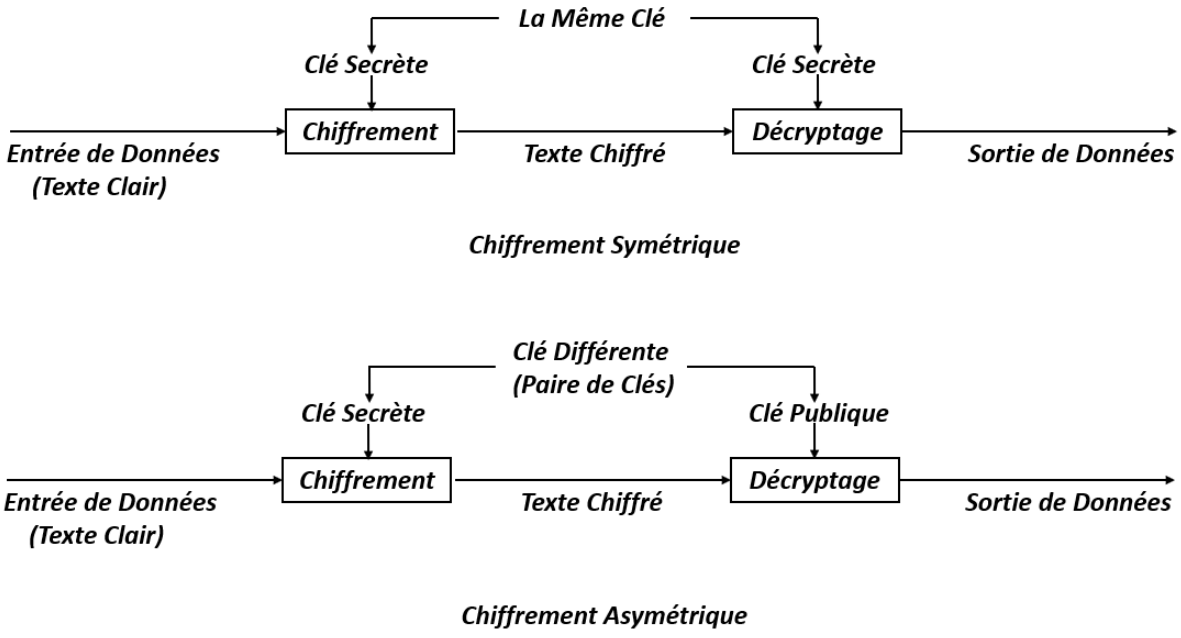

Figure 1.2 Représentation graphique du chiffrement symétrique et asymétrique [16][17].

1.2.2 Authentification

L'authentification a été proposée depuis les années 70 [19][20][21]. Ce principe assure qu'un utilisateur ou un appareil est celui qu'il prétend être. Le facteur d'authentification le plus couramment utilisé est la combinaison de connexion et de mot de passe bien connue. Les mécanismes d'authentification ont été élargis au fil des années et trois familles de facteurs sont généralement soulignées.

La première famille est le facteur de connaissances, qui incorpore ce que les utilisateurs ou les machines connaissent (par exemple mot de passe, ID). La deuxième famille est le facteur de possession, qui incorpore des choses qui sont en la possession des utilisateurs (par exemple des dispositifs matériels tels que des jetons ou des cartes). Enfin, la dernière famille est le facteur d'hérédité. Ces facteurs incorporent des éléments qui font partie d'un utilisateur, tels que l'authentification biométrique (par exemple, balayage rétinien, reconnaissance de l'iris, balayage d'empreinte digitale, reconnaissance vocale, reconnaissance faciale) [22], les traits cognitifs (par exemple, les défis qui visent à sélectionner les images correctes parmi un ensemble) [23] et d'autres mécanismes basés sur le comportement (par exemple, défilement d'une page, vitesse et dynamique des frappes) [24].

1.2.3 Non-répudiation

Non-répudiation [25] garantit qu'une partie ne peut pas nier avoir reçu ou envoyé un message ou un document. Ce principe est intéressant dans les scénarios impliquant la confiance lors d'échanges sensibles. Ainsi, ce principe de sécurité est souvent utilisé dans certains types de sociétés, telles que les banques, les assurances et les agences gouvernementales. Au fil des années, différents types de non-répudiation ont été définis, en fonction de qui (expéditeur ou destinataire) applique le mécanisme de non-répudiation [26].

Du point de vue de l'expéditeur, on serait prêt à s'assurer que son message a été reçu par le destinataire (non-répudiation de la réception (NRR)), a été envoyé au destinataire (non-répudiation de la soumission (NRS)) ou que le message a été correctement remis au destinataire (non-répudiation de remise (NRD)).

Du point de vue du destinataire, on serait prêt à s'assurer que le message qu'il a reçu a été envoyé par un expéditeur authentique (non-répudiation d'origine).

1.2.4 Vie privée

Historiquement, la vie privée a été décrite comme le « droit de l'individu à être laissé seul » [27]. Plus près des problèmes de sécurité informatique, [28] a décrit la protection de la vie privée comme l'intérêt que les individus ont à maintenir un « espace personnel », sans interférence de la part d'autres personnes et organisations. Le concept de la vie privée englobe de nombreux domaines de recherche et domaines, y compris la législation, la sociologie, la psychologie, l'anthropologie et la sécurité informatique. Dans ce dernier domaine, la confidentialité concerne la possibilité pour un individu (ou une organisation) de déterminer quelles informations peuvent être partagées avec des tiers. Ainsi, la vie privée est assez complexe à réaliser, car c'est une notion très large et subjective. Cependant, plusieurs initiatives ont été proposées au fil des années pour définir et fournir des cadres efficaces et respectueux de la vie privée. Ces initiatives incluent par exemple «Privacy by Design» [29] ou le «Droit d'être oublié» [30].

1.3 Solutions pour fournir un contrôle d'accès

Dans cette section, nous présentons d'abord le principe du moindre privilège (PoLP) et la séparation des tâches (SoD), deux principes fondamentaux du contrôle d'accès (AC). Nous présentons ensuite un modèle d'abstraction à trois niveaux souvent utilisé pour représenter le contrôle d'accès.

Après ces trois niveaux, nous présentons les principaux travaux proposés pour définir les politiques de contrôle d'accès, les modèles de contrôle d'accès et les mécanismes de contrôle d'accès. Enfin, nous soulignons les principaux inconvénients du contrôle d'accès.

1.3.1 Principe du moindre privilège (PoLP)/(PoLA).

Le principe du moindre privilège (PoLP) [31] (également connu sous le nom de principe de la moindre autorité (PoLA)) a été défini comme un principe qui « exige que l'utilisateur n'ait pas plus de privilèges que nécessaires pour effectuer un travail. Le travail de l'utilisateur consiste à déterminer l'ensemble minimum de privilèges requis pour effectuer ce travail, et à restreindre l'utilisateur à un domaine avec ces privilèges et rien de plus [31]. En d'autres termes, ce principe vise à limiter les dommages potentiels de toute atteinte à la sécurité accidentelle ou malveillante en accordant aux personnes le plus bas niveau de droits d'utilisateur qu'elles peuvent avoir tout en continuant à faire leur travail. Ainsi, ce principe est souvent utilisé lors de la définition du contrôle d'accès.

1.3.2 Séparation des tâches (SoD)

Un autre concept important de contrôle d'accès est la séparation des tâches (SoD) [32]. SoD limite la quantité de pouvoir détenu par un individu afin de réduire les risques d'erreurs, fraudes, conflits d'intérêts, etc. En sécurité informatique, SoD tend à améliorer la sécurité (en créant par exemple différents comptes avec des jeux d'actions limités au lieu d'un compte principal par utilisateur). Pour ces raisons, ce concept est intéressant lors de la définition des stratégies de contrôle d'accès.

1.3.3 Représentation du contrôle d'accès

[1] ont défini trois couches de contrôle d'accès : politiques, modèles et mécanismes :

- Les politiques intègrent des règles de haut niveau selon lesquelles le contrôle d'accès doit être réglementé.

- Les modèles représentent la mise en œuvre formelle des politiques.

- Les mécanismes définissent les éléments de logiciel ou d'équipement de bas niveau (matériel) qui implémentent le contrôle d'accès défini par le modèle et donc imposé par les politiques.

Dans le reste de cette section, nous utilisons cette abstraction à trois niveaux pour décrire les principales solutions proposées dans le domaine de la recherche sur le contrôle d'accès.

1.4 Modèles de contrôle d'accès

La complexité de la gestion de l'accès aux sources d'information s'est accrue en raison de l'évolution constante des systèmes d'information. Afin de garantir la qualité de l'accessibilité sécurisée aux ressources du système d'information, plusieurs modèles de contrôle d'accès ont été proposés.

Dans cette sous-section, nous présentons les principaux modèles AC qui ont été proposés au cours des années pour assurer une politique AC efficace.

1.4.1 Contrôle d'accès discrétionnaire (DAC)

Le contrôle d'accès discrétionnaire (DAC) [2] est le plus ancien modèle de protection. Il est le système de protection majoritairement utilisé dans les systèmes d'exploitation actuels (Unix, MSWindows, Mac OSX, ...). La caractéristique principale est que le propriétaire des ressources définit les droits d'accès sur ces dernières. Les droits d'accès

sont donc à la discrétion du propriétaire. Par exemple, sous Unix, le propriétaire d'un fichier définit les droits de lecture, d'écriture et d'exécution de ses fichiers pour les différents utilisateurs du système (lui-même, les membres de son groupe et les autres).

Le Département de la Défense (Department of Defense) des États-Unis (DoD) a défini les critères d'évaluation des systèmes informatiques fiables « Trusted Computer System Evaluation Criteria (TCSEC) » [33] dans les années 80. TCSEC est un ensemble de directives et de normes de sécurité qui a défini le contrôle d'accès discrétionnaire (DAC). Dans les modèles DAC, les utilisateurs peuvent définir, modifier ou partager le contrôle d'accès de leurs ressources grâce à un ensemble de règles. Le DAC a été développé afin de mettre en œuvre le concept de matrice de contrôle d'accès défini par [34] et formalisé par [35]. Une matrice de contrôle d'accès peut être représentée comme une matrice bidimensionnelle où les lignes représentent des sujets, les colonnes représentent des objets et les intersections entre les lignes et les colonnes représentent une action qu'un sujet peut effectuer sur une ressource spécifique. Grâce à cette représentation simple, DAC est utilisé pour limiter l'accès d'un utilisateur à un fichier. De nombreux systèmes d'exploitation modernes tels que la famille Windows, GNU / Linux et Mac OS sont basés sur des modèles DAC. Le DAC peut être défini avec des listes de contrôle d'accès (ACL) ou des capacités.

Listes de contrôle d'accès (ACL) : ACL a été introduit pour la première fois dans [36]. Principalement utilisées dans les systèmes multi-utilisateurs de 1970 [37], les ACL consistent en une liste d'entrées qui informe sur le droit d'accès que chaque utilisateur à une ressource spécifique. En tant que DAC, un ACL est traditionnellement représenté dans une matrice bidimensionnelle. Une telle matrice est illustrée dans la table 1, tandis que le concept général d'ACL est représenté à la figure 1.3.

	o1	o2	o3	o4
s1	ecrire	lire		
s2			lire	lire, ecrire
s3		lire, ecrire	lire	
s4	ecrire			lire

Table 1 Représentation d'une matrice AC.

Les utilisateurs sont représentés en tant que lignes et les fichiers sont représentés en tant que colonnes. Les intersections représentent ce qu'un utilisateur spécifique peut faire pour un fichier spécifique. Dans cet exemple, l'utilisateur s1 peut par exemple lire le document o2.

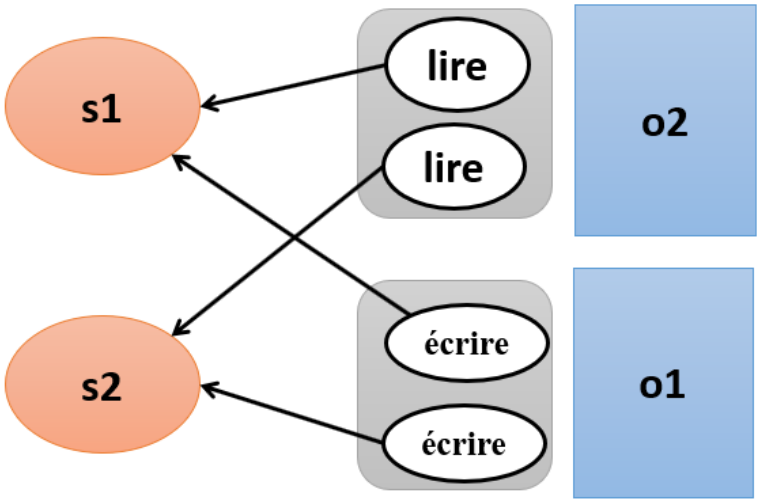

Figure 1.3 Une représentation générique du modèle ACL [36].

Sécurité basée sur la capacité : introduite dans [38], les capacités ont été définies comme « un jeton, un ticket ou une clé qui donne au possesseur la permission d'accéder à une entité ou à un objet dans un système informatique ». Techniquement parlant, une capacité est représentée par une paire (x, r), où x représente le nom d'un objet (par exemple un fichier) et r est un ensemble de privilèges ou de droits (par exemple, en lecture ou en écriture). Grâce à cette représentation (représentée graphiquement dans la Figure 1.4, chaque sujet peut avoir ses propres capacités, lui permettant d'effectuer certaines actions sur les ressources.) De plus, une capacité peut être transférable et doit être infalsifiable.

Cependant, les systèmes basés sur les capacités ont la réputation d'être lents et complexes et ont été remplacés au fil des années par ACL ou un autre modèle dans des systèmes d'exploitation couramment utilisés. Néanmoins, plusieurs travaux ont été proposés afin d'utiliser les capacités dans des contextes modernes, y compris les systèmes distribués [39], Web [40] et l'ingénierie logicielle [41]. De plus, des travaux intéressants ont essayé de populariser ce mécanisme en détruisant le mythe construit autour des capacités des dernières décennies [42].

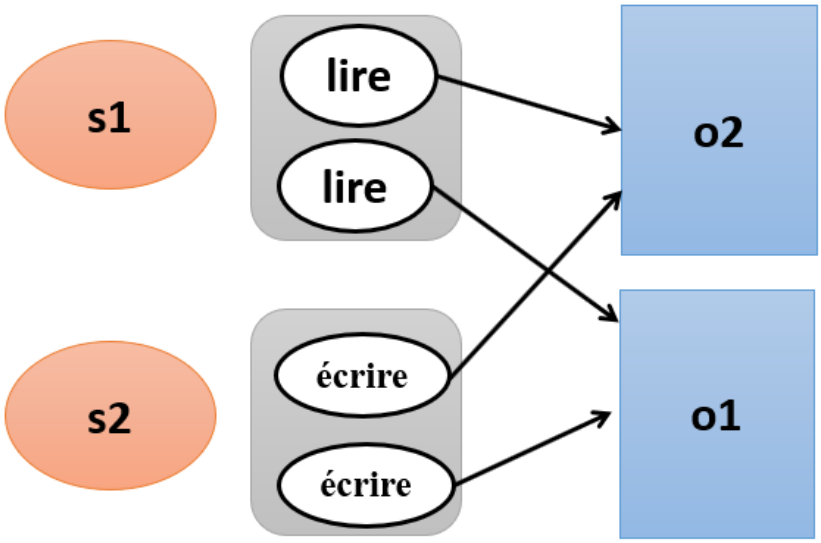

Figure 1.4 Une représentation générique du modèle de capacité [38].

1.4.1.1 Modèle de Lampson

Le modèle de Lampson est proposé dans [34]. Dans ce modèle les entités qui ont accès aux objets sont appelées domaines. Par soucis d'uniformité nous utiliserons le terme sujets. Supposons un ensemble S de sujets et un ensemble O d'objets. Le modèle de contrôle d'accès proposé par Lamps on introduit une notion de matrice d'accès. Cette matrice contient l'ensemble des droits d'accès associés aux sujets et aux objets. Soit *Mso* une matrice d'accès qui associe pour des couples (s; o) des ensembles de droits d'accès. Le modèle peut alors être présenté avec le triplet (S; O; Mso).

Dans la table 1 les droits d'accès «lire» et «écrire» sont associés au couple (s3; o2) qui correspond à l'association d'un sujet s3 et d'un objet o2. s3 peut donc accéder en lecture et en écriture à l'objet o2. Les politiques de sécurité exprimées sur la base du modèle de Lampson sont coûteuses en mises à jour. En effet si de nouveaux sujets, objets ou droits d'accès sont introduits ou supprimés dans la politique, il est nécessaire de produire des modifications sur chaque enregistrement concerné.

1.4.1.2 Modèle de Harrison-Ruzzo-Ullman (HRU)

Dans [35], Harrison, Ruzzo et Ullman présentent un modèle qui généralise celui de Lampson. Soit S un ensemble de sujets, O un ensemble d'objets et *Mso* une matrice correspondant à l'ensemble des droits d'accès associé à un sujet s et un objet o. Comme pour le modèle de Lampson, un sujet peut donner des droits d'accès sur des objets à d'autres sujets. Par exemple si un droit own est associé à un couple (s; o), alors le sujet s

est propriétaire de l'objet o et peut attribuer des droits d'accès à d'autre sujets sur l'objet o.

Dans le modèle HRU, un ensemble de commandes correspondant à des opérations élémentaires est fourni pour permettre aux sujets d'attribuer des droits d'accès sur les objets dont ils sont propriétaires, ou suivant les droits qui leurs ont été concédés. Ces commandes permettent également de créer denouvelles opérations. Les opérations élémentaires sont données ci-dessous :

- entrer un droit r dans Mso
- supprimer r de Mso
- créer un sujet s
- supprimer un sujet s
- créer un objet o
- supprimer un objet o

A titre d'exemple supposons que la création d'un nouveau fichier est suivie de l'affectation d'un certain sujet à ce fichier en tant que propriétaire, avec le droit own. Une nouvelle commande « createf » peut être définie pour exécuter cette opération avec la séquence suivante :

- créer un objet (le fichier)
- entrer le droit *own* dans *Msf* ; *s* étant un sujet et *f* le fichier nouvellement créé.

1.4.1.3 Avantages et inconvénients du DAC :

Le DAC a été décrit comme étant à grain fin par [43]. Grâce à ses contrôles à granularité fine, DAC peut être utilisé pour implémenter l'accès avec les privilèges les plus bas. En effet, des objets spécifiques peuvent avoir des restrictions de contrôle d'accès pour limiter l'accès de chaque sujet aux droits minimum requis. En outre, DAC offre une implémentation intuitive et est invisible pour les utilisateurs [44].

Cependant, le DAC souffre de plusieurs inconvénients. Tout d'abord, parce que les autorisations sont directement appliquées à des sujets individuels, la définition et la maintenance du DAC peuvent être très difficiles au sein des grandes infrastructures ou

des entreprises. De plus, [45] a déclaré que le DAC présentait les principaux inconvénients suivants :

- L'octroi d'accès en lecture est transitif (c'est-à-dire lorsqu'un propriétaire de ressource accorde l'accès à un autre sujet, le sujet autorisé peut accéder à la ressource et permettre à d'autres sujets de la lire sans se référer au propriétaire). Ainsi, aucune restriction ne s'applique à l'utilisation des informations lorsque l'utilisateur l'a reçoit.

- Vulnérabilité a un code malveillant tel que cheval de Troie, car le programme malveillant héritera de l'identité de l'utilisateur appelant.

- L'information peut être copiée d'un objet à l'autre; par conséquent, il n'y a aucune assurance réelle sur le flux d'informations dans un système.

- Les privilèges d'accès aux objets sont décidés par le propriétaire de l'objet, plutôt que par le biais d'une stratégie à l'échelle du système qui reflète les exigences de sécurité de l'organisation.

Maintenant que la famille DAC a été présentée, nous présentons les concepts de contrôle d'accès obligatoire (MAC).

1.4.2 Contrôle d'accès obligatoire (MAC)

Également défini dans (Tcsec) [33], MAC fait référence à une famille de modèles où le système attribue des labels de sécurité ou des classifications aux ressources (par exemple « classifié », « secret » ou « top secret ») et permet l'accès à des sujets ou des applications selon leur niveau de dédouanement. Ce mécanisme de catégorisation est souvent appelé Multi-Level Security (MLS) dans plusieurs travaux [46].

MAC est adéquat aux infrastructures où les politiques de contrôle d'accès ne doivent pas être décidées par les propriétaires et dans les systèmes qui doivent appliquer des décisions de protection (c'est-à-dire que le système a le dernier mot). Le MAC est généralement associé à deux modèles : Bell LaPadula [47], qui assure la confidentialité, et le modèle Biba [48], qui garantit l'intégrité.

1.4.2.1 Modèle Bell-LaPadula (BLP):

Le modèle Bell-LaPadula [47] est une extension du paradigme militaire de sécurité multi-niveau. C'est un modèle qui attribue des étiquettes de sécurité (telles que « Confidentiel » et « Très secret ») aux sujets et aux objets. Grâce à la notion de commandes partielles et totales, BLP définit deux propriétés principales :

- La propriété de sécurité simple : lorsqu'un sujet demande un accès en lecture sur un objet, son niveau d'habilitation doit être supérieur ou égal à celui de l'objet. Cette règle assure la confidentialité de l'information (Non-Read up).

- La propriété * ou simple security property : lorsqu'un sujet demande un accès en lecture sur un objet, son niveau d'habilitation doit être supérieur ou égal à celui de l'objet. Cette règle assure la confidentialité de l'information (No-Write down).

Ces propriétés ont été discutées dans plusieurs travaux [49][50][12] et sont présentées à la figure 1.5.

Modèle de Biba : Alors que le modèle de Bell-LaPadula décrit des méthodes pour assurer la confidentialité des flux d'information, Biba [48] applique des principes inverses pour fournir la propriété d'intégrité (voir Figure 1.5). Dans Biba, les objets et les utilisateurs sont marqués avec des niveaux d'intégrité. Ces niveaux forment un ordre partiel, similaire au modèle BLP. Cependant, Biba applique les deux principes suivants :

- L'axiome d'intégrité simple : qui stipule qu'un sujet à un niveau d'intégrité donné ne doit pas lire un objet à un niveau d'intégrité inférieur ("Non-Read-Down").

- L'axiome d'intégrité : qui stipule qu'un sujet à un niveau d'intégrité donné ne doit écrire aucun objet à un niveau d'intégrité supérieur ("No-Write up").

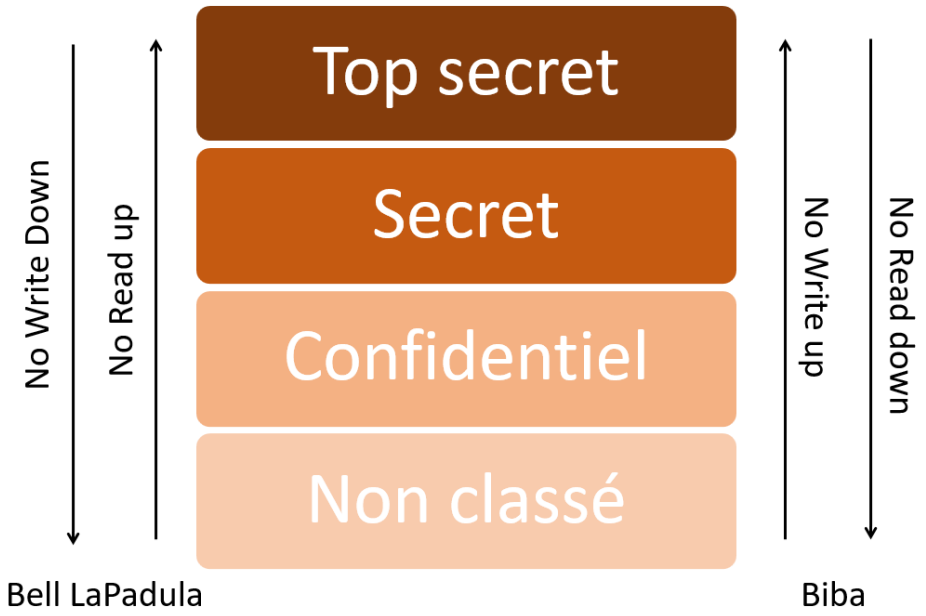

Figure 1.5 Modèles Bell-LaPadula et Biba [47].

Bell-LaPadula [47] assure la confidentialité avec les propriétés «no read up » et «no write down» tandis que Biba assure l'intégrité avec les propriétés «no write up» et «no read down».

1.4.2.2 Modèle de Brewer et Nash

Le modèle de Brewer et Nash dit de la Muraille de Chine a été introduite dans [51] pour résoudre les problèmes de conflits d'intérêts dans lesorganisations commerciales.

L'objectif du modèle est de fournir un contrôle du flux d'informations qui pourrait provenir d'un sujet se trouvant dans une situation de conflit d'intérêts. On parle de conflit d'intérêts lorsqu'un sujet a accès aux informations confidentielles de deux organisations en compétitions.

Les éléments du modèle sont : les objets (éléments appartenant à une organisation), l'organisation (disposant de plusieurs objets), les classes de conflits d'intérêts (regroupant les organisations en concurrence). Ces éléments sont présentés hiérarchiquement de façon à ce que, les objets correspondent au niveau inférieur (bas niveau), les organisations au niveau intermédiaire et les classes de conflits d'intérêts au niveau supérieur (haut niveau).

La Figure 1.6 montre une illustration du modèle de la Muraille de Chine. Comme énoncé précédemment les classes de conflits d'intérêts C1 et C2 sont représentées au

niveau supérieur. Au niveau intermédiaire, on distingue les organisations Org1 et Org2 qui sont dans la classe C1 et Org3 qui est dans la classe C2. Au niveau inférieur on distingue les objets inf1 appartenant à Org1, inf2 et inf3 appartenant à Org2, inf4 appartenant à Org3.

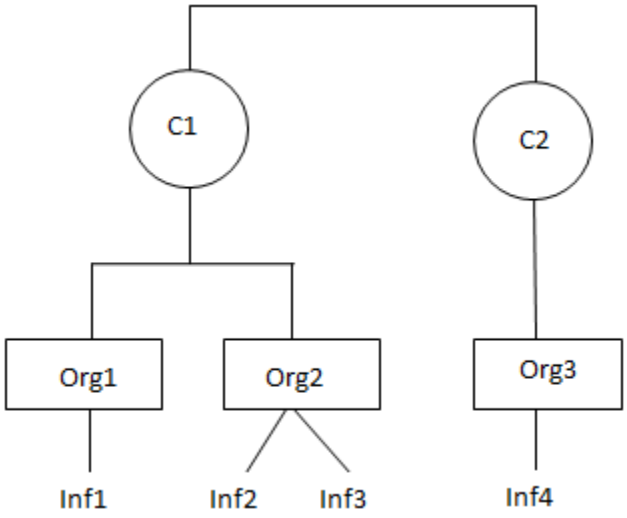

Figure 1.6 Modèle de la Muraille de Chine [51].

Dans le modèle de la Muraille de Chine, un sujet se place dans une classe de conflit d'intérêts lorsqu'il accède la première fois aux informations d'une organisation de la même classe. Les propriétés simples et étoile sont définies pour ce modèle comme suit :

- **Propriété simple :** Cette propriété interdit l'accès en lecture d'un sujet à un objet si cet objet appartient à une organisation de la même classe de conflit d'intérêts que les organisations propriétaires d'objets auxquels ce sujet a déj à accédé en lecture.
  Dans la Figure 1.6, si un sujet accède en lecture à inf1, il ne peut pas accéder aux objets inf2 et inf3 car les organisations Org1 et Org2 se trouvent dans la même classe de conflit d'intérêts C1.

- **Propriété étoile :** Cette propriété interdit à un sujet d'écrire dans un objet d'une organisationsi ce sujet a déjà accédé en lecture aux objets d'une autre organisationde la même classe de conflits d'intérêts. Si un sujet accède en lecture à inf1,il ne peut pas accéder en écriture à inf2 ou inf3.
  Ces deux propriétés doivent être respectées afin d'empêcher un sujet d'écrire dans Org2 des informations qu'il a lu dans Org1.

1.4.2.3 Avantages et inconvénients MAC

Grâce à ses propriétés, Bell-LaPadula (BLP) [47] offre un bon moyen d'assurer la confidentialité dans les infrastructures militaires et critiques. De plus, BLP n'est pas vulnérable aux violations de la sécurité des chevaux de Troie, car les utilisateurs n'ont pas la possibilité de déclassifier les informations [52]. Généralement, les solutions MAC sont simples et évolutives [43]. Cependant, MAC souffre de quelques inconvénients. Parmi ceux-ci, MAC ne contrôle que les informations dans le système qui passent par des canaux manifestes (c'est-à-dire des canaux fonctionnant de manière légitime). Ainsi, les canaux non légitimes (appelés canaux cachés) ne sont pas couverts par MAC. De plus, MAC peut être trop rigide. En effet, la classification des sujets et des objets peut ne pas être réalisable dans des contextes complexes qui nécessiteront de nombreuses catégories différentes.

1.4.3 Contrôle d'accès basé sur le rôle (RBAC)

Le contrôle d'accès basé sur le rôle (RBAC) [3] est basé sur la notion de rôle. Un rôle est un ensemble de sujets partageant des attributs communs (par exemple, un rôle de "medecin" contenant tous les medecins d'un hopital). Dans ce modèle, être membre d'un ou de plusieurs groupes permet aux utilisateurs d'accéder à certaines ressources. Ainsi, RBAC est souvent utilisé dans les entreprises, où la structure organisationnelle est déjà basée sur les rôles. Grâce à ce regroupement, la gestion des politiques est facilitée, en particulier dans les grandes infrastructures complexes. De plus, RBAC couvre le problème du moindre privilège (c'est-à-dire évitant l'attribution de permissions inutiles) et la séparation des tâches (c'est-à-dire la diffusion des tâches et des privilèges associés entre plusieurs utilisateurs). Au fil des années, la première version de RBAC (c.-à-d. RBAC0), illustrée à la figure 1.7, a été améliorée pour répondre aux besoins et aux exigences changeants des infrastructures informatiques modernes [53]. De telles améliorations intègrent la hiérarchie des rôles (RBAC1), la séparation dynamique des tâches (RBAC2), ou les deux (RBAC3).

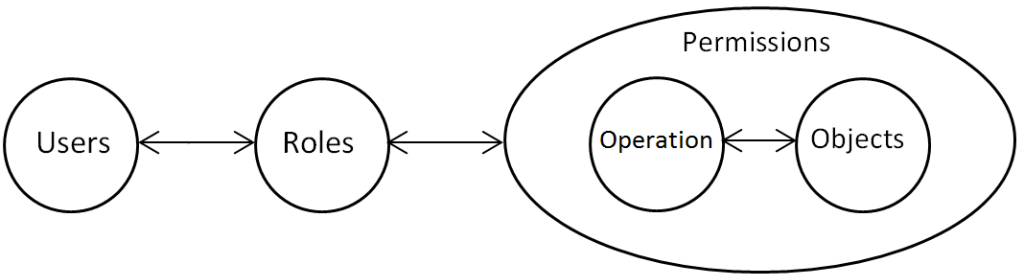

Figure 1.7 Représentation graphique du modèle RBAC.

La figure 1.7 illustre le modèle RBAC de base et ses composants. Les éléments de base sont résumés comme suit [3] :

- **Les utilisateurs (Users)** sont des êtres humains. Il est important de savoir qu'il peut être étendu pour représenter des processus, ou toute entité active du système agissant au nom de l'utilisateur.

- **Les rôles (Roles)** sont des fonctions professionnelles dans le contexte d'une organisation avec une sémantique associée concernant l'autorité et la responsabilité conférées à l'utilisateur affecté au rôle.

- **Les objets (Objects)** sont des données ou des ressources accessibles.

- **Les opérations (Operation)** sont des processus qui exécutent une fonction au nom de l'utilisateur.

- **Les autorisations (Permissions)** sont définies comme l'approbation d'un mode d'accès particulier à un ou plusieurs objets du système. Les autorisations établissent une relation entre les opérations et les objets. C'est-à-dire qu'ils définissent les opérations à effectuer sur les objets. Ainsi, les autorisations sont exprimées comme suit :

$$\mathbf{PRMS = 2^{OPS \times OBS}} \quad [54]$$

C'est-à-dire que $2^{OPS \times OBS}$ désigne le jeu de puissance de l'ensemble **OPS × OBS**

Le modèle RBAC de base définit les relations entre les objets suivants :

- L'assignation d'utilisateur (UA) définit les relations entre les utilisateurs et les rôles. Un utilisateur peut être affecté à plusieurs rôles et que plusieurs utilisateurs peuvent être affectés au même rôle. Il s'agit d'un mappage plusieurs-à-plusieurs, où :

$$\mathbf{UA \subseteq USERS \times ROLES} \quad [54]$$

- L'attribution d'autorisation (PA) définit la relation entre les rôles et les autorisations. Un rôle peut être attribué à plusieurs autorisations. Semblable à UA, il définit un mappage plusieurs-à-plusieurs :

PA ⊆ PRMS×ROLES [54]

Le modèle RBAC de base se caractérise par l'application de deux concepts principaux :

- Le moindre privilège est une action administrative visant à éviter l'attribution d'autorisations aux utilisateurs qui ne sont pas nécessaires pour accomplir une tâche. Le moindre privilège signifie qu'une fois les exigences d'accès déterminées, ce rôle ne doit recevoir que les autorisations nécessaires pour accomplir les tâches requises ; aucune autorisation supplémentaire ne doit être accordée. Cela empêche les utilisateurs de ne pas recevoir plus d'autorisations que nécessaire.

- Séparation des tâches Le concept applique des contraintes sur les rôles coexistant dans une même session et qui peuvent avoir des autorisations conflictuelles, ainsi le modèle réduit le nombre d'autorisations potentielles disponibles pour l'utilisateur. La procédure est utilisée pour prévenir les fraudes.

1.4.3.1 Contrôle d'accès basé sur les attributs (ABAC)

ABAC [55] a été défini comme une méthode de contrôle d'accès où les demandes de sujets pour effectuer des opérations sur des objets sont accordées ou refusées en fonction des attributs assignés au sujet, des attributs assignés au objet, des conditions d'environnement et d'un ensemble de règles spécifiées [56]. La notion d'attributs incorpore divers critères concernant un sujet (par exemple nom, position, sexe), un objet (par exemple nom, niveau de sécurité, extension) ou l'environnement (par exemple période, jour de la semaine, date). Ainsi, ABAC peut être vu comme une extension de RBAC. Au cours des dernières années, de nombreuses solutions ont été proposées pour fournir ABAC [55][57][58]. ABAC s'adapte aux entreprises modernes. De plus, les règles ABAC peuvent être combinées afin de fournir des règles complexes et explicites avec un minimum d'effort. Pour effectuer cette combinaison de stratégies, un moteur d'autorisation est souvent utilisé. Un exemple de ce moteur est illustré à la figure 1.8.

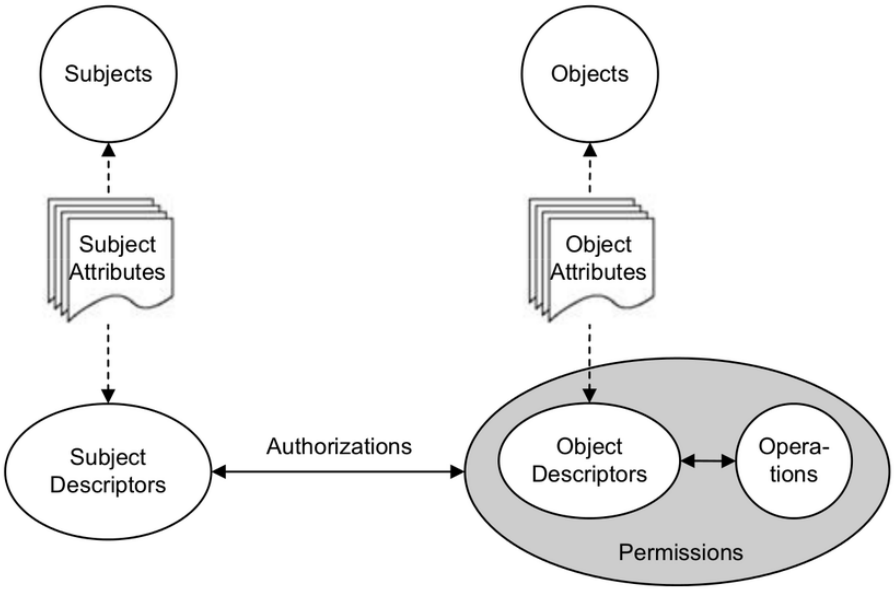

Figure 1.8 Représentation graphique du modèle ABAC [55].

Les attributs provenant de diverses sources peuvent être vérifiés par le moteur pour autoriser ou refuser l'accès en fonction des stratégies définies.

Au cours des dernières décennies, plusieurs modèles ont été proposés pour couvrir le problème du contrôle d'accès. Ces modèles proposent de prendre en compte différentes notions, notamment le contexte [59][60], temporal-RBAC [61].

1.4.3.2 Spatial Temporal Role Based Access Control (STRBAC)

Le STRBAC (Spatial Temporal RoleBased Access Control) [62] qui rajoute des contraintes spatio-temporelles au modèle RBAC. Ce modèle nous intéresse dans notre travail, il est décrit dans la Figure 1.9.

Le but de ces modèles est de permettre des modélisations sans ambiguïtés pour les méthodes formelles et de faciliter la compréhension et l'interprétation des modèles. Nous présenterons donc dans la section suivante quelques technique de vérification de règles de contrôle d'accès.

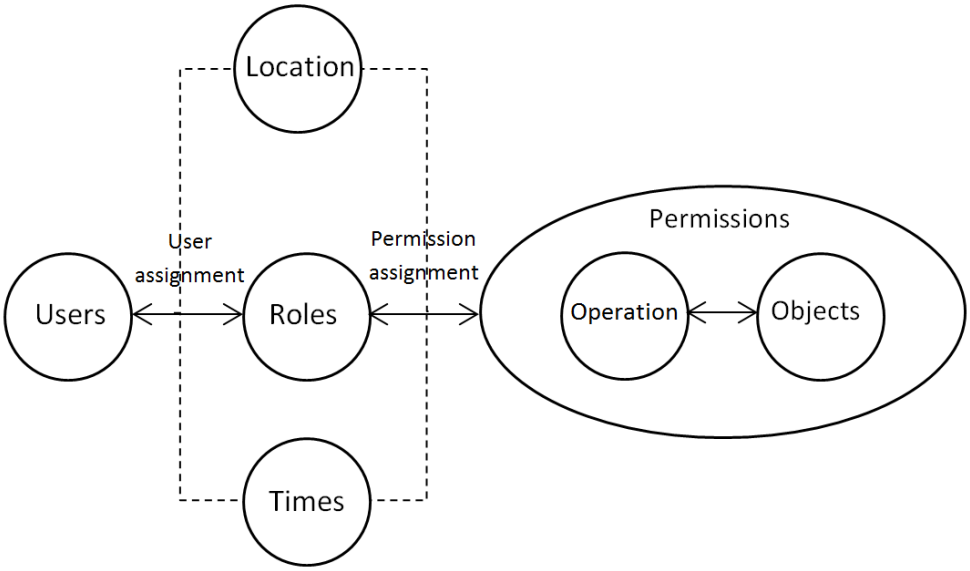

Figure 1.9 Représentation graphique du modèle STRBAC [62].

STRBAC est une extension du modèle RBAC (Role Based Access Control) qui prend en charge les contraintes temporelles et de localisation [62]. Pour l'aspect spatial, STRBAC utilise la définition de la localisation brute (ou localisation physique qui peut être le signal du mobile, du capteur infrarouge ou du dispositif GPS de l'utilisateur) et de la localisation logique. STRBAC limite l'accès aux ressources à partir d'un ensemble particulier d'emplacements prédéfinis : emplacement par adresse (selon l'emplacement physique), emplacement par utilisation (associer l'emplacement à son utilisation), emplacement par organisation (distinguer l'emplacement en fonction du niveau de l'organisation) et l'emplacement défini par l'utilisateur (à d'autres fins). Pour l'aspect temporel, STRBAC propose des intervalles de temps particuliers. Le temps est décrit par une date particulière est appelé intervalle non récurrent (par exemple, chaque date a la date de début et la date de fin). D'autre part, l'intervalle récurrent est représenté par un intervalle particulier (par exemple, quotidien, hebdomadaire, mensuel et annuel). En résumé, STRBAC combine les deux facteurs spatio-temporels dans les conditions d'accès.

Afin d'effectuer un contrôle d'accès basé sur la localisation, il faut effectuer des opérations sur les informations de localisation et protéger les informations de localisation. Il existe deux types d'emplacements : physiques et logiques. Tous les utilisateurs et objets sont associés à des emplacements qui correspondent au monde physique. Ceux-ci sont appelés emplacements physiques. Un emplacement physique est formellement défini par un ensemble de points dans un espace géométrique tridimensionnel.

Ce modèle utilise deux types d'informations temporelles. Il nous semble nécessaire de distinguer ces deux types d'informations car elles ont une sémantique très différente. Le

premier est connu sous le nom d'instantané et l'autre est l'intervalle de temps. Le temps peut être représenté comme un ensemble de points discrets sur la ligne de temps.

Un exemple d'intervalle continu est de 9h00 à 15h00 le 25 décembre. Un exemple d'intervalle non continu est de 9h00 à 18h00 du lundi au vendredi du mois de mars. Certains chercheurs qualifient les intervalles de temps d'expressions temporelles.

1.4.3.3 E-RBAC

Dans les situations d'urgence, les utilisateurs doivent parfois accéder à des ressources non autorisées dans des situations normales. la flexibilité du RBAC est améliorée en proposant un RBAC d'urgence (E-RBAC) [8], qui utilise la politique BTG (Break the Glass) (pour gérer le système en situation d'urgence).

La figure 1.10 illustre le RBAC d'urgence « Emergency role-based access control (E-RBAC) ». Il utilise RBAC [3] augmenté d'obligation [63], puis deux concepts à savoir l'attribut d'utilisateur et le rôle d'urgence sont ajoutés. La section suivante explique l'état du système et les composants du modèle. L'état du système est classé en trois catégories comme suit :

- Situation normale : l'accès des utilisateurs dans cette situation est connu et la politique de contrôle d'accès peut être déterminée formellement.

- Urgence (situation d'urgence anticipée) : cette situation peut prédire des événements tels qu'un incendie et l'accès nécessaire peut également être déterminé. Cependant, l'heure d'occurrence n'est pas précisée et l'accès ne peut pas être donné à l'utilisateur car cela contredit l'idée de moindre privilège.

- Exception (situation d'urgence imprévue) : l'accès requis est inconnu et la politique de contrôle d'accès ne peut pas être prédéfinie.

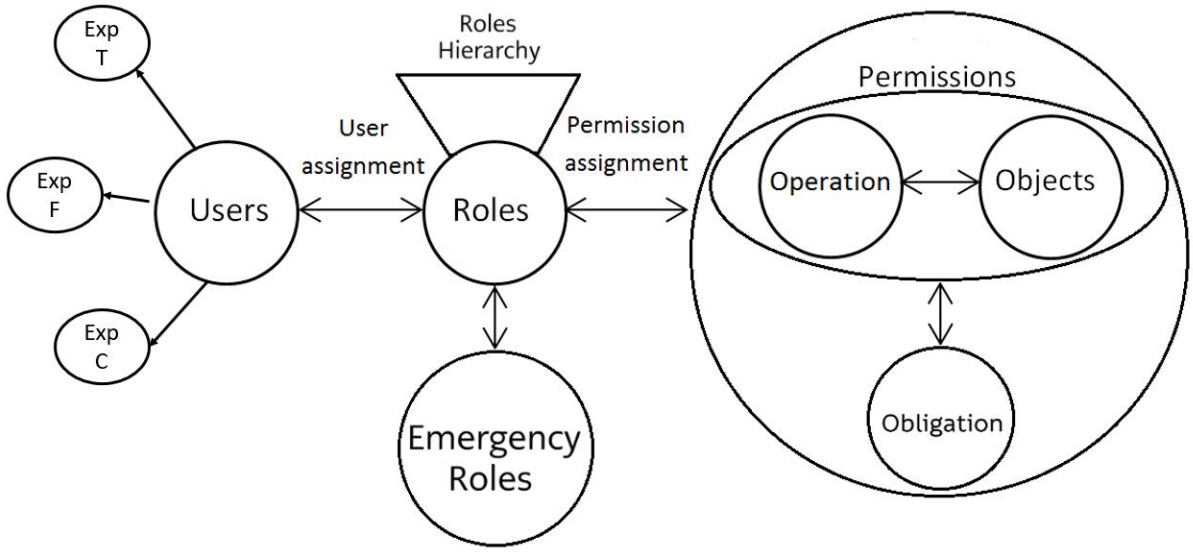

Figure 1.10 Modèle E-RBAC [8].

E-RBAC implique plusieurs éléments de base :

- **Users** (ensemble d'utilisateurs),
- **Roles** (ensemble de rôles dans le système),
- **Emergency Roles** (ensemble de rôles d'urgence pour la gestion des situations d'urgence),
- **Operation** (système opérations),
- **Objects** (ensemble d'objets qui supportent le système),
- **Obligations** (ensemble d'obligations qui doivent être remplies par l'utilisateur avant d'obtenir un rôle),

Les utilisateurs ont des niveaux de confiance (Nc) tels que : FAIBLE, MOYEN ou ÉLEVÉ. Le niveau de confiance est défini pour chaque utilisateur. Pour calculer le niveau de confiance (Nc) pour les utilisateurs, RBAC [3] et ABAC [55] sont intégrés afin que l'utilisateur avec un attribut spécifique puisse être actif en situation d'urgence et d'exception, ainsi l'ensemble d'attributs utilisateur (At) est défini.

- L'attribut d'expérience de travail de l'utilisateur (Exp T),
- L'attribut d'heures de formation de l'utilisateur (EXP F)
- L'attribut de compétence de l'utilisateur (Exp C).

Ces attributs ont été affichés sur la figure 1.10.

1.5 Mécanismes de contrôle d'accès

Les modèles de contrôle d'accès ont été mis en œuvre par le biais de mécanismes matériels et logiciels. Les sous-sections suivantes présentent les principales implémentations qui ont été proposées au fil des année.

1.5.1 Implémentations matérielles du contrôle d’accès :

Les implémentations matérielles ont été définies depuis les années 60. Par exemple, la famille Burroughs (B5000, B6000) [64] a proposé la séparation de code et de données en mémoire, suivi dans les années 70 par le Plessey System 250 [65], le premier système informatique opérationnel à implémenter l'adressage basé sur les capacités. Une approche basée sur les objets a été définie dans des systèmes tels que IBM System/38 [66] et iAPX 432 [67]. Ces systèmes ont proposé une architecture intégrée de sécurité, permettant le contrôle d'accès sur chaque objet. Les détails sur ces systèmes peuvent être trouvés dans [68].

En dépit du système d'exploitation, plusieurs modules (c'est-à-dire l'extension du noyau) ont été proposés. Ces modules incluent des solutions telles que SELinux [69], AppArmor [70] et Smack (noyau de contrôle d'accès obligatoire simplifié) [71].

Les systèmes d'exploitation Windows intègrent le contrôle d'intégrité obligatoire (MIC) [72]. MIC utilise des niveaux d'intégrité et des règles obligatoires pour évaluer l'accès aux objets. Enfin, Apple Mac OS X offre un cadre de contrôle d'accès obligatoire basé sur TrustedBSD (SEDarwin) [73].

Cependant, ces systèmes d'exploitation ont été abandonnés (par exemple, Multics) ou ciblent des secteurs très spécifiques, tels que la défense, le monde universitaire ou la gouvernance.

1.5.2 Implémentations logicielles de DAC :

DAC a également été implémenté de plusieurs façons. Les systèmes d'exploitation les plus modernes tels que les systèmes d'exploitation de la famille Windows, GNU / Linux et Apple Mac, mais aussi d'autres systèmes de type Unix sont basés sur des modèles DAC. Pour implémenter le DAC, ces systèmes utilisent des listes de contrôle d'accès ou des fonctionnalités. La liste de contrôle d'accès (ACL) peut être définie et gérée dans les systèmes de fichiers de GNU/Linux et d'autres systèmes compatibles POSIX. Les autorisations de système de fichiers les plus couramment utilisées sont basées sur la notion de modes. Les modes sont des permissions accordées à "utilisateur" (ie le propriétaire du fichier ou du répertoire), "groupe" (ie ensemble de membres du groupe du fichier) et "autres" (utilisateurs qui ne sont pas propriétaires du fichier ou membres

du groupe). Concernant les autorisations, les modes intègrent des actions communes telles que "lire", "écrire" ou "exécuter" un fichier ou un répertoire. De telles politiques AC peuvent être définies dans des systèmes de type Unix grâce à la commande chmod. Access Control List (ACL) peut également être défini et géré grâce à la ligne de commande setfacl. Cette commande offre une granularité plus fine, permettant par exemple à un administrateur ou à un expert de donner des droits supplémentaires à un utilisateur spécifique.

Des capacités ont été mises en œuvre pour de nombreux systèmes et contextes au fil des ans. Par exemple, TAHOE-LAFS [74] est un système de fichiers distribué qui incorpore des concepts de capacité pour couvrir le principe de la moindre autorité (POLA). Les systèmes d'exploitation tels que EROS[75] ou Coyotos [76] ont été implémentés pour supporter les capacités. Cependant, ces systèmes d'exploitation sont assez marginaux (par exemple, EROS a été proposé pour un usage scientifique uniquement et n'a jamais été déployé dans des entreprises). Ainsi, ces solutions ne sont pas souvent utilisées par les entreprises.

1.6 Problèmes généraux de contrôle d'accès

Au sein des entreprises, les modèles de contrôle d'accès traditionnels offrent de bonnes solutions pour restreindre l'accès aux ressources. Cependant, un employé peut récupérer un document d'un centre de données et envoyer ce document à un collègue non autorisé. Si cet utilisateur a des droits privilégiés sur son ordinateur, il pourra lire ou écrire le contenu du fichier. Une telle action provoquera une fuite de données, car un utilisateur non autorisé aura accès à des informations qu'il n'est pas censé avoir. De plus, la fuite ne sera pas remarquée par l'entreprise.

Des solutions telles que les modèles MLS (Multi-Level Security) et la sécurité basée sur les capacités peuvent éviter ce problème, car les droits d'accès sont intégrés au fichier, empêchant d'autres modifications des droits si l'utilisateur n'a pas la capacité correcte ou le bon niveau d'accréditation. Cependant, MLS et les capacités ne sont pas souvent utilisés dans les entreprises et les logiciels d'aujourd'hui. En effet, les capacités ont été abandonnées au fil des années pour des raisons techniques et de complexité alors que MLS vise souvent des secteurs très spécifiques.

1.7 Conclusion

Dans ce chapitre, nous avons suivi l'évolution des besoins en modélisation de contrôle d'accès et présenté les principaux modèles de contrôle d'accès (DAC, MAC et RBAC) introduits pour répondre aux défis de la gestion des accès dans les systèmes d'information modernes.

À la fin de ce chapitre, nous énonçons les principales raisons de l'utilisation et de l'adoption généralisées du modèle RBAC, qui a réussi à utiliser les rôles pour gérer efficacement les autorisations d'accès au sein des systèmes d'information. Comme indiqué, le RBAC existe sous de nombreuses formes, mais même sa forme la plus simple est considérée comme une amélioration par rapport à d'autres méthodes alternatives.

Les fonctionnalités RBAC telles que la neutralité des politiques, le principe du moindre privilège et la facilité de gestion font des modèles RBAC des candidats particulièrement approprié. Essentiellement, les modèles RBAC peuvent fournir un cadre générique pour exprimer diverses exigences de sécurité [77].

RBAC offre la possibilité de visualiser et de gérer les privilèges des utilisateurs sur des plateformes et des applications hétérogènes. RBAC est capable de définir, contraindre, réviser et appliquer des politiques de contrôle d'accès en tant que relations utilisateur/rôle, rôle/rôle ou rôle/privilège.

Le RBAC est considéré comme neutre en matière de politique dans le sens où, en utilisant des hiérarchies de rôles et des contraintes, un large éventail de politiques de sécurité peut être exprimé pour inclure une variété de politiques de séparation des tâches non discrétionnaire (SOD) via la définition de contraintes.

Chapitre 2: Sécurité et UML

SOMMAIRE

2 Modélisation semi-formelle des politiques de sécurité

2.1 Introduction

Afin de présenter des politiques de sécurité dans des cadres bien définis, plusieurs méthodes de modélisations semi-formelles de règles de contrôle d'accès ont été proposées. Ces méthodes ont essentiellement pour objectif d'aider les intervenants dans des projets à comprendre les grandes lignes des exigences de ces projets. Elles facilitent également l'implémentation des systèmes. Ainsi les méthodes que nous présentons dans ce chapitre, permettent d'exprimer à l'aide de langages semi-formels comme UML (Unified Modeling Language), des politiques de sécurité de façon à respecter les exigences des modèles de contrôle d'accès.

Au chapitre 1, nous avions montré comment des technologies comme le Cloud Computing ou encore des fusions d'entreprises pouvaient obliger les ingénieurs à modéliser des systèmes basés sur des modèles de contrôle d'accès hybrides. Nous montrons dans ce chapitre comment des langages semi-formels nous permettent d'exprimer ces modèles de contrôle d'accès hybrides. Nous mettrons particulièrement l'accent sur la méthode UACML.

2.2 Les modèles semi-formels

2.2.1 SecureUML

SecureUML [5] est un langage de modélisation graphique, qui permet de spécifier sous forme d'un diagramme de classes UML, les informations relatives à la sécurité d'un système. Son méta-modèle (syntaxe abstraite) est une extension du méta-modèle du diagramme de classe d'UML.

Dans ce modèle, Basin et al. [78] ont commencé par définir un modèle de sécurité, qui est une extension de RBAC modélisée en UML. Pour définir un langage en UML, une approche a été adoptée, en utilisant directement des MOF (Meta-Object Facility). Ces derniers se focalisent directement sur le problème ainsi que sur ses domaines sans aucune dépendance de UML (sémantiquement). Le résultat est un langage qui définit des domaines spécifiques par un vocabulaire qui est plus concis que le vocabulaire utilisé par le langage à base d'UML.

Pour modéliser des informations de sécurité, SecureUML intègre les éléments de base du modèle RBAC. La Figure 2.1 présente le méta-modèle du langage. Dans cette figure, les éléments User, Role et Permission sont introduits pour les éléments de mêmes noms dans le modèle RBAC. L'héritage de l'élément ResourceSet indique que des classes du modèle UML peuvent jouer le rôle de ressource c'est à dire un objet.

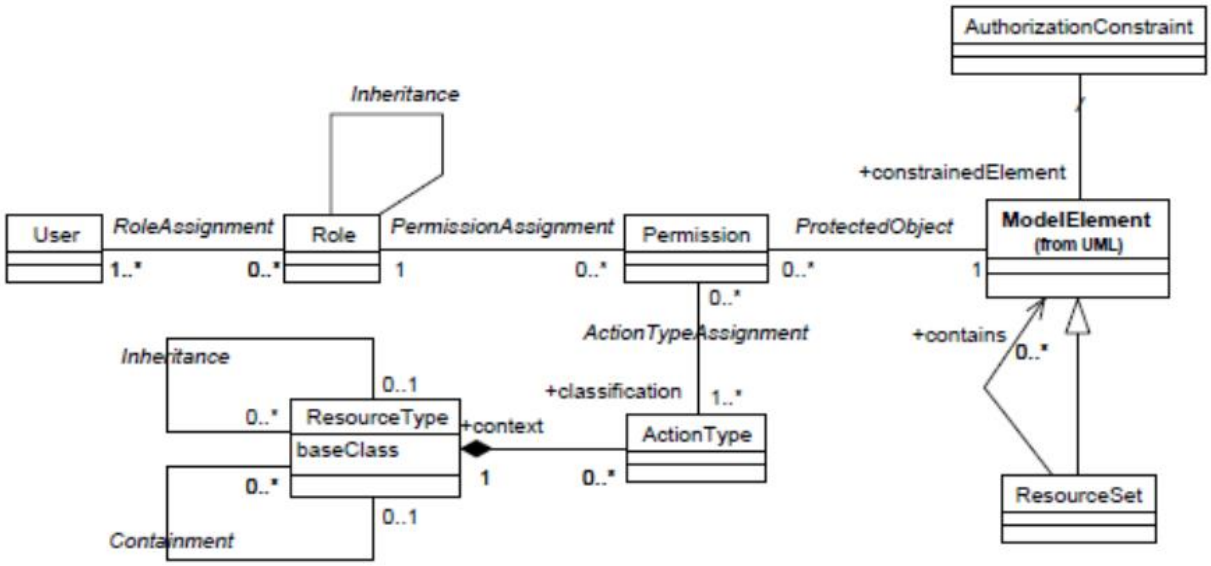

Figure 2.1 Méta-modèle de SecureUML [78].

L'élément Permission désigne un ensemble de permissions et est représenté dans un modèle par une classe d'association entre des éléments Role et ModelElement (une classe UML). Pour chaque permission, il peut exister plusieurs classes d'opérations de type ActionType. Chaque élément ActionType représente une action à exécuter sur une ressource. ResourceType indique au besoin l'ensemble des éléments de type ActionType définis pour un modèle particulier. Enfin AuthorizationConstraint permet d'exprimer des contraintes d'accès dans un modèle.

Supposons la règle de contrôle d'accès suivante : «un individu ayant le rôle étudiant peut s'inscrire à un cours s'il en obtient l'autorisation». Nous nommons cette règle R1 dans le reste de notre document. R1 indique que dans le système à modéliser nous devons identifier toutes les personnes qui ont le rôle Etudiant et nous assurer qu'elles ont bien l'autorisation de s'inscrire avant de leur permettre d'exécuter l'action Inscrire qui conduit à leur inscription effective à un cours. Pour cet exemple, nous supposerons qu'il existe une unique autorisation inscrireCours qui permet de s'inscrire à un cours.

a) L'utilisation

SecureUML laisse ouverte la nature de la ressource à protéger. Le schéma général de la combinaison du modèle de sécurité avec le modèle de conception se fait à travers un dialecte (figure 2.2), ce dernier identifie des primitives du langage de modélisation comme des ressources à protéger.

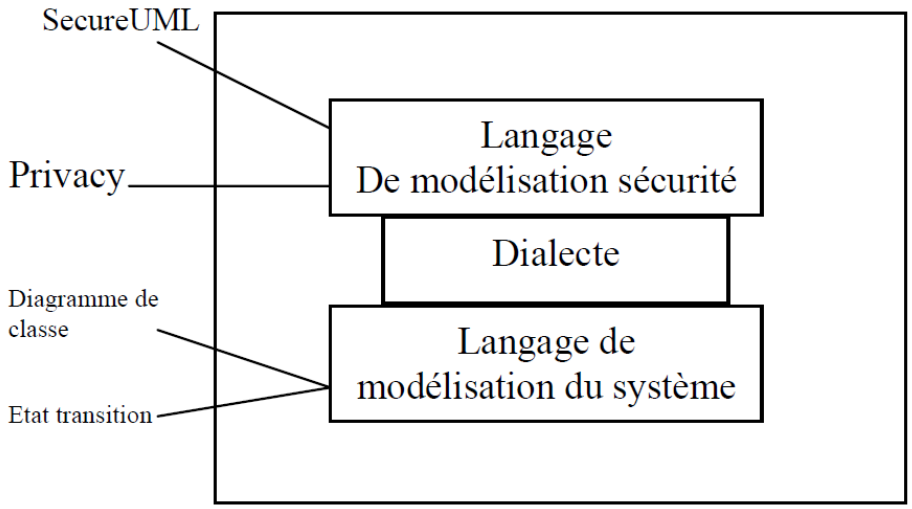

Figure 2.2 Schéma d'un langage de conception de sécurité.

La figure 2.2 nous indique que :

- Le langage de modélisation de sécurité exprime les politiques de sécurité.
- Le langage de modélisation de la conception du système traduit la construction du modèle de conception.
- Le dialecte fournit une passerelle à partir de la définition des points d'interaction entre les deux modèles. Les éléments du modèle de conception sont classés comme ressources à protéger.

b) Sémantique

SecureUML formalise la décision de contrôle d'accès qui dépend de deux types d'information

- La décision de contrôle d'accès déclarative : elle dépend de l'information statique qui est l'affectation d'un rôle à un utilisateur ou une permission.
- La décision de contrôle d'accès par programmation : elle dépend de l'information dynamique, qui est la satisfaction des contraintes d'autorisation dans l'état actuel du système.

2.2.2 UMLsec

UMLsec, proposé par Jürjens [6], est une extension du langage UML qui fournit des artéfacts pour le développement de systèmes sécurisés. La modélisation des éléments de

sécurité dans les modèles UML se fait grâce à la spécification d'un profil UML. Les modèles ainsi obtenus sont évalués par rapport à des vulnérabilités. UMLsec s'applique aux fragments UML suivants : diagramme d'activité, diagramme de classe, diagramme de séquence, diagramme d'états, diagramme de paquetage et diagramme de déploiement. UMLsec fournit des mécanismes de sécurité au langage UML en définissant des profils UMLsec. Ces mécanismes de sécurité sont exprimés grâce à des stéréotypes, des balises UMLsec et des contraintes UMLSec. Un profil UMLsec qu'on note pUMLsec est une combinaison d'un stéréotype, d'une balise et d'une contrainte. Il est décrit de façon formelle par :

$$pUMLsec = (stereotypes, tags, constraints)$$

Avec stereotypes$\in$Stereotypes (ensemble de stéréotypes), tags $\in$ Tags (ensembles de balises) et constraints$\in$Constraints (ensemble de contraintes). Le flux d'information est exprimé sur des systèmes grâce au stéréotype *no-down flow* et à la balise *secret* et permet de s'assurer qu'il n'y a pas de fuite d'informations lorsque les systèmes s'exécutent.

Pour spécifier une politique de flux d'information sur des diagrammes d'états, un profil UMLsec est défini en combinant le stéréotype no down-flow et une balise secrète. On souhaite que lors de l'exécution d'un diagramme d'états, la contrainte no down-flow soit respectée (pas de fuite d'information des variables publiques vers les variables privées). De plus, les variables et méthodes annotées avec la balise secrète sont considérées privées (les autres sont considérées publiques). Ainsi, au niveau du diagramme d'états on vérifie qu'aucune information n'est divulguée durant son exécution. On distingue lors de l'exécution d'un diagramme d'états deux types de méthodes : les méthodes qui écrivent (modifient) dans les variables et les méthodes qui lisent les variables. Les méthodes classées publiques peuvent lire des variables classées publiques, et les méthodes classées privées peuvent lire des variables classées privées ou publiques. De plus, les méthodes classées publiques ne peuvent lire que des variables qui ont été écrites à partir de méthodes classées publiques, et les méthodes classées privées ne peuvent lire que des variables qui ont été écrites à partir de méthodes classées privées ou publiques.

2.2.3 BAAC@UML

Le profil BAAC@UML [79] étend les diagrammes d'activités UML2 [80] pour modéliser les activités des processus métier et le contrôle d'accès à ces activités. L'adéquation des diagrammes d'activités UML2 pour la modélisation des processus métier a été évaluée par plusieurs travaux tels que [81][82]. Les diagrammes d'activité

UML2 incluent divers mécanismes pour représenter les éléments de processus métier tels que le nœud d'activité, le nœud d'action, le nœud de contrôle, le flux de contrôle, l'alternative et la séquence.

Cette approche de modélisation des processus métier part des cas d'utilisation. La spécification de modèles de processus métier à partir de cas d'utilisation correspondant aux besoins des utilisateurs est une bonne pratique dans le développement de systèmes. Plusieurs travaux, tels que [83][84] proposent des approches de description des processus métiers basées sur des cas d'utilisation. Chaque cas d'utilisation encapsule un comportement qui représente les interactions entre le système et ses acteurs. Le profil BAAC@UML détaille ces interactions en considérant une politique RBAC. Deux niveaux pour visualiser le comportement d'un cas d'utilisation sont considérés. Un niveau abstrait (figure 2.3) qui ne détaille pas les permissions RBAC et un niveau concret (figure 2.4) où les conditions préalables de contrôle d'accès sont associées aux actions.

La figure 2.3 montre un exemple de la spécification abstraite du cas d'utilisation de la réponse suivante avec le profil BAAC@UML: L'initiateur visualise la réponse de l'invitation à la réunion et suite à la réponse, il choisit de confirmer l'invitation, d'annuler l'invitation ou de répondre à une demande de changement. Les quatre tâches de l'activité sont placées dans une partition qui spécifie l'acteur qui peut les exécuter.

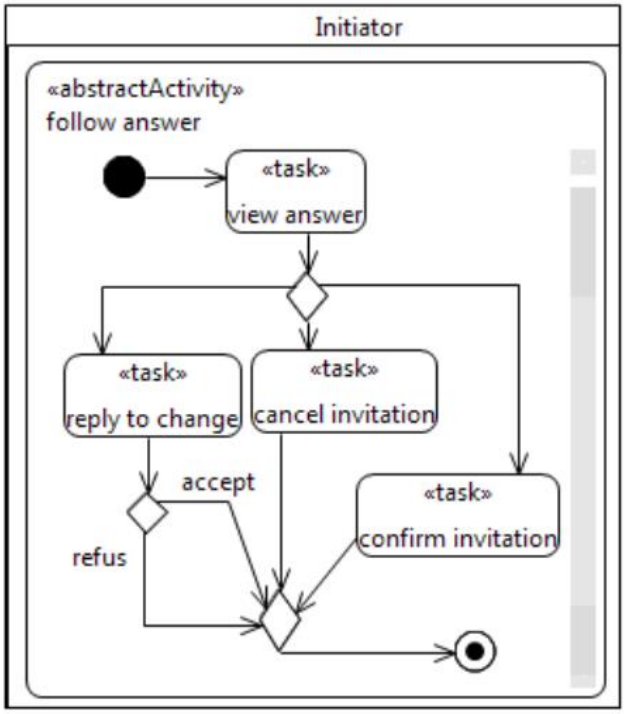

Figure 2.3 Un exemple de la spécification abstraite du cas d'utilisation [79].

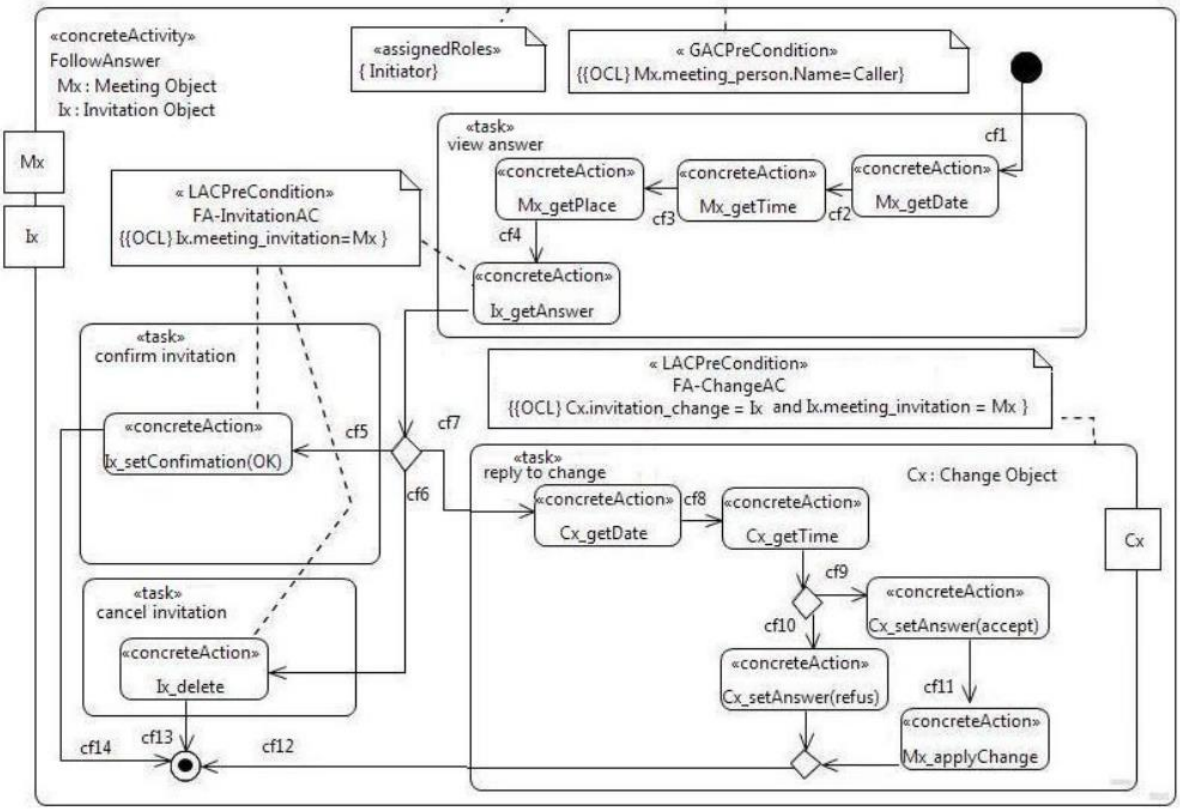

Figure 2.4 La spécification abstraite du cas d'utilisation concrète [79].

2.2.4 AuthUML

authUML [85] est basé sur le Flexible Authorization Framework (FAF) [57] de Jajodia et al., et est une tentative de faire progresser son application à la phase de spécification des exigences du cycle de vie du développement logiciel. Par conséquent, authUML est une version personnalisée de FAF qui doit être utilisée dans l'ingénierie des exigences. A cet effet, authUML utilise des composants similaires de FAF avec quelques modifications dans le langage et le processus pour s'adapter au modèle de cas d'utilisation utilisé dans UML. Parce que FAF spécifie des modules d'autorisation dans les systèmes informatiques, FAF est invoqué pour chaque demande d'autorisation. En revanche, authUML doit être utilisé par les ingénieurs des exigences pour éviter les conflits et l'incomplétude des accès. Par conséquent, alors que FAF est fréquemment utilisé pour traiter chaque demande de contrôle d'accès pendant l'exécution, authUML doit être utilisé moins fréquemment pendant la phase d'ingénierie des exigences pour analyser les exigences de contrôle d'accès.

authUML se compose de quatre phases principales où chacune se compose de plusieurs étapes. Comme le montre la figure 2.5, authUML prend les autorisations à partir des

spécifications des exigences, puis les analyse pour produire des autorisations complètes, cohérentes et sans conflit.
La première phase est la reformulation des besoins de contrôle d'accès sous forme d'instanciation de variables de prédicats (proposée par AuthUML) : cette phase permet d'unifier la forme d'écriture, ainsi que la vérification de la consistance et de l'absence de conflit des besoins de contrôle d'accès.

Dans la deuxième phase, les accès (formulés dans la première phase) sont propagés aux hiérarchies d'acteurs et aux cas d'utilisation, et toute inconsistance est résolue par les règles de résolution de conflit.

Après ces deux phases, tous les accès sont consistants, mais reste le problème de la complétude, qui est résolu par la prise de décision, qui attribue des autorisations soit à une politique fermée, soit à une politique ouverte.

- Politique fermée : pas de permission d'accès, pas d'accès.

- Politique ouverte : pas d'interdiction d'accès, l'accès est permis.

Après cette étape, la consistance est revérifiée par rapport aux besoins de sécurité. La troisième phase applique le même procédé que celui de la deuxième phase, mais n'a pas de prise de décision parce que les cas d'utilisation propagent leurs accès à toutes leurs opérations.

La quatrième phase est différente des autres phases, celle-ci n'est pas exécutée d'une manière séquentielle, mais intervient à tous les niveaux précédents. Elle permet de contrôler les séparations des devoirs au niveau des besoins, l'attribution des rôles aux utilisateurs, et l'exécution des opérations.

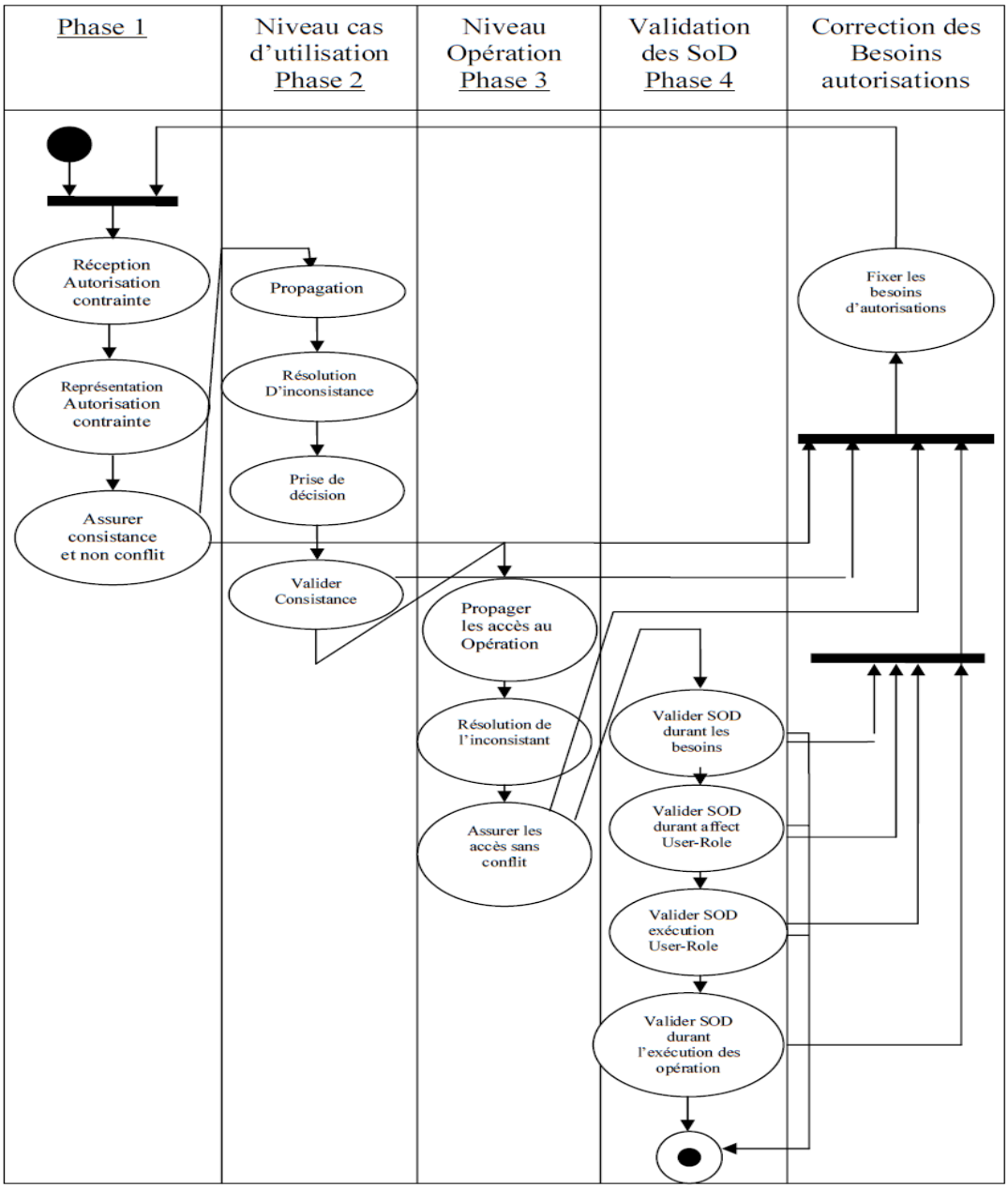

Figure 2.5. Architecture AuthUML.

2.2.5 UACML

UACML [7] est un méta-modèle de contrôle d'accès basé sur UML et conçu pour modéliser des politiques de sécurité pouvant être hybrides. Contrairement à SecureUML qui est basé sur le modèle RBAC, UACML est indépendant d'un modèle de contrôle d'accès particulier. Cette neutralité lui permet de supporter des règles de contrôle d'accès basées sur plusieurs modèles (RBAC, MAC, etc.).

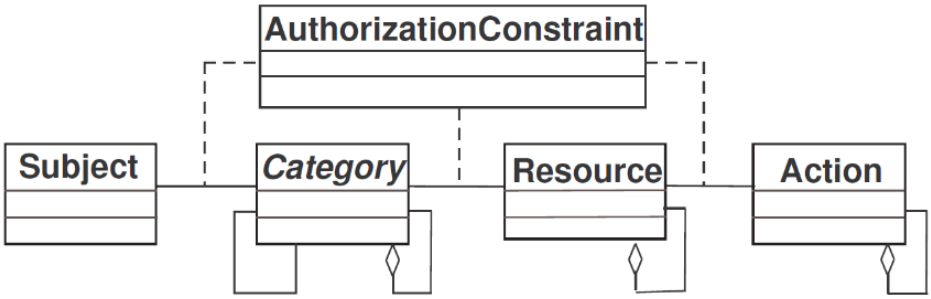

Figure 2.6 Méta-modèle de UACML.

La syntaxe du langage UACML est basée sur une extension du méta-modèle de contrôle d'accès de Barker [86]. La figure 2.6 illustre le méta-modèle. Essentiellement, la sémantique du langage signifie que chaque bord d'association représenté entre les classes définit une relation sur les classes.

Le méta-modèle UACML est présenté à la figure 2.6. L'élément Subject représente des sujets; Resource correspond aux objets. Action représente les actions qu'un sujet peut exécuter sur une ressource. Category est l'élément central du méta-modèle UACML. Il permet de spécifier des concepts propres à chaque modèle de contrôle d'accès. Par exemple, on peut instancier l'élément Category vers les éléments rôles pour RBAC, vers des éléments niveaux de sécurité pour MAC, etc... Enfin AuthorizationConstraint permet de spécifier des contraintes sur les différentes associations présentes entre les éléments d'un modèle UACML.

UACML fournit donc assez de généricité pour contenir la plupart des modèles de contrôle d'accès et pour construire des modèles hybrides. Toutefois son méta-modèle impose la notion de Category dont l'ensemble des instances doit toujours relier un sujet à une ressource. On peut donc difficilement créer un modèle dans lequel des instances de Category seraient destinées à être directement reliées à l'élément Action. Par exemple il sera difficile de modéliser des groupes d'actions dans un modèle UACML.

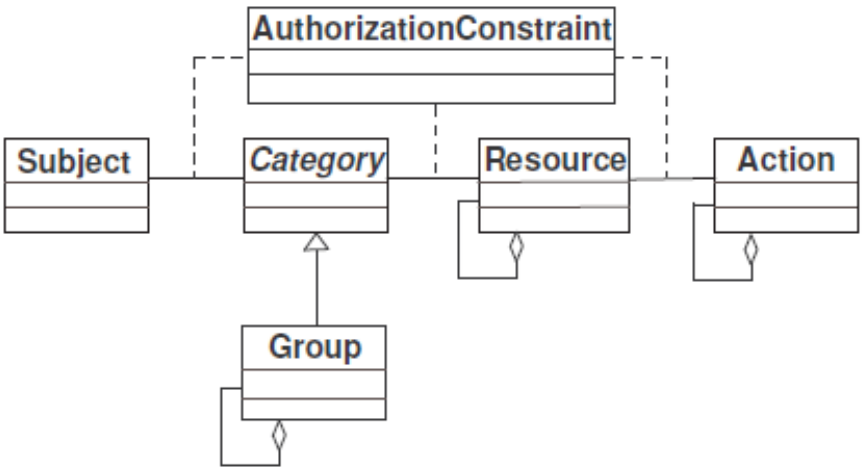

Figure 2.7 Contrôle d'accès basé sur les groupes.

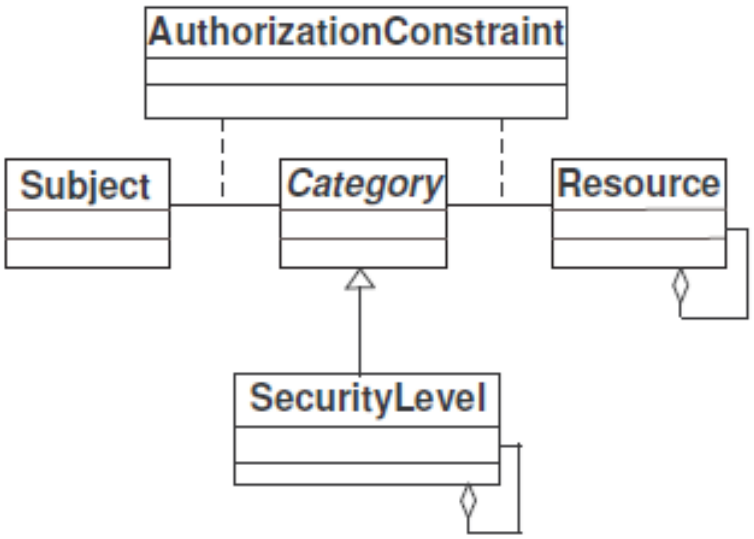

Figure 2.8 Contrôle d'accès obligatoire.

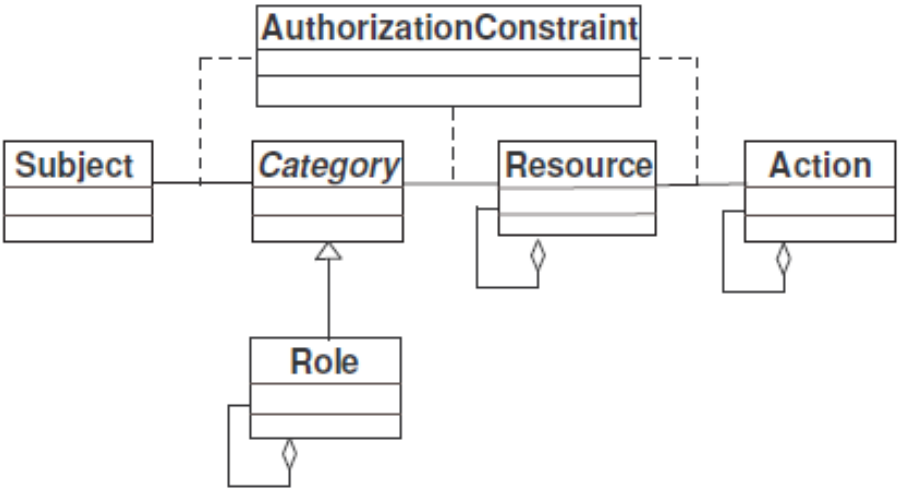

Figure 2.9 Contrôle d'accès basé sur les rôles.

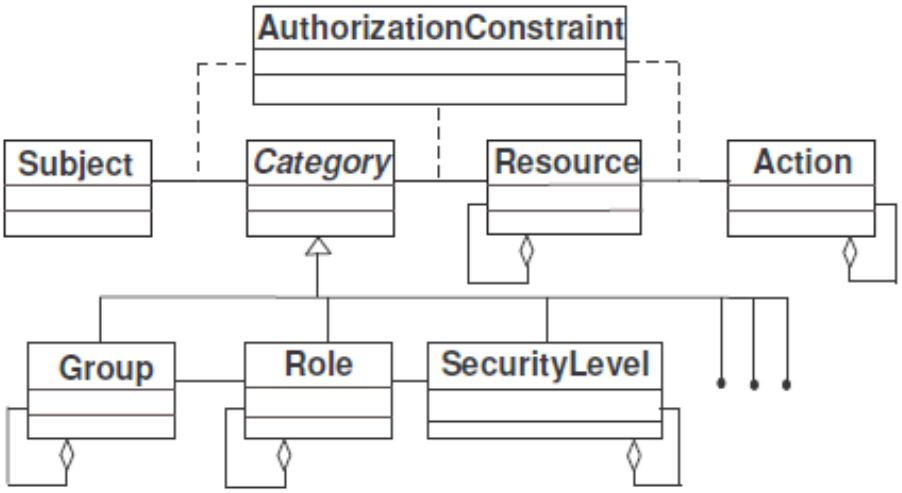

Figure 2.10 Contrôle d'accès hybride.

La Figure 2.10, extraite de [7], montre un modèle hybride contenant un groupe, un rôle, et un niveau de sécurité. Les figures (2.7, 2.8 et 2.9) présentent respectivement un modèle basé sur les groupes, un modèle basé sur les niveaux de sécurité, et le modèle RBAC.

2.3 Conclusion

Dans ce chapitre nous avons présenté des méthodes semi-formelles de modélisation de politiques de sécurité. Nous avons porté notre attention sur les modèles basés sur des diagrammes de classes UML et sur les modèles UACML en particulier, car UACML est suffisamment générique pour modéliser un nombre important de modèles de contrôle d'accès. Cependant il existe d'autres méthodes de modélisation qui ne sont pas basées sur les diagrammes de classes UML. Notre travail consiste à créer des modèles de contrôle d'accès hybrides. Avec les enjeux que représente la sécurité de l'information, une modélisation semi-formelle avec UACML ne serait être suffisante pour exprimer adéquatement une politique dans son ensemble. Nous avons donc choisi de compléter les modélisations UACML avec le modèle STRBAC et E-RBAC. Nous présentons cette méthode au prochain chapitre.

PARTIE III: CONTRIBUTIONS

Chapitre 3: Technique de vérification par model-checking

SOMMAIRE

3 Techniques de vérification par model-checking

3.1 Introduction

Le but de cette section est de décrire la technique de vérification formelle des propriétés utilisée dans notre approche. Nous décrivons la chaîne d'outil OBP [87] utilisée, la spécification en langage CDL [88], des propriétés, d'une part, et d'autre, part des scénarios. Le CDL utilise des logiques temporelles telles que CTL, LTL et CTL* [89][90][91][92]. Lors de la phase de spécification, ces logiques peuvent être utilisées avantageusement du fait qu'elles expriment le comportement d'une manière non ambiguë.

Le but de cette section est d'introduire le formalisme CDL pour l'expression des scénarios et des propriétés de contrôle d'accès qui sera utilisé dans notre approche.

Ainsi, nous décrivons l'outillage que nous utilisons dans notre approche de la vérification formelle des propriétés de contrôle d'accès. L'outillage de vérification exploite des scenarios de contextes afin de réduire la complexité des modèles durant les vérifications.

3.2 Model-checking

Selon la définition de [93], un model-checking « Model-checking is an automated technique that, given a finite-state model of a system and a logical property, systematically checks whether this property holds for that model » est un ensemble de techniques de vérification automatique de propriétés temporelles. L'algorithme de model-checking prend en entrée :

- une abstraction du modèle (par exemple notre modèle E-UACML [9]), sous forme d'un système de transitions.

- une formule d'une certaine logique temporelle, et répond si l'abstraction satisfait ou non la formule.

On dit alors que le système de transitions est un modèle de la formule, d'où le terme anglais « model-checking ». Plusieurs attentions ont été prêtées aux techniques de model-checking. Ceci est fondamentalement dû à leur capacité d'être implémentées efficacement avec les différentes technologies récentes tels que les microprocesseurs, les BDDs (Binary Decision Diagrams), SAT (Satisfiability) solveurs et les SMT (Satisfiability Modulo Theories), SMT (Satisfiability Modulo Theories) solveurs, et d'autre part fournissent une manière automatique (i.e; push-buttonway) de vérification.

L'avantage principal de model-checking (L'exploration de l'espace des états) est de faire une exploration des états du système jusqu'à exploration de tout l'espace des états en entier, ou bien jusqu'à la détection d'un état d'erreur. De plus, ces modèles n'exigent pas une compréhension profonde des concepts mathématiques.

Model-checking	Outils de vérification	Langage
SPIN [94]	LTL Promela	Non formel
NuSMV [95]	LTL/CTL, BDD	Formel
Maude [96]	LTL, RL	Formel
UPPAAL [97]	Timed automata	Formel

Table 3.1 Résumé des caractéristiques de différents model-checkers.

La table 3.1 représente un résumé de caractéristiques de différents model-checking les plus utilisés.

L'un des avantages de la technique de vérification de modèle est qu'elle est automatique et renvoie un contre-exemple dans le cas où la propriété n'est pas vérifiée. D'un point de vue théorique, la principale limitation de la vérification de modèle est que le système de transition doit être fini, c'est-à-dire que le programme ne doit gérer que des variables de champ finies. D'un point de vue pratique, la principale limitation de cette technique est la taille importante du système de transition due au phénomène d'explosion combinatoire du nombre d'états. Le problème d'explosion combinatoire évolue de manière exponentielle, principalement du fait de l'augmentation du nombre de variables et de leur taille, et d'autre part du nombre de composants du système (cas des composants concurrents).

3.2.1 Model-checking pour la vérification des modèles de contrôle d'accès

La Table 3.2 décrit certaines approches qu'on peut trouver dans la littérature dans le domaine de la vérification des politiques de contrôle d'accès basées sur les automates. L'utilisation des méthodes formelles dans l'analyse du contrôle d'accès a été présentée depuis longtemps et plusieurs langages de modélisation, en plus des automates qui ont été utilisés pour modéliser et analyser les politiques de contrôle d'accès tels que Alloy [99] et les réseaux de pétri [100][101]. SPIN [94] a été proposé pour vérifier des politiques de sécurité dans de nombreux travaux [102][103]. Dans SPIN, on peut exprimer des propriétés (par exemple de vivacité et d'équité) comme des formules de LTL (logique temporelle linéaire), et à partir de ces propriétés, on transforme leurs négations en automates de Buchi.

D’après la Table 3.2, nous remarquons que les approches citées ne traitent qu’un seul modèle de sécurité (RBAC, DAC ou bien MAC). Cependant notre approche propose une formalisation et une vérification des modèles de sécurité basées sur un modèle seul ou combinés (Modèle hybride).

Travaux	Modéliser par	Modèles de contrôle d'accès						Model-checking	Outil de vérification
		MAC	DAC	RBAC	T-RBAC	ST-RBAC	E-RBAC		
Schnieder 2000 [104]	Automate de sécurité	x	x	-	-	-	-	-	-
Shafiq, Joshi, and Ghafoor 2002 [100]	Réseau de Pétri	-	-	-	x	-	-	x	-
Mondal and Sural 2008 [26]	Automate temporisé	-	-	-	x	-	-	x	UPPAAL
Rakkay 2009 [101]	Réseau de Pétri	-	-	x	-	-	-	x	-
Jiague 2012 [106]	Patron de securité	-	x	x	-	-	-	x	ASTD
Geepalla, Bordbar, and Okano 2012 [107]	Automate temporisé	-	-	-	-	x	-	x	UPPAAL
Yunchuan, Lihua, and Chao 2014 [108]	Automate temporisé	-	-	-	-	x	-	x	UPPAAL
Aoki and Matsuura 2014 [109]	Automate fini	-	-	-	-	-	-	x	UPPAAL
Cheminod et al. 2015 [110]	Automate fini	-	-	x	-	-	-	x	-
Vasilikos, Nielson, and Nielson 2017 [111]	Automate temporisé	-	-	x	-	-	-	x	UPPAAL
Notre approche [9][11]	**Automate Ficare + Automate observateur**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**OBP Tool**

Table 3.2 Les modèles d'automate de vérification du contrôle

3.2.2 Le model-checking dans le cycle de développement

Afin de détecter les erreurs au plutôt possibles, il est nécessaire d'appliquer une exploration avant l'étape d'implantation. Pour cela, le processus du model-checking prend place au niveau de la phase de conception. Afin d'appliquer la vérification par model-checking, on a besoin que le système et ses spécifications soient représentés dans un modèle formel.

3.3 La vérification de propriétés par exploration de modèle : L'outil OBP

Pour établir la vérification d'un ensemble d'exigences sur un modèle, il faut disposer d'un modèle simulable et des exigences formalisées sous la forme, par exemple, de formules logiques ou d'automates observateurs (Figure3.1). Le modèle simulable, les exigences et les interactions avec l'environnement (le contexte) constituent les données pertinentes et suffisantes pour pouvoir mener les vérifications. Pour pouvoir exécuter une exploration du modèle, il faut intégrer dans le modèle le comportement de l'environnement.

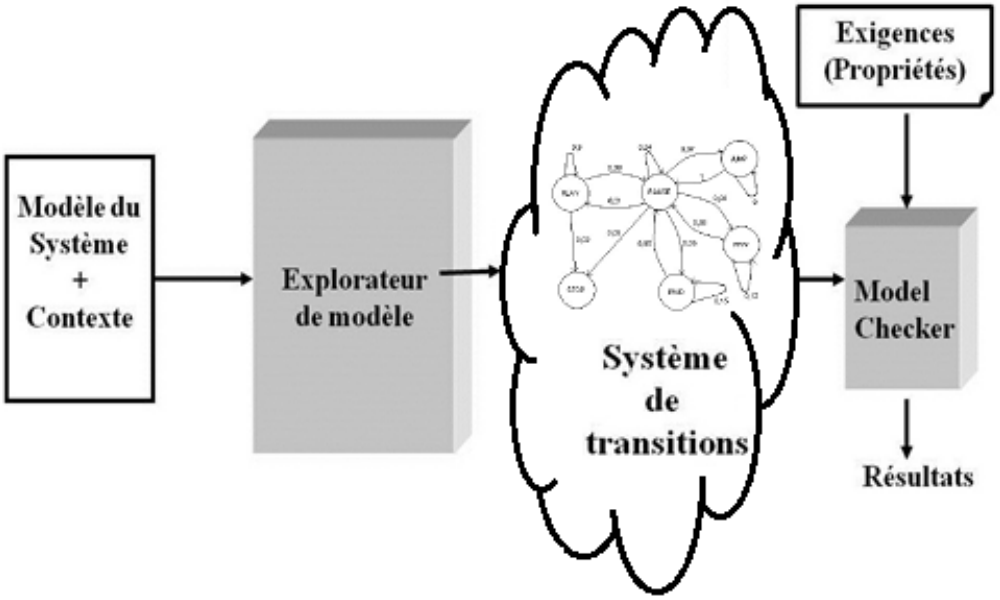

Figure 3.1 Vérification de propriétés par exploration de modèle.

Ce modèle est ensuite simulé et exploré par un outil de vérification. L'exploration génère un Système de Transitions (SdT). Celui-ci représente tous les comportements du modèle dans son environnement sous la forme d'un graphe de configurations et de transitions. Sur ce SdT, une vérification des propriétés peut être conduite, soit en appliquant des algorithmes de model-checking sur les formules logiques, soit en appliquant une analyse d'accessibilité des états d'erreur des observateurs. La difficulté

liée à cette technique est la production du SdT qui peut être de grande taille, dépassant la taille mémoire disponible (explosion combinatoire).

3.4 Exploration du modèle avec OBP

Pour explorer notre modèle E-UACML [9] (qui sera présenté dans la section suivante), nous utilisons l'outil OBP Explorer [112]. OBP est architecturé en 3 modules. Le front-end d'OBP importe les modèles Fiacre provenant d'une traduction des spécifications E-UACML. Il importe également les modèles CDL décrivant les propriétés et les scénarios de contextes. OBP Explorer [112] explore le modèle et, après chaque transition du modelé exécuté, laisse la main au dispositif d'observation (Observation Engine). Celui-ci capte les occurrences d'évènements et évalue, à chaque pas d'exécution du modèle, la valeur des prédicats et l'état de tous les observateurs impliqués. Une vérification des invariants et une analyse d'accessibilité des états d'erreurs des observateurs est donc ainsi menée. A partir des diagrammes d'un modèle CDL, OBP génère un ensemble des graphes acycliques de contexte. Chaque graphe est transformé en un automate Fiacre. Ceux-ci représentent l'ensemble des interactions possibles entre le modèle et l'environnement. Pour valider le modèle, il est nécessaire de composer chaque graphe avec le modèle. Chaque propriété référencée dans le modèle CDL doit être vérifiée sur le résultat de cette composition

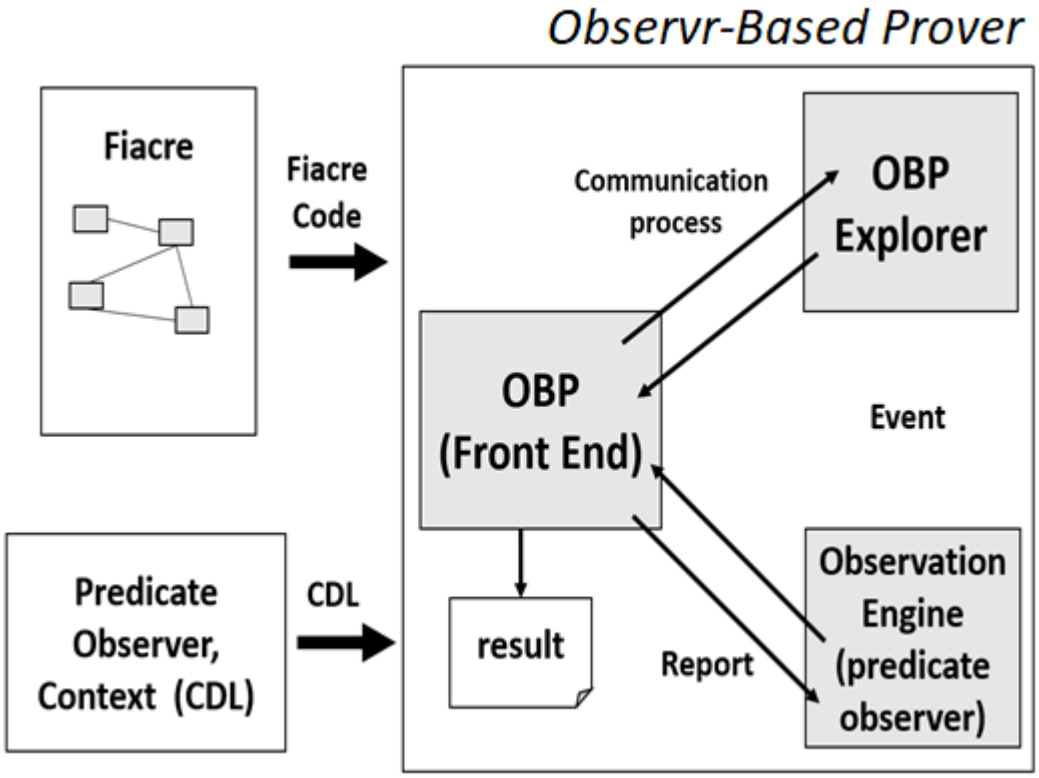

Figure 3.2 Architecture de l'outil OBP.

3.5 Le Langage CDL

Ce DSL (Domain Specific Language) est inspiré des diagrammes de cas d'utilisation basés sur des diagrammes d'activité et de séquence de [113]. D'une part, le modèle CDL équipé d'une syntaxe textuelle décrit l'utilisation des diagrammes d'activité et des diagrammes de séquence. Le comportement d'un environnement est considéré comme une séquence de scénarios qui décrivent les interactions entre le modèle simulé et les entités (acteurs) qui composent l'environnement de ce modèle. D'autre part, le modèle CDL décrit également les propriétés à valider. Inspiré des travaux de [113], le méta-modèle de CDL est défini et la sémantique est décrite en termes de traces [112]. Un programme CDL est constitué de plusieurs parties déclaratives :

- Déclarations des événements (event declaration)
- Déclarations des activités (activity declaration)
- Déclarations des prédicats (predicate declaration)
- Déclarations des propriétés (property declaration)
- Déclarations des contextes (cdl declaration)

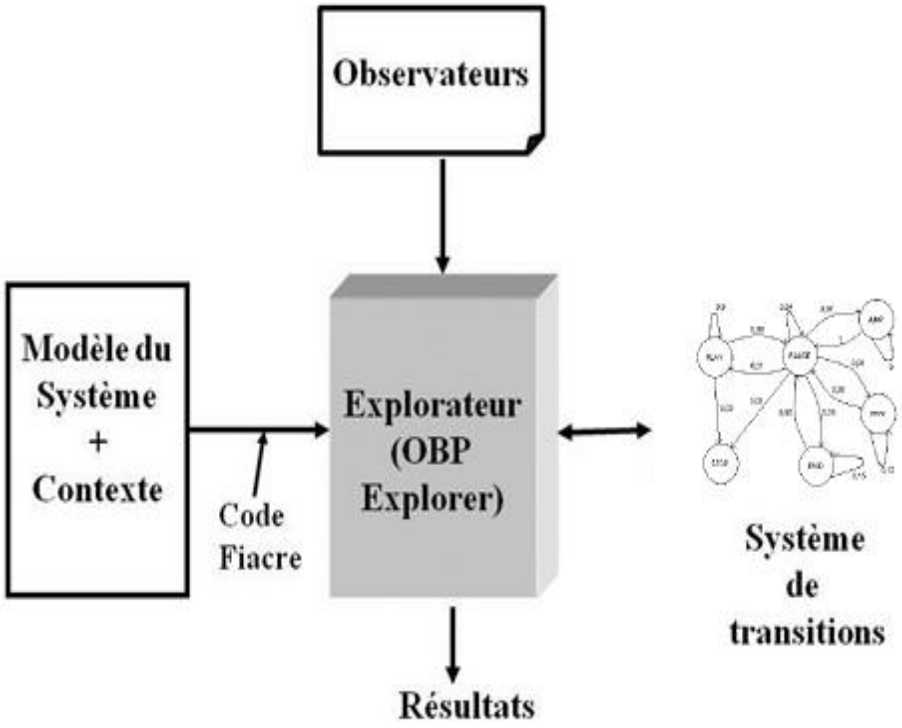

Figure 3.3 Outil de vérification expérimenté OBP Explorer.

Lors de la compilation du modèle CDL, le graphe correspondant à chaque acteur est déroulé (mise à plat de chaque boucle finie) puis entrelacé entre eux (selon la sémantique asynchrone des opérateurs parallèles dans des langages tels que LOTOS, FIACRE, etc…). L'entrelacement d'un ensemble de MSC (diagrammes de séquence de messages) [114] décrivant le comportement contextuel donne un graphe représentant toutes les exécutions des acteurs dans le contexte considéré. Le graphe est ensuite partitionné pour générer un ensemble de sous-graphes correspondant à des sous-contextes. Lors de l'exploration par le vérificateur, chaque sous-graphe est combiné avec le modèle à valider, et les propriétés sont vérifiées sur le résultat de cette combinaison.

3.6 Langage Fiacre

Fiacre [10] est un langage formel de spécification utilisé pour décrire les aspects comportementaux et temporisés des systèmes temps réels. Il intègre les notions de processus et de composants : (i) Les automates (processus) sont décrits par un ensemble d'états, une liste de transitions entre ces états avec des constructions classiques (affectations des variables, if-elsif else, while, compositions séquentielles), des constructions non déterministes et des communications par ports et par variables partagées. (ii) Les composants (component) décrivent les compositions de processus. Un système est construit comme une composition parallèle de composants et/ou processus qui peuvent communiquer entre eux par des ports ou variables partagées. Les priorités et les contraintes de temps sont associées aux opérations de communication.

Les processus Fiacre peuvent se synchroniser de manière synchrone avec ou sans passage de valeur via les ports. Ils peuvent également s'échanger des données via des files de communications asynchrones implantées par des variables partagées. Ce sont ces modes de communication que nous utilisons dans notre approche pour obtenir AC_Fiacre [9]. Les instances des processus Fiacre communiquent soit par des variables partagées, qui permettent alors de modéliser des registres partagés, soit par des files de messages pour une communication asynchrone en mode FIFO (First In, First Out). Pour les communications synchrones, les ports sont utilisés.

L'exemple de modèle Fiacre ci-dessus comprend un composant « Elevator » qui regroupe les instances des processus qui s'exécutent en parallèle (opérateur ||) :«LiftPanel», «LiftControl» et «HoistControl». Ces instances communiquent par les variables partagées (LiftPanel_1,events_To_itsLiftControl, events_To_itsHoistControl) [115].

Listing 1. Un exemple simple du programme Fiacre

```
component Elevator
is
var
LiftPanel_1 : FIFO_Events_LiftPanel,
events_To_itsLiftControl : FIFO_Events_LiftControl,
events_To_itsHoistControl : FIFO_Events_HoistControl
init
events_To_itsLiftControl := {||};
events_To_itsHoistControl := {||}
par
LiftPanel (&LiftPanel_1, &events_To_itsLiftControl)
        || LiftControl (&events_To_itsLiftControl, &events_To_itsHoistControl)
        || HoistControl (&events_To_itsHoistControl)
End par
Elevator
```

Le modèle E-UACML sera traduits en une spécification formelle exprimée dans ce langage (Fiacre). Ce dernier sera utilisé pour vérifier formellement les propriétés avec l'outil OBP Explorer.

3.7 Le processus de vérification

Tout d'abord, le modèle de conception sur lequel s'applique la vérification des exigences doit pouvoir être simulable et traduisible dans un format accepté par le vérificateur. Ensuite, les exigences sont disponibles et peuvent être formalisées sous la forme de propriétés logiques. Enfin, les interactions entre l'environnement et le modèle doivent être décrites précisément. Le modèle simulable, les exigences et les interactions avec l'environnement constituent les données pertinentes et suffisantes pour aborder l'approche que nous proposons. Celle-ci inclut donc les phases suivantes (Figure 3.4):

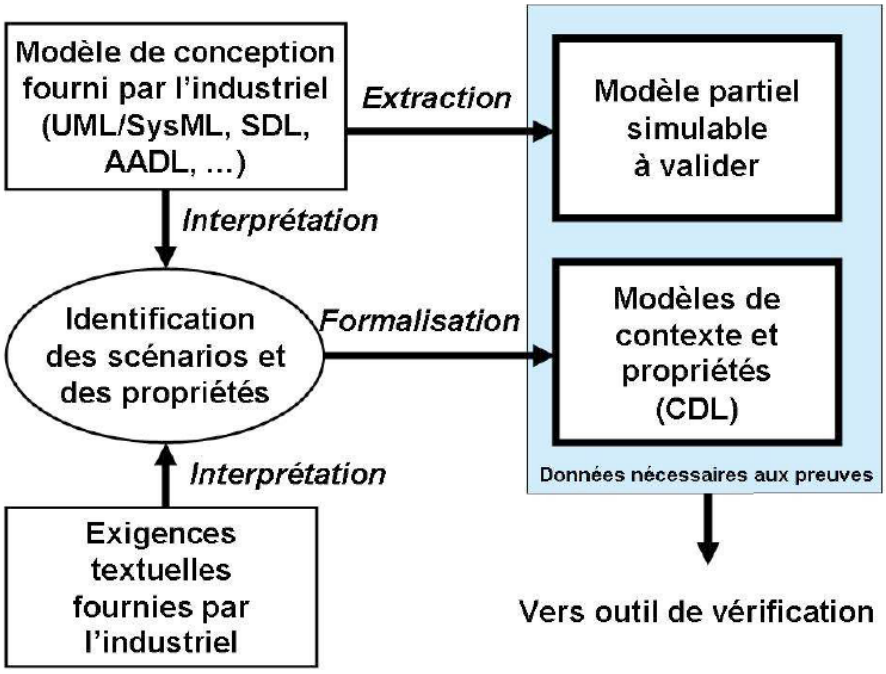

Figure 3.4 Processus de prise en compte des données d'exigences et du modèle à valider.

- A partir d'un modèle de conception, un modèle formel est généré (manuellement ou semi-automatiquement) par une extraction des données utiles du premier modèle et une traduction adéquate. Ces données permettent d'obtenir un modèle comportemental formel simulable par l'explorateur du vérificateur. Dans notre cas nous supposons que le modèle formel généré est sémantiquement conforme à la partie utile du modèle fourni par l'industriel. Cette conformité doit être assurée par la traduction. Nous ne décrivons pas ces transformations de modèle, mais des informations à ce sujet peuvent être trouvées dans des projets tels que TopCased4 ou Oméga5. Dans nos expérimentations, nous avons construit manuellement les modèles au format du vérificateur utilisé.

- A partir d'une interprétation d'exigences textuelles (en langage naturel) et d'autres éléments de spécification disponible fournie dans des documents d'exigences, les propriétés et les scénarios CDL sont formalisés. La rédaction des propriétés formelles et la modélisation du contexte sont d'autant plus aisées que, d'une part, la description du comportement de l'environnement est déjà formalisée (cas d'utilisation, scénarios, diagrammes de séquences, etc.) et que, d'autre part, les exigences sont exprimées sans ambiguïté.

A ce niveau, il nous semble important que les exigences et le contexte soient bien dissociés des éléments du modèle de conception et formalisés de façon non ambiguë. Ils doivent également porter sur des éléments bien identifiés du modèle de conception.

Une condition préalable pour la mise en œuvre de ce processus est de disposer, de la part de l'industriel, de spécifications qui permettent d'identifier les contextes et les exigences qui peuvent être soumises à la vérification et qui peuvent être reliées à un contexte. Le processus de développement doit inclure une étape de spécification de l'environnement, permettant d'identifier un ensemble complet de toutes les interactions entre l'environnement et le modèle, ce qui doit assurer un taux de couverture de 100%. L'atteinte de cette couverture correspond à l'hypothèse formulée précédemment, qui postule que le concepteur est capable d'identifier toutes les interactions possibles entre le système et son environnement. L'ensemble des interactions doit lui être fourni formellement comme un résultat du processus d'analyse de l'architecture logicielle conçue. Compte tenu de la complexité de l'ensemble des interactions, il est préférable que l'utilisateur construise un ensemble structuré de modèles CDL spécifiques, chacun correspondant à des cas d'utilisation spécifiques.

En ce qui concerne la formalisation des propriétés, les exigences comportent souvent beaucoup d'informations sur le système et son environnement. Pour simplifier leur formalisation avec les patrons, elles doivent être décomposées en exigences élémentaires. Par exemple, dans l'étude de cas décrits précédemment, l'exigence R (listing 1) peut être décomposée en trois sous-exigences comme suit :

- R1: "Au cours de la procédure d'initialisation, SM doit associer un identificateur à chaque périphérique (HMI) du système ayant demandé à se loguer, avant un délai MaxD_log."

- R2: "Chaque périphérique logué peut demander une opération et reçoit en retour un rôle avant un délai MaxD_oper."

- R3: "L'initialisation s'achève avec succès, lorsque SM a affecté tous les identificateurs et les rôles aux périphériques."

Une fois ce travail de décomposition des exigences réalisé, la formalisation avec les patrons CDL est facilitée. Mais malgré cette décomposition, certaines exigences peuvent nécessiter un dialogue avec la personne qui a spécifié l'exigence initiale. Par exemple, l'exigence R3 mérite une demande de clarification concernant la signification de la mention "avec succès". Les méthodologies étudiées dans le domaine de l'ingénierie des exigences par exemple dans [116][117][118] sont une source

d'inspiration pour compléter ce travail. Compte tenu de l'ampleur du sujet, nous choisissons de ne pas aborder ces aspects dans notre étude.

3.8 Discussion et Conclusion

Un model-cheking est vu comme un testeur automatique, qui a la capacité de vérifier tous les comportements possibles du système pour des chemins d'exécution bien fixés. Cette technique de vérification a fait une percée remarquable dans les champs d'application industrielles. Citons quelques produits comme : Bull, IBM, Microsoft et Siemens. On peut par exemple citer le langage PSL (Property Specification Language) [119] qui a été standardisé au niveau industriel dans le but de spécifier des propriétés temporelles (mis au point par un consortium entre Intel et IBM). Cette technique est également fréquemment utilisée dans le domaine de systèmes critiques, i.e; Bosch et Airbus dans une approche de Model Driven Developping (MDD).

Dans ce qui suit, nous exposons l'intégration de notre modèle semi-formel de contrôle d'accès (E-UACML) dans le processus de validation. En outre, nous discutons nos approches facilitants la spécification des transformations (mapping), dans le but de faciliter l'utilisation du model-checking pour la vérification formelle de contrôle d'accès.

Chapitre 4: Le méta-modèle E-UACML

SOMMAIRE

4 Le méta-modèle E-UACML

4.1 Introduction

L'architecture dirigée par les modèles (MDA) est une approche bien connue pour soutenir le processus de développement logiciel [4]. Model Driven Security (MDS) est une méthodologie, basée sur l'approche MDA, pour développer des systèmes sécurisés [78]. MDS permet aux concepteurs de spécifier des modèles de système ainsi que leurs exigences de sécurité et d'utiliser des outils pour générer automatiquement des architectures de système à partir de modèles de système. L'avantage important de l'utilisation de l'approche MDS est le développement de modèles de système sécurisés qui sont à la fois réutilisables et évolutifs.

Les exigences de contrôle d'accès des organisations sont devenues de plus en plus complexes ces dernières années, rendant les mécanismes de contrôle d'accès basés sur un modèle de contrôle d'accès unique (traditionnel) soit inefficaces soit inappropriés. Par conséquent, les systèmes de contrôle d'accès modernes exigent généralement que les exigences de contrôle d'accès soient spécifiées (et appliquées) en utilisant une combinaison de modèles de contrôle d'accès. Nous appelons politiques de contrôle d'accès « hybrides » les politiques basées sur une combinaison de modèles de contrôle d'accès.

L'objectif de ce chapitre est de développer un langage de modélisation visuel basé sur UML qui prend en charge un large éventail de modèles de contrôle d'accès de manière « générique » tout en restant aussi simple que possible. L'idée clé de notre travail est d'utiliser un « méta-modèle de contrôle d'accès » pour le langage de modélisation que nous proposons.

L'adoption d'un méta-modèle de contrôle d'accès comme base de notre langage de modélisation offre au moins deux avantages : (i) il simplifie la syntaxe de notre langage de modélisation et (ii) il fournit aux auteurs de politiques un cadre général pour représenter une variété d'exigences de contrôle d'accès. Par conséquent, le langage de modélisation que nous développons en utilisant un méta-modèle de contrôle d'accès sera un langage simple, mais puissant, pour modéliser plusieurs types différents de politiques de contrôle d'accès ; à la fois seul et combinés.

Nous nous intéressons particulièrement à la méthode semi-formelle UACML (Unified Access Control Modeling Language) [7]. Cette méthode permet grâce à des concepts assez généraux de produire des modèles de contrôle d'accès hybrides basés sur UML. Les besoins en sécurité sont diversifiés aujourd'hui, en effet, on rencontre de plus en plus de situations dans lesquelles, il ne suffit plus d'un modèle de contrôle d'accès pour

répondre aux exigences de sécurité d'un système. Dans ce genre de situation, Il est nécessaire de combiner deux, voire plusieurs concepts (rôles, groupes, niveaux de sécurité, etc.) pour renforcer le contrôle d'accès. Par exemple, avec l'utilisation du Cloud Computing, on peut avoir besoin de combiner les trois modèles de contrôle d'accès de base DAC, MAC et RBAC (contrôle d'accès hybrides).

Aujourd'hui l'importance d'UML dans la modélisation des systèmes n'est plus à démontrer. UACML constitue donc pour nous, un élément essentiel pour la modélisation des systèmes de contrôle d'accès. Nous avons également constaté qu'une modélisation avec UACML uniquement n'est pas suffisante pour garantir que le futur système à implémenter soit sûr par rapport aux propriétés de sécurité qu'il doit satisfaire. Pour cela, nous étendons les diagrammes UACML par le model E-RBAC et des contraintes spatio-temporelles en nous inspirant du modèle STRBAC [120] définis à la section 1.4.3.2 chapitre 1. A partir de la sémantique d'UACML, E-RBAC et du modèle STRBAC, nous avons proposé le méta-modèle E-UACML [9].

Dans le contrôle d'accès basé sur les rôles (RBAC), les utilisateurs ont accès à des rôles et autorisations prédéterminés. Ainsi, les résultats attendus ne sont pas atteints dans les situations d'urgence par le biais de la politique RBAC. Dans les situations d'urgence, les utilisateurs doivent parfois accéder à des ressources non autorisées dans des situations normales. Par exemple, un jeune médecin à l'hôpital peut effectuer les tâches d'un médecin principal dans une situation d'urgence. Parmi les principales contributions de cette thèse, est la proposition d'une extension du modèle UACML par le model E-RBAC [8] qui permettra de contrôler le système dans les situations d'urgence en tenant compte des contraintes du modèle. Dans le cadre de notre étude, nous proposons le profil E-UACML (Emergency Unified Access Control Modeling Language) qui étend les diagrammes d'activité UACML pour représenter la politique RBAC d'urgence (E-RABC).

Plutôt que de développer un autre méta-modèle de contrôle d'accès, nous étendons le méta-modèle de contrôle d'accès UACML pour prendre en charge le modèle E-RBAC et les contraintes spatio-temporelles. Essentiellement, notre travail propose un langage de modélisation "générique" pour le contrôle d'accès à travers un méta-modèle qui peut être instancié pour dériver des modèles, qui représenteront spécifiquement des modèles de contrôle d'accès bien connus, tels que MAC et RBAC, ainsi que des combinaisons de modèles de contrôle d'accès, pour prendre en charge les modèles de contrôle d'accès hybrides.

4.2 Motivation: Exemple Pol_1

Pour faciliter la compréhension de notre méta-modèle E-UACML, nous commençons par donner un exemple simple de politique de sécurité hybride « **Pol_1** » décrite comme suit : dans un service hospitalier, les accès des membres sont gérés en fonction de leurs rôles. Nous définissons quatre rôles :
Médecin (Doctor) « **D** », Médecin stagiaire (TraineeDoctor) « **TD** », Infirmier (Nurse) « Nur » et le Directeur (Director) « **Dir** ». Les membres du service hospitalier sont divisés en deux groupes qui sont : le groupe de personnel soignant (Caregiving staff) « **CS** » et le groupe de personnel administratif (Administrative staff) « **AS** ».

Lorsqu'un nouveau membre intègre le service hospitalier qui n'a pas de rôle spécifique à lui comme un étudiant de médecine, il est directement affecté à un groupe et obtient ses accès conformément à ce groupe.

La table 4.1 indique les autorisations associées pour chaque rôle. Les rôles et les groupes de chaque sujet sont indiqués dans la table 4.2 et 4.3. Dans la table 4.4, la première colonne affiche le type de situation d'urgence, la deuxième colonne indique le rôle d'urgence, la troisième indique les autorisations associées, la quatrième et sixième indiquent respectivement le temps et la localisation qui doivent être respectés par ce rôle d'urgence.

Supposons qu'un patient ait besoin d'un traitement d'urgence et qu'aucun médecin ne soit disponible à l'hôpital. Le sujet **S1** informe cette situation et définit l'état du système en cas d'urgence. A partir de ce moment, toutes les opérations seront enregistrées. Le sujet **S1** demande le rôle d'urgence (**ER1**) conformément à la table 4.4. **S1** doit s'acquitter d'obligations en ne dépassant pas les 30 minutes et il doit être dans la salle des urgences.

Rôle	*Permission*
Doc	Read, medical_file (R,MF) Write, medical_file (W,MF) Read, prescription (R,Prescription) Write, prescription (W,Prescription) Read, nurse_report (R, NR)
TD	Read, medical_file (R,MF) Read, prscription (R,Prescription) Read, nurse_report (R,NR)
Nur	Read, nurse_report (R,NR) Write, nurse_report (W,NR)
Dir	Read, staff_table (R,ST) Write, staff_table (W,ST)

Table 4.1 Permission-role assignment

Sujet	*Rôle*
S1	TD
S2	TD
S3	Nur
S4	Doc
S5	Dir

Table 4.2 Subject-role assignment

Sujet	*Groupe*
S1	CS
S2	CS
S3	CS
S4	CS
S5	AS

Table 4.3 Subject-group assignment

Situation d'urgence	*E_role*	*E_permission*	*Temps*	*Locations*
Emergency medical	ER1	Read, medical_file Read, nurse_report Read, prescription Write, prescription	30 minutes	Emergency_room

Table 4.4 Permission-emergency role assignment

Les permissions des employés de ce service hospitalier sont déterminées par les règles de contrôle d'accès suivantes :

- R1 : Un individu ayant le rôle Médecin stagiaire «**TD**» ne peut lire que les fichiers (medical_file, prescription et nurse_report) s'il en obtient l'autorisation.
- R2 : Cet individu peut obtenir le rôle d'urgence ER1 pendant 30 minutes s'il est dans la Emergency room.
- R3 : Seul un médecin du centre hospitalier a le droit de modifier les fichiers (medical_file et prescription)
- R4 : Seul le directeur du centre hospitalier a le droit de lire et de modifier la staff_table

Dans le reste de notre document, R1 indique que dans le système à modéliser nous devons identifier toutes les personnes qui ont le rôle TD et nous assurer qu'elles ont bien l'autorisation de lire les fichiers (medical_file, la prescription et nurse_report) avant de leur permettre d'exécuter l'action « Read » qui conduit à accéder effectivement à ces fichiers. R2 indique que le rôle ER1 est assigné à une personne pendant 30 minutes seulement si la personne est présente à la Emergency room.

4.3 Les concepts du méta-modèle E-UACML

La syntaxe de notre langage de modélisation est basée sur une extension du méta-modèle de contrôle d'accès « UACML » [7]. La figure 4.1 illustre notre modèle. Essentiellement, la sémantique de notre langage signifie que chaque bord d'association représenté entre les classes définit une relation sur les classes.

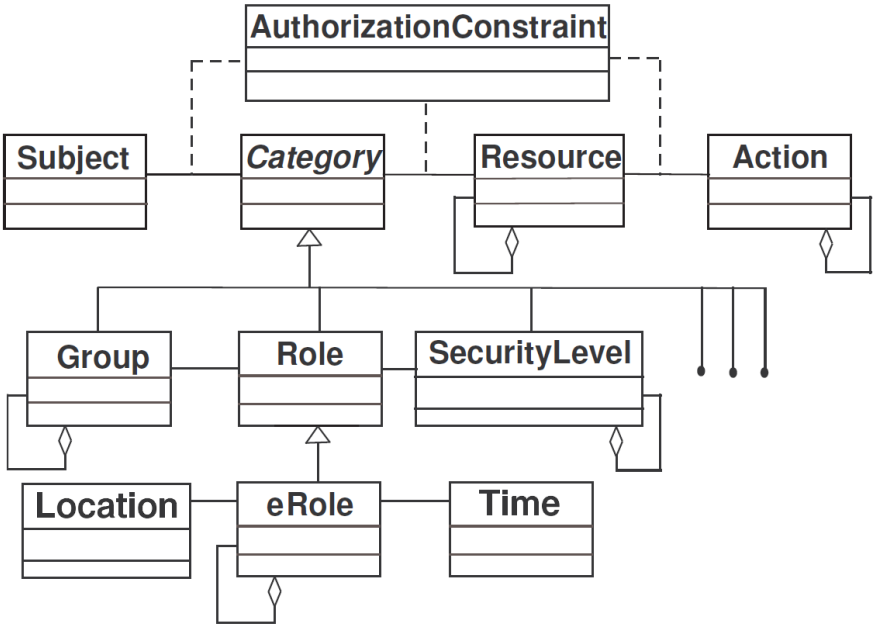

Figure 4.1 Une représentation de notre méta-modèle E-UACML [9].

Le composant central de notre langage de modélisation est la notion de catégorie qui est instrumentalisée pour abstraire les composants clés de divers modèles de contrôle d'accès, tels que les groupes, les niveaux de sécurité et les rôles. Essentiellement, les catégories fournissent des moyens d'associer des sujets et des autorisations, qui sont représentés par des paires <ressource, action>. Notons que la classe Category est définie comme une classe abstraite (représentée par une police en italique). Par conséquent, la classe Catégorie doit être spécialisée pour créer des catégories spécifiques, tels que des groupes, des rôles et des niveaux de sécurité, auxquels des sujets ou des ressources peuvent être affectés.

Le côté gauche de la classe Category figure 4.1 affiche une classe Subject qui représente les entités au sein du système qui sont capables d'initier des demandes d'accès. La classe Subject a une association avec la classe Category.

Le côté droit de la classe Catégorie affiche deux classes : Ressource et Action. La classe Ressource représente les ressources protégées au sein du système et est associée à la

classe Category. Une telle affectation est utile pour représenter quelles ressources sont accessibles par une catégorie donnée.

La classe Action représente les opérations qui peuvent être effectuées sur des ressources protégées. La classe Ressource a une association avec Class Action; ainsi, représentant quelles actions peuvent être effectuées sur des ressources données.

Il est parfois souhaitable, pour certaines applications, de combiner différents modèles de contrôle d'accès au sein d'une même application pour prendre en charge des politiques de contrôle d'accès hybrides. Notre langage de modélisation prend en charge les politiques de contrôle d'accès hybrides en permettant aux catégories d'être associées à d'autres catégories (représentées, dans la Figure 4.1, comme un bord d'auto-association sur la catégorie Category). De plus, notre langage permet également de spécifier les relations hiérarchiques entre les catégories en agrégeant les catégories appropriées (représentées, dans la figure 4.1, comme un bord d'auto-association).

Notre modèle prend également en charge la création d'une hiérarchie de ressources et d'une hiérarchie d'actions en agrégeant respectivement les ressources et les actions. De telles relations hiérarchiques sont représentées par une association d'agrégation (représentée sur le bord d'association de la figure 4.1) sur les classes Objet et Action. Enfin, la classe AuthorizationConstraint est utilisée pour représenter des contraintes qui spécifient des restrictions sur les affectations de catégorie de sujet, les affectations de catégorie-ressource et les affectations de ressource-action.

4.3.1 Génération de modèles

Dans cette section, nous décrivons la modélisation des politiques de contrôle d'accès en instanciant les modèles décrits dans la section précédente. Considérons, L’exemple « Pol_1 » qui illustre un état d'autorisation présenté dans la section 4.2.

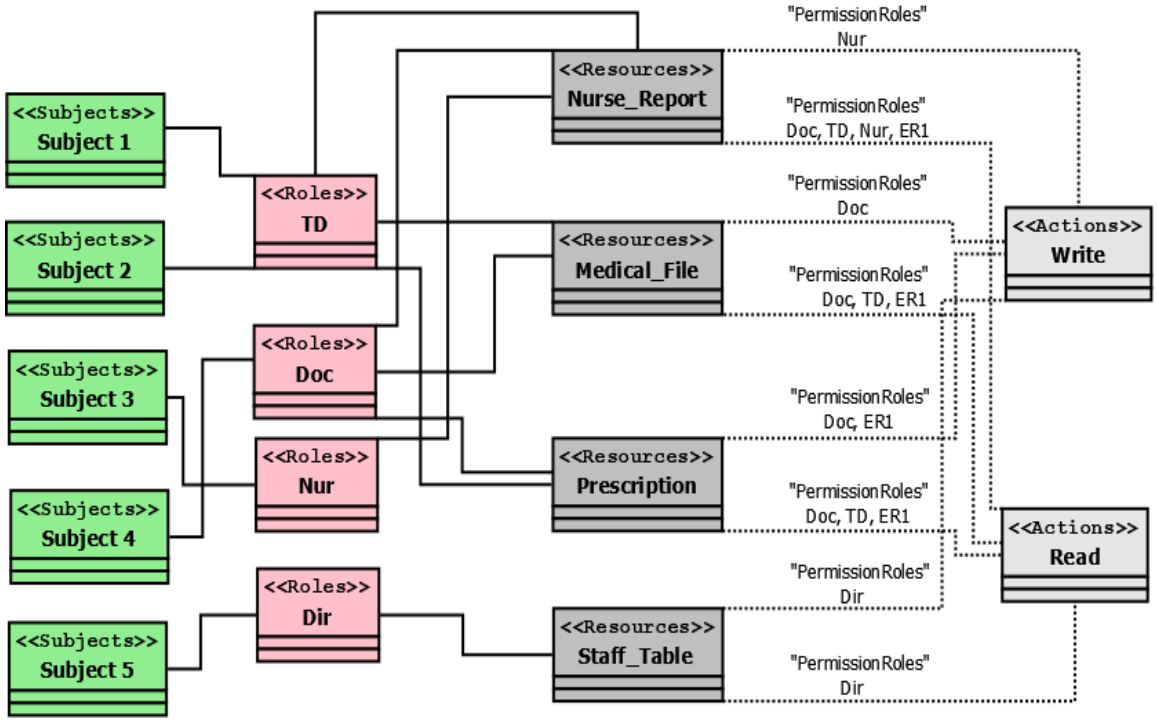

Figure 4.2 Modèle basé sur RBAC.

La figure 4.2 illustre un modèle concret qui respecte le modèle RBAC (illustré à la figure 1.7 section 1.4.3 chapitre 1) pour les autorisations données dans la table 4.1 et 4.2. Plus précisément, la moitié gauche de la Figure 4.2 montre quatre rôles Médecin (Doctor) «Doc», Médecin stagiaire (TraineeDoctor) «TD», Infirmier (Nurse) «Nur» et le Directeur (Director) «Dir» et leurs utilisateurs assignés, tandis que la moitié droite montre l'assignation des ressources aux rôles et actions qui peuvent être effectuées sur les ressources. Notre langage désigne l'association d'actions à des ressources par un stéréotype « PermissionRole », qui spécifie l'ensemble de rôles auxquels s'applique l'association « Resource-action ». Par exemple, dans la figure 4.2, l'association entre la ressource « Nurse_Report » et l'action « Write » est spécifiée pour le rôle Infirmier (Nurse) «Nur», tandis que l'association de l'action « Read » à la ressource « Nurse_Report » est spécifiée pour le rôle Médecin stagiaire (TraineeDoctor) «TD».

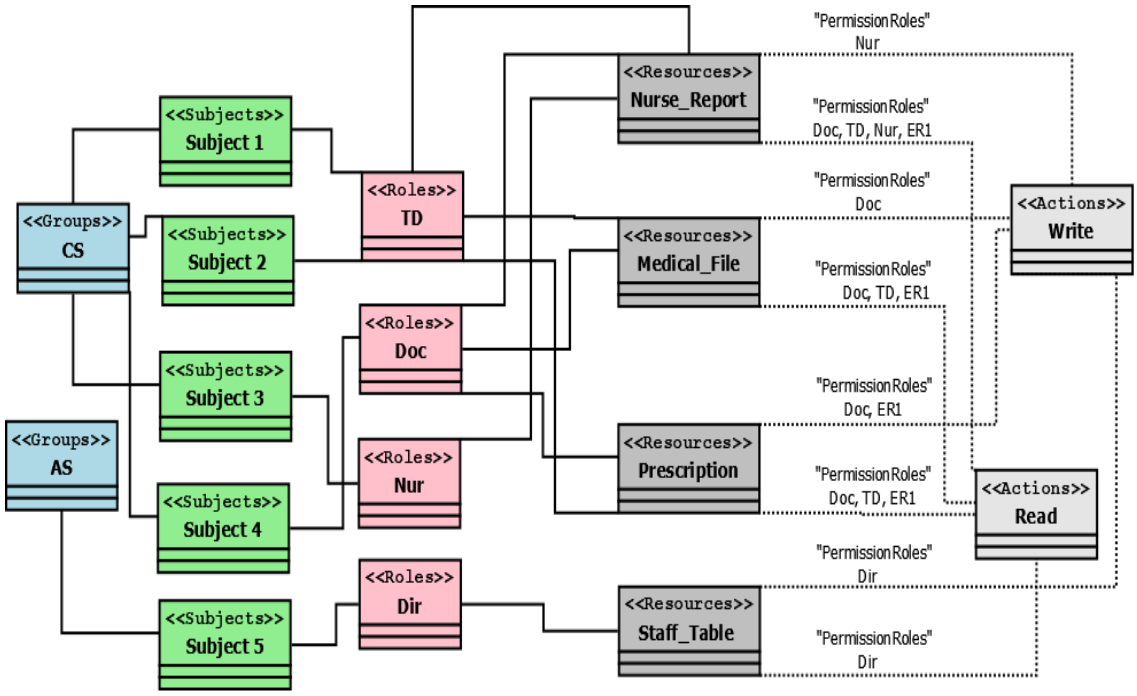

Figure 4.3 Un modèle basé sur les groupes d'utilisateurs et le RBAC.

La figure 4.3 étend la figure 4.2 en prenant en charge les groupes d'utilisateurs et illustre le modèle concret pour les autorisations données dans la table 4.3. Le modèle concret de la figure 4.3 respecte le méta-modèle de contrôle d'accès hybride illustré dans la figure 2.10 section 2.2.5 chapitre 2. Les affectations de groupes d'utilisateurs sont affichées en définissant une association entre un utilisateur et les groupes auxquels l'utilisateur appartient. Par exemple, les utilisateurs « Subject 1 » et « Subject 2 », qui sont affectés au rôle Médecin stagiaire (TraineeDoctor) « TD », appartiennent au groupe de personnel soignant (Caregiving staff) « CS ».

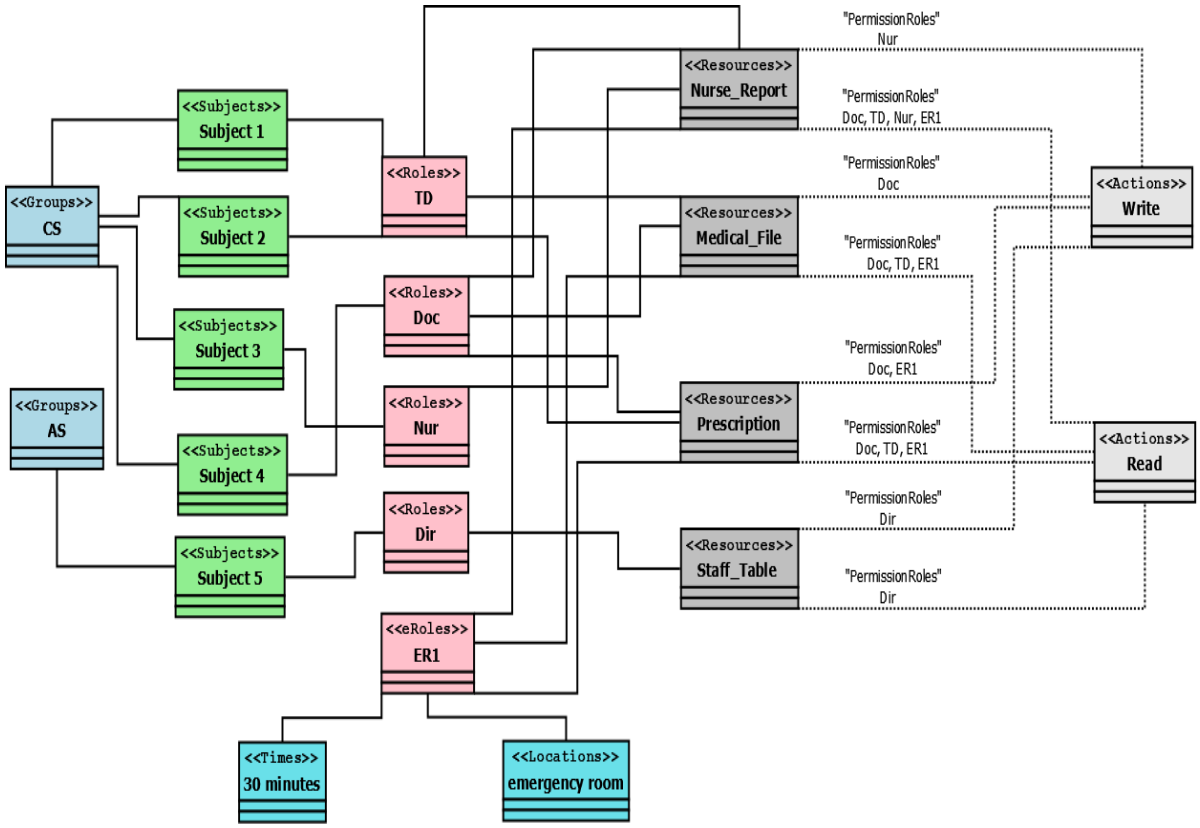

Figure 4.4 Modélisation E-UACML de Pol_1 (politique hybride).

La figure 4.4 étend la figure 4.3 en définissant les rôles d'urgence et les contraintes spatio-temporelles pour la prise en charge des politiques E-RBAC et STBAC. Par exemple, le rôle d'urgence médical (Emergency medical) « ER1 » est attribué à une personne pendant 30 minutes uniquement si la personne est présente à l'urgence.

4.4 Comparaison des profiles UML de sécurité

Plusieurs recherches ont abouti au développement de langages de modélisation basés sur UML qui intègrent les exigences de sécurité dans les modèles de conception de systèmes
[121] [122] [123] [124] [125].

Epstein et Sandhu ont proposé un cadre qui utilise les notations UML pour utiliser UML comme langage pour représenter les exigences RBAC [121]. Shin et Ahn ont proposé une technique alternative pour utiliser la notation UML pour décrire la modélisation RBAC [122]. Le travail de Doan et al. a fourni un support pour l'incorporation de MAC dans les diagrammes UML [126].

Jurjens a proposé une approche, appelée UMLsec [6], pour développer des systèmes sécurisés utilisant une extension d'UML. UMLsec offre un support pour l'annotation des modèles UML avec des exigences de sécurité formellement spécifiées, comme la confidentialité ou le flux d'informations sécurisé.

Les travaux fondateurs de Basin et al. ont démontré l'application de MDS au domaine du contrôle d'accès [5]. Ce travail comprend un langage de modélisation de la sécurité, appelé SecureUML [5], qui fournit un support pour le développement de modèles de conception de système qui incluent les exigences de contrôle d'accès. SecureUML est développé sur une extension de RBAC [3]. Nous pensons que cette fonctionnalité de SecureUML limite son applicabilité lorsqu'un ensemble diversifié d'exigences de contrôle d'accès doit être pris en charge. Par exemple, SecureUML ne peut pas exprimer à la fois des politiques de contrôle d'accès non-RBAC et hybrides. En comparaison, notre langage de modélisation est développé en utilisant le concept de catégorie, qui peut être spécialisé pour créer divers types de contrôle d'accès, tels que des rôles, des niveaux de sécurité et des groupes. Par conséquent, nous pensons que notre langage de modélisation est plus générique que SecureUML; et considèrent SecureUML comme un cas particulier de notre langage de modélisation.

Le travail de Pavlich-Mariscal et al [124] propose un langage de modélisation qui prend en charge une variété de modèle de contrôle d'accès, telles que DAC, MAC et RBAC. Leur langage de modélisation prend également en charge les politiques de contrôle d'accès hybrides, qui combinent différents modèles de contrôle d'accès pour spécifier les exigences des politiques. Cependant, leur langage de modélisation ne peut pas prendre en charge des politiques autres que DAC, MAC et RBAC. En revanche, notre langage de modélisation (E-UACML) est basé sur un concept générique, qui peut être spécialisé (selon les besoins) pour créer des types de contrôle d'accès spécifiques. Cette fonctionnalité fournit une approche plus générique pour supporter différents modèles de contrôle d'accès.

En comparant des modèles de sécurité définis dans la Table 5.1, nous distinguons une similitude des approches UMLsec et SecureUML. Nous remarquons aussi que presque tous les profils ne traitent qu’un seul modèle de sécurité (RBAC). Cependant le profil E-UACML est suffisamment générique pour modéliser un nombre important de modèles de contrôle d’accès.

Concept	*Modèle de sécurité*	*Mapping UML*	*Lisibilité*	*Objectifs*	*Contraintes*
Secure UML	RBAC	Class State/transition	Lisible	Objectif commercial Objective	OCL
UML sec	RBAC	Use case Classe état/ transition	Moins lisible	Objectif commercial	N'est pas identifié
Misus Cases	-	Use case	Moins lisible	Objective threats	-
Abus Cases	-	Use case	Moins lisible	Objectif commercial	-
Security Use Case	-	Use case	Moins lisible	Objectif commercial	-
UACML	DAC MAC RBAC	Class State/transition	Lisible	Objectif commercial	OCL
BAAC@UML	RBAC	Class State/transition	Lisible	Objectif commercial	-
Auth UML	RBAC	Use case	Moins lisible	Objectif commercial	OCL
MoDelO PIM	ORBAC	Class State/transition	Lisible	Objectif commercial	OCL
E-UACML	**Tous les modèles + E-RBAC**	**Class State/transition**	**Lisible**	**Objectif commercial**	**OCL**

Table 5.1. Comparaison des profiles UML de sécurité

Dans [109] les auteurs proposent une méthode pour vérifier les exigences de sécurité dont des caractéristiques sont basées sur le langage de modélisation unifié (UML) en utilisant la technique du model-checking et le critère commun de notion de sécurité. Les critères communs aident à la définition des exigences de sécurité adéquates sous la forme d'une table. Cela permet aux développeurs de vérifier si les modèles d'analyse des exigences UML répondent à ces exigences dans les premiers stades de développement de logiciels. Le modèle d'UML est transformé en un automate fini dans l'outil de vérification UPPAAL [97]. Dans le document [127] l'auteur propose d'élaborer un cadre formel pour la spécification et la validation de systèmes de contrôle d'accès hybrides. En particulier, il propose d'élaborer une méthodologie pour spécifier des modèles de contrôle d'accès, par raffinement successif, à l'aide d'une notation proche d'UML. Les spécifications sont ensuite automatiquement traduites en notation formelle

B-événementiel afin de prouver formellement des propriétés de modèles. L'article [79] présente une approche qui combine les langages UML et B pour la spécification et la validation de politiques RBAC au niveau des activités d'un processus métier. Cette approche commence par la spécification semi-formelle des règles de contrôle d'accès à l'aide des diagrammes d'activité d'UML2. Les modèles d'activité sont ensuite traduits en une spécification formelle exprimée dans le langage B. La spécification B définit un cadre rigoureux favorisant la validation et la vérification de la politique RBAC. Le papier [128] propose, un travail portant sur une technique de vérification de propriétés par model-checking exploitant le langage CDL (Context Description Language) [88] et l'outil OBP (Observer-basedProver). La technique s'appuie sur une traduction des modèles MARTE [129] et des contraintes CCSL [130] en code Fiacre. CDL permet d'exprimer des prédicats et des observateurs. Ceux-ci sont vérifiés lors de l'exploration exhaustive du modèle complet par OBP.

4.5 Conclusion

Nous avons proposé un langage de modélisation basé sur UML pour incorporer les exigences de contrôle d'accès dans les modèles de conception de systèmes. Le langage de modélisation proposé est basé sur un méta-modèle de contrôle d'accès, plutôt que sur des modèles de contrôle d'accès spécifiques. Nous avons également étendu le méta-modèle de contrôle d'accès UACML pour prendre en charge le E-RBAC et les contraintes spatio-temporelles.

L'adoption d'un méta-modèle de contrôle d'accès comme base pour développer notre langage de modélisation a été utile pour développer un langage de modélisation simple, mais suffisamment puissant pour capturer différents modèles de contrôle d'accès (à la fois indépendamment et combinés). Plus précisément, notre langage de modélisation pourrait être utilisé comme cadre générique pour intégrer une variété d'exigences de contrôle d'accès dans les modèles de conception de systèmes. Nous avons démontré que E-UACML pouvait être « spécialisé » en divers modèles de contrôle d'accès spécifique.

Dans ce qui suit, nous exposons les procédures de vérification et de validation de notre méta-modèle E-UACML. Car une démarche IDM reste insuffisante pour appliquer une analyse formelle.

Chapitre 5: Approche pour la validation des systèmes de contrôle d'accès

SOMMAIRE

5 Approche pour la validation des systèmes de contrôle d'accès

5.1 Introduction

Les développeurs en général éprouvent des difficultés à spécifier correctement des exigences de contrôle d'accès adéquates pendant les phases initiales de développement. Pour modéliser ces exigences, les développeurs tendent vers Ingénierie Dirigée par les Modèles (IDM) [4] qui sont des systèmes modélisés à l'aide d'une notation semi-formelle et sont par la suite validés puis implantés. Plusieurs méthodes ont été proposées, comme securUML [5], UMLsec [6], UACML [7] (voir chapitre 2), etc... afin de faciliter la spécification et l'analyse de toutes les contraintes du contrôle d'accès.

Une fois que la politique de contrôle d'accès est modélisée, la difficulté se situe dans l'expression des propriétés et leur vérification formelle. E-UACML [9] à lui seul ne peut garantir que l'implémentation des modèles créés à partir de notations UML puisse satisfaire aux exigences de sécurité d'un système. Il est alors nécessaire d'introduire de nouvelles techniques pour faire en sorte que les implémentations des modèles E-UACML soient correctes par rapport à des propriétés de sécurité. Notons qu'une démarche IDM reste insuffisante dans le sens où elle n'indique pas comment utiliser les modèles pour appliquer une analyse formelle, pour cela, des techniques formelles de transformation doivent être introduites.

Nous allons par conséquent nous intéresser dans ce chapitre, aux procédures de modélisation et de vérification des systèmes de contrôle d'accès. Dans ce cadre de la modélisation, de nombreux problèmes de sécurité sont causés par une mauvaise architecture de contrôle d'accès ou une mauvaise utilisation des installations (modèles) de contrôle d'accès par le concepteur du système informatique. L'utilisation des méthodes formelles est une réponse à ces préoccupations.

Dans ce travail, nous introduisons une méthodologie (architecture globale) et une chaine de transformation (tool-chain) pour formaliser les propriétés de sécurité [11] [9]. Les modèles E-UACML [9], Fiacre [10] , OBP/CDL2 sont utilisés pour exprimer et vérifier des contraintes de contrôles d'accès. Nous utilisons dans cette architecture notre modèle E-UACML qui est un méta-modèle de contrôle d'accès qui offre au moins deux avantages : (i) simplifier la syntaxe de la modélisation et (ii) fournir aux auteurs des politiques un cadre général pour représenter une variété de modèles pour modéliser plusieurs types combinés de politiques de contrôle d'accès. Comme E-UACML est un langage semi-formel, une traduction vers une méthode formelle est nécessaire. Nous commençons par la spécification semi-formelle des règles de contrôle d'accès à l'aide des diagrammes d'activité E-UACML accompagnés de contraintes de contrôle d'accès et des contraintes spatio-temporelles. Les modèles d'activité sont ensuite traduits en une

spécification formelle [9] exprimée dans le langage Fiacre [10]. Ce dernier est utilisé pour vérifier formellement les propriétés avec l'outil OBP Explorer basé sur le model-checking.

5.2 Principes de l'approche

Nous présentons dans cette section le principe général de notre approche [9] ainsi que la méthodologie proposée. Notre objectif principal est de modéliser et formaliser les politiques de contrôle d'accès. Dans la section 2.2 chapitre 2, nous avons présenté quelques techniques pour modéliser les systèmes qui implémentent ces modèles. Nous avons constaté, qu'UACML est la technique de modélisation qui répond le mieux aux besoins de la construction des systèmes de contrôle d'accès. Elle s'inspire de la modélisation avec des diagrammes de classes UML, pour fournir des représentations graphiques des systèmes. Dans les situations d'urgence, les utilisateurs doivent parfois accéder à des ressources non autorisées dans des situations normales. Pour augmenter la flexibilité du contrôle d'accès hybride une extension du modèle UACML est proposée en ajoutant le RBAC d'urgence (E-RBAC). A partir de la sémantique d'UACML, E-RBAC et du modèle STRBAC, nous avons proposé le modèle E-UACML [9] (chapitre 4). A partir de E-UACML des transformations vers Fiacre seront établies.

Afin de formaliser des politiques de contrôle d'accès, il faut s'assurer que la sémantique du système soit préservée et que toutes les propriétés qui sont vérifiées dans le monde E-UACML soient aussi préservées dans le monde Fiacre. Ensuite, le programme cible Fiacre sera utilisé pour vérifier formellement (model-checking) les propriétés avec l'outil OBP Explorer. Pour cela, les propriétés du système et les contraintes de contrôles d'accès encodées par des prédicats et des automates observateurs sont décrits avec le langage CDL (Context Description language) [88].

Enfin, la spécification Fiacre générée et les propriétés CDL sont données comme entrées à l'outil OBP, qui vérifie les spécifications Fiacre par rapport aux propriétés CDL. Nous utilisons l'outil OBP pour effectuer nos expériences et vérifier qu'une politique (initialement décrite dans E-UACML) contenant des contraintes de sécurité garantit les exigences attendues (exprimées en CDL). La figure 5.1 représente la chaine de transformation sous forme d'une architecture utilisant les modèles et les outils nécessaires au processus de vérification formelle des modèles de contrôle d'accès. Les composants de notre architecture, à savoir, AC_Fiacre (Access Control Fiacre) et AC_CDL (Access Control CDL) sont les modèles obtenus après transformation de Fiacre et de CDL.

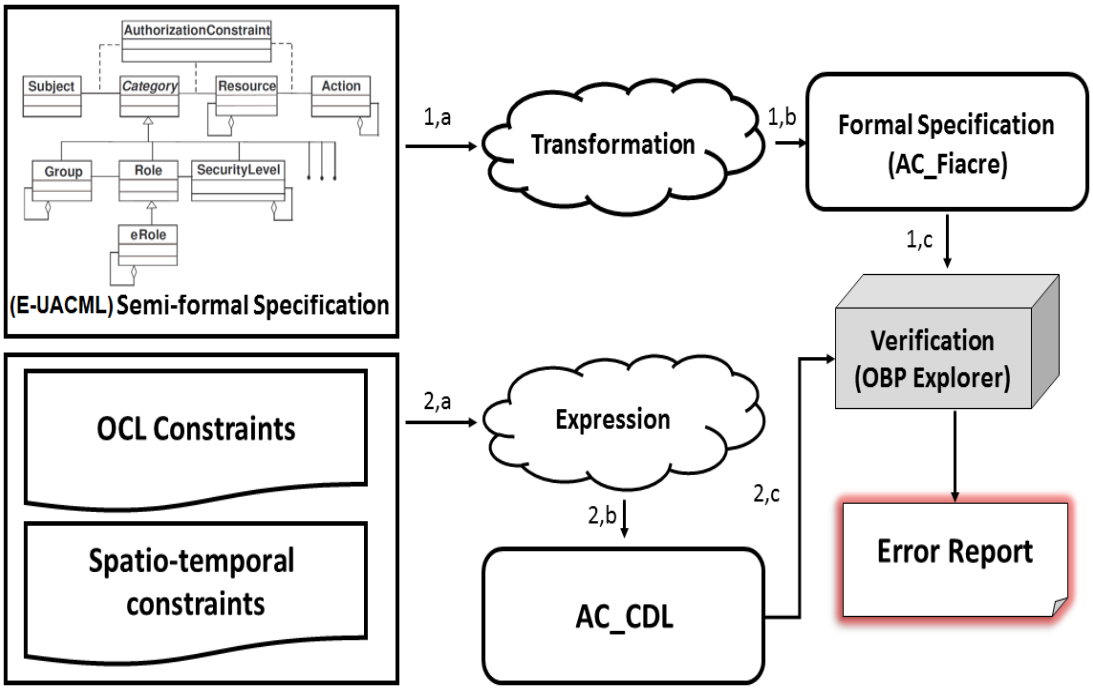

Figure 5.1 Vue d'ensemble de l'architecture proposée [9] [11].

5.3 Algorithme de Traduction du Modèle E-UACML vers Fiacre / CDL

Nous proposons dans cette section un pseudo-algorithme qui montre les entrées et les sorties ainsi que les étapes de notre architecture de transformation.

Algorithme 1.

Input : E-UACMLModel, (classes, association, roles, hiérarchies des rôles, Affectation de sujets à des catégories, affectations entre catégories, affectations aux ressources/actions, affectations obligatoires) et OCL (les contraintes)
Output : A.C Fiacre, A.C CDL, automate observateur, succès/reject

```
1  begin
2      Load (EUACMLModel);
3      //Lecture du nom du fichier E-UACML
4      String fic_euacml ← lectureNomFichierEUACML ();
5      if (fic_euacml == null) return null;
6              Re ← resources;
7              A ← action;
8              C ← categories;
9              R ← Roles;
10             G ← Group ;
11     //Affichage de l'architecture E-UACML (Graphique)
12     AfficherArchitecture (euacml_archi) ;
13     //Transfo de euacml_archi vers une architecture Fiacre (AC_Fiacre)
```

```
14  FiacreArchitectureAC_Fiacre ←
15  fiacreArchiGene.transfoEUACMLArchitecture_to_FiacreArchitecture
16     (uaml_archi);
17  //Generation du modele Fiacre (Program/code)
18  ProgramfiacreModel←fiacreModelGene.generer_model_Fiacre (AC_Fiacre);
19  //Generation du modele Fiacre à partir de l'objet (FiacreArchitecture)
20  Fonction generer_model_Fiacre ()
21          // 1- creation de la liste des references des constantes pour le program
22          constantRefList← create_Constants (AC_Fiacre.constantList);
22          // 2- liste des references des types pour le program
23          typeIdList← create_Types (AC_Fiacre.typeList);
24          // 3- création des process de l'application
25  FiacreProcessGenerator processGen← processGen.create_AllProcess
26              (AC_Fiacre.fiacreProcessList);
27          Creation automate Subject ()
28          Creation automate Roles ()
29          Creation automate Ressources ()
30          Creation automate Actions ()
31          Creation automate Category ()
32          //4- création du component
33          FiacreComponentGenerator componentGen ←
34          componentGen.createComponentDecl (AC_Fiacre);
35  end
36  //Génération du modèle CDL (Program/code)
37  CDLArchitecture AC_CDL ← exprssion(Contraintes);
38  Obpexplorer () ← (AC_Fiacre, AC_CDL);
39  Return (Automate observateur, A.C results(succé/reject)) ;
40 end
```

5.4 Traduction de méta-modèles E-UACML en programmes Fiacre : AC_Fiacre

Cette section présente les principes de la traduction des modèles E-UACML enrichis de contraintes OCL en programmes Fiacre. Plus spécifiquement, nous détaillons comment transformer en processus Fiacre chaque type d'affectation, qui peut être des trois types suivants : (i) Affectations de sujets aux catégories. (ii) Affectations de catégories à d'autres catégories. (iii) Affectations de catégories à des ressources / actions.

5.4.1 Schéma du processus de traduction

Nous présentons ici les principes de la traduction des modèles E-UACML enrichis de contraintes OCL (Figure 5.2) (a) en code Fiacre (Figure 5.2) (b). Plus précisément, nous détaillons comment E-UACML et les contraintes OCL sont transformées en processus Fiacre. La traduction consiste à générer les éléments AC_Fiacre suivants (Figure 5.2) (b) : (i) un ensemble de processus correspondant aux objets actifs du modèle E-UACML. (ii) les processus correspondant aux contraintes OCL. (iii) un composant Fiacre (Component) décrivant l'architecture contenant tous les processus générés.

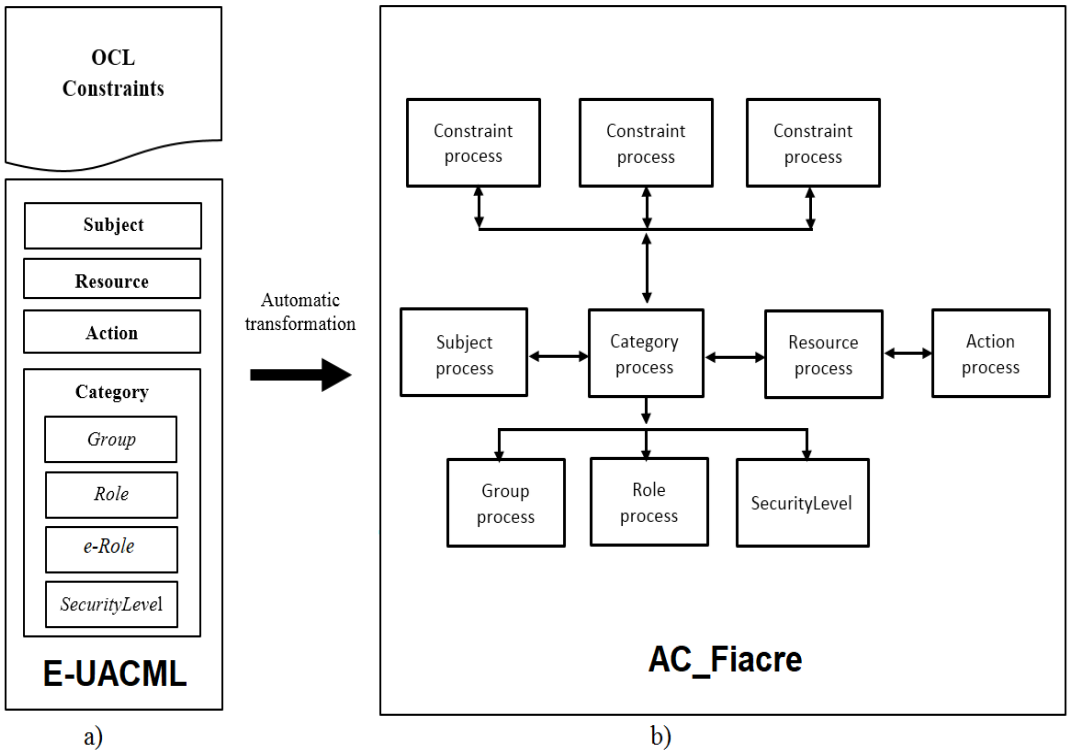

Figure 5.2 Vue globale du processus de traduction E-UACML-Fiacre [9].

Les objets actifs du modèle E-UACML correspondent aux parties fonctionnelles du modèle. Ils sont générés dans des processus Fiacre que nous appelons par leur nom (Subject process, Role process, …) (Figure 5.2 (b)). La traduction est appliquée à tous les éléments « Subject », « Groupe », « Role », « Resource », « Action ». Cette traduction est basée sur une sémantique semi-formelle du langage UML (E-UACML) que nous avons créé et qui a été décrite dans le chapitre 4. Notons que nous n'expliquons pas le principe de transformation complète de UML, qui a fait l'objet d'un autre travail (voir [131]). Les contraintes OCL attachées au modèle E-UACML sont traduites en automate observateur à l'aide du langage (CDL). Les éléments de communication asynchrone sont traduits dans la structure de données de file d'attente

prédéfinie dans Fiacre et la ressource partagée devient une variable partagée associée aux processus Fiacre correspondant, (Figure 5.3).

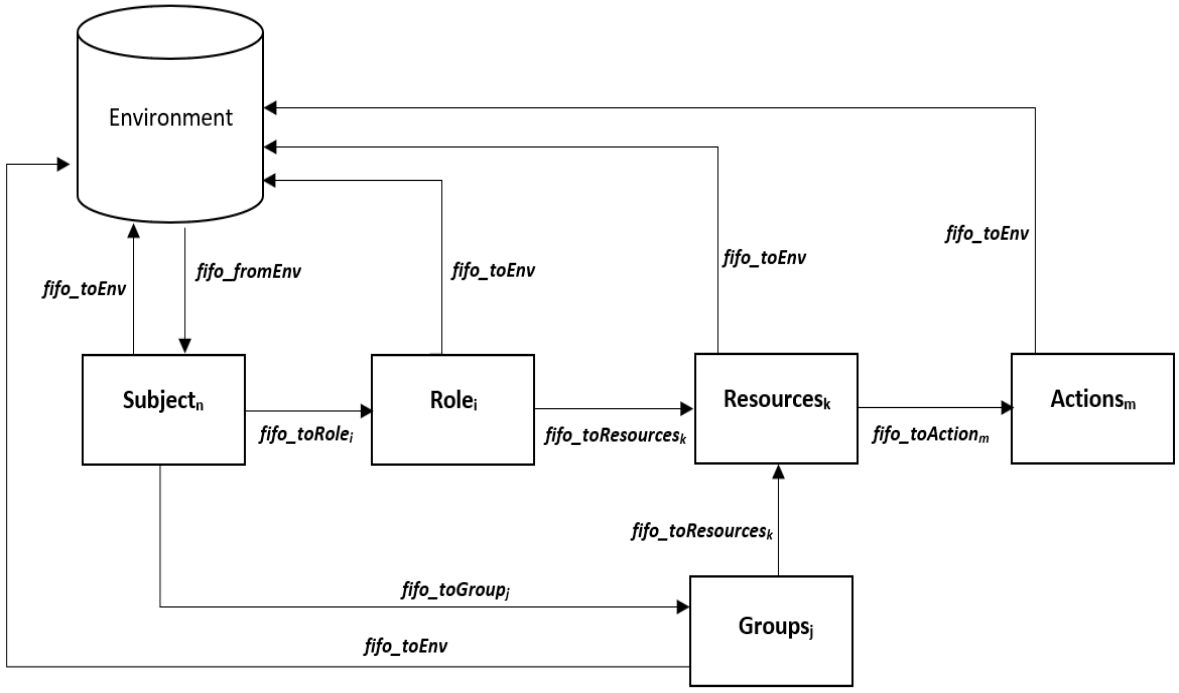

Figure 5.3 Communication asynchrone entre les processus d'AC_Fiacre.

La Figure 5.3 illustre l'architecture Fiacre de notre exemple Pol_1 résultant de la traduction du modèle E-UACML. Dans cette figure, les flèches illustrent les liens de communication asynchrones. Notre algorithme de transformation « Algorithme 1 » génère cinq automates : « Subject », « Role », « Action », « Resource », et « Group ». La Figure 5.3 montre que le processus « Subject$_n$ » est connecté avec le processus « Role$_i$» via une communication asynchrone via une Fifo nommée (fifo_toRole$_i$). De même, « Subject$_n$ » interagit avec l'environnement via deux Fifo (fifo_fromEnv et Fifo_toEnv).

Les processus représentant les éléments E-UACML, les contraintes OCL et l'interprétation de ces contraintes sont systématiquement instanciés dans un composant Fiacre appelé AC_C, et spécifiées en tant qu'entités indépendantes via l'opérateur « || ». Les processus de contraintes et les processus fonctionnels sont tous synchronisés via leurs ports de communication

5.4.2 Construction des automates AC_Fiacre

Partant du schéma du processus de traduction défini en section 5.4.1, et en nous inspirons de [132], nous construisons formellement tous les automates de AC_Fiacre. Nous présentons dans cette sous-section quelques automates à savoir, l'automate

subjects, l'automate rôles, l'automate groupe, l'automate ressource, ainsi que l'automate action.

5.4.2.1 Automate subjects

L'algorithme 2 construit un automate Fiacre pour représenter les « *Subjects* » (Figure 5.4). La variable n stocke le nombre de « Subjects », i le nombre de « *Roles* » et j le nombre de « *Groups* ». Les tuples de l'automate sont définis par les lignes de 9 à 29. La variable « *activ_Role_i* » de type booléen est utilisée pour spécifier l'affectation d'un « Subjects » à un « *Roles* » et « *activ_Group_j* » est utilisée pour spécifier l'affectation d'un « *Subjects* » à un « *Groups* ». La variable « *activ_Subject_n* » de type booléen est utilisée pour indiquer quel sujet est actif.

La structure de l'automate « Subjects » peut ne pas rester la même. Par exemple, un sujet peut être autorisé à activer uniquement un seul rôle ou un seul groupe, tandis qu'un autre sujet peut être autorisé à activer plusieurs rôles et/ou plusieurs groupes.

Les états « *TraiteReq*, *SendToRol*, *SendToGroupe* et *SendRet* » sont créés pour représenter une communication entre les automates « *Subjects* » et « *Roles* »/« *Groupes* » ainsi entre « *Subjects* » et l'environnement.

Algorithme 2. Construction de l'Automate « Subjects »

```
1:  for each n ∈ Subject do
2:  process Subject_n ( &activ_subject_n  : read write bool,
3:                      &activ_role_i     : read write bool,
4:                      &activ_groupe_j   : read write bool,
5:                      &fifo_fromEnv     : read write t_fifo_req,
6:                      &fifo_toRole_i    : read write t_fifo_req,
7:                      &fifo_toGroup_j : read write t_fifo_req)
8:  is
9:  states Stop, Start, TraiteReq, SendToRol,SendToGroupe ,SendRet, Error
10: init to Stop
11: from Stop activ_subject_n := False  to Stop
12: from Stop /*Active Rrole ou/et Goupe*/
13:          activ_subject_n := True; activ_role_i  := True; to Start
14: from Start /* Desactive Role ou/et Groupe*/
15:          activ_subject_n := False; activ_role_i  := False; to Stop
16: from Start /* ENV Send Req to Subject_n*/
17:       case (empty fifo_fromEnv) of false -> null end case;
18:            reqFromEnv := first fifo_fromEnv;
19:            fifo_fromEnv := dequeue (fifo_fromEnv);  to TraiteReq
```

```
20:  from TraiteReq //------- send req to Role -------
21:          if (full fifo_toRole_i ) then loop end;
22:          fifo_toRole_i := enqueue (fifo_toRole_i, reqFromEnv); to SendToRol
23:  from TraiteReq //------- send req to Group -------
24:          if (full fifo_toGroup_j ) then loop end;
25:          fifo_toGroup_j := enqueue (fifo_toGroup_j, reqFromEnv); to
26:                SendToGroup
27:  from TraiteReq        ret := hasRight_Subject_n (req) to Error
28:  from Error //------- send ret to ENV -------
29:                ret := RET_NON to SendRet
30:  from Error loop
31: end for
```

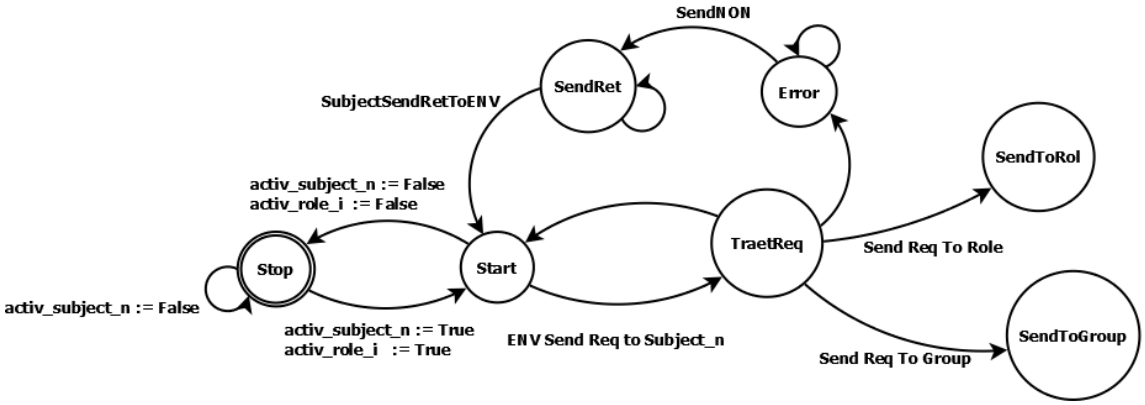

Figure 5.4 Automate Subject.

5.4.2.2 Automate rôle

L'algorithme 3 décrit les étapes requises pour la construction d'automates Roles (Figure 5.6). Pour chaque rôle, un automate est construit. La variable *i* stocke le nombre de « Roles » et *k* stocke le nombre de « *Resources* » affectées à ce « *Role* ». Ensuite, l'automate est construit. Les différents tuples de cet automate Fiacre sont définis dans les lignes 7 à 23. Une variable au minimum est utilisée *(activ_Resources_k*) pour identifier les « *Resources* » qui dépendent de ce Rôle. Les états « *WaitSubject*, *SendToResou* et SendRet » sont créés pour représenter une communication entre l'automate rôles et ressource ainsi qu'entre rôles et subject.

Algorithme 3. Construction de l'automate « Roles »

```
1:  for each i ∈ Roles do
2:    process Role_j (  &activ_Role_i          : read write bool,
3:                      &activ_Resources_k    : read write bool,
4:                      &fifo_fromSubject_n   : read write t_fifo_req,
5:                      &fifo_toResource_k    : read write t_fifo_ret )
6:    is
7:    states Stop, Start, WaitSubject, SendToResou,SendRet, Error
8:    init  to Stop
9:    from Stop
10:   if (activ_ Role_i  = True) then activ_Resources_k:= True; to Start end
11:   from Start
12:   if (activ_ Role_i  = False) then activ_Resources_k:= False; to Stop end
13:   from Start /* Subject_n Send Req to Role_i */
14:         case (empty fifo_fromSubject_n) of false -> null end case;
15:         reqFromSubject_n := first fifo_fromSubject_n;R
16:         fifo_from Subject_n := dequeue (fifo_from Subject_n); to WaitSubject
17:   from WaitSubject //------- send req to Resources_k -------
18:         if (full fifo_toResources_k ) then loop end;
19:         fifo_toResources_k := enqueue (fifo_ toResources_k, reqFromSubject_n);
20:   to SendToResou
21:   from WaitSubject  ret := hasRight_ Role_i (req) to Error
22:   from Error loop
23:   from Error ret := RET_NON /*@ Send NON*/  to SendRet
24: end for
```

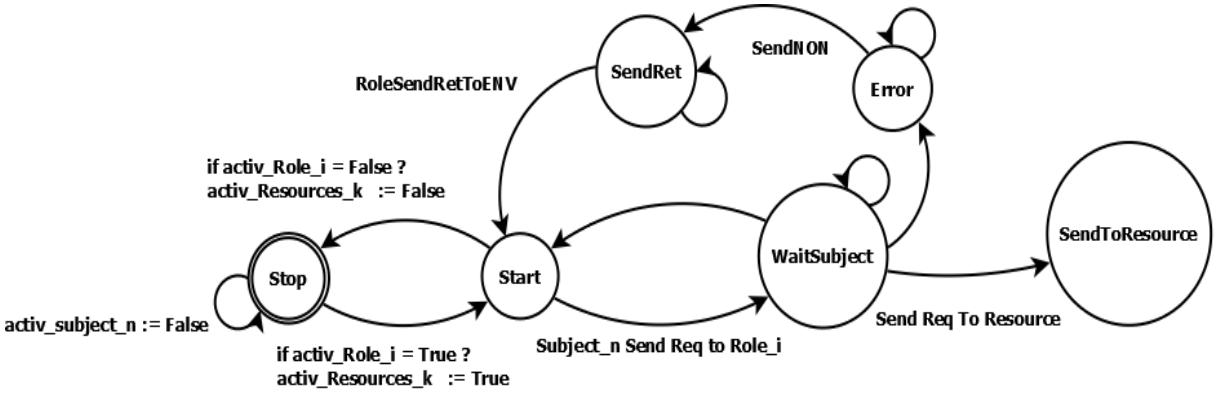

Figure 5.6. Automate Rôles.

5.4.2.3 Automate groupe

L'algorithme 4 décrit les étapes requises pour la construction d'automates « *Groups* » (Figure 5.7). Pour chaque groupe, un automate est construit. La variable *j* stocke le nombre de « *Groups* » et k stocke le nombre de « *Resources* » affectées à ce « *Groups* ». Ensuite, l'automate est construit. Les différents tuples de cet automate Fiacre sont définis dans les lignes 7 à 23. Une variable au minimum est utilisée (*activ_Resources_k*) pour identifier les « Resources » qui dépendent de ce groupe.

Pour représenter les comportements d'activation et de désactivation, deux états de contrôle appelés « *Stop* » et « *Start* » sont créés où « *Stop* » est considéré comme l'état initial. Avec la première affectation d'utilisateur, un groupe dans l'état « Stop » passe à l'état « *Start* ». Une affectation d'utilisateur est effectuée par une opération d'activation invoquée par l'utilisateur. Toute opération d'activation ultérieure ne modifie pas l'état du groupe. Lorsque tous les utilisateurs sont non affectés au groupe, le groupe revient à l'état « *Stop* ». À partir de l'état « Start », un groupe peut revenir à l'état « Stop ». Maintenant, la transition de « Stop » à « Start » est marquée par une opération d'activation de groupe initiée par un utilisateur. Cette action est représentée par (*if (activ_Group_j = True))* et (*activ_Resources_k:= True*) où (*activ_Group_j*) est le groupe considéré et (*activ_Resources_k*) représente la ressource qui est activée par ce groupe.

Algorithme 4 : Construction de l'automate « *Groupe* »

```
1  for each j ∈ Groups do
2     process Group_j (&activ_Group_j          : read write bool,
3                      &activ_Resources_k      : read write bool,
4                      &fifo_fromSubject_n     : read write t_fifo_req,
5                      &fifo_toResource_k      : read write t_fifo_ret )
6     is
7     states Stop, Start, WaitSubject, SendToResou,SendRet, Error
8     init  to Stop
9     from Stop
10    if (activ_Group_j  = True) then activ_Resources_k:= True; to Start end
11    from Start
12    if (activ_Group_j  = False) then activ_Resources_k:= False; to Stop end
13    from Start /* Subject_n Send Req to Group_j */
14          case (empty fifo_fromSubject_n) of false -> null end case;
15          reqFromSubject_n := first fifo_fromSubject_n;
16          fifo_from Subject_n := dequeue (fifo_from Subject_n); to WaitSubject
17    from WaitSubject //------- send req to Resources_k -------
18          if (full fifo_toResources_k ) then loop end;
19          fifo_toResources_k := enqueue (fifo_ toResources_k, reqFromSubject_n);
```

```
20    to SendToResou
21    from WaitSubject  ret := hasRight_Group_j (req)      to Error
22    from Error loop
23    from Error ret := RET_NON /*@ Send NON*/ to SendRet
24  - end for
```

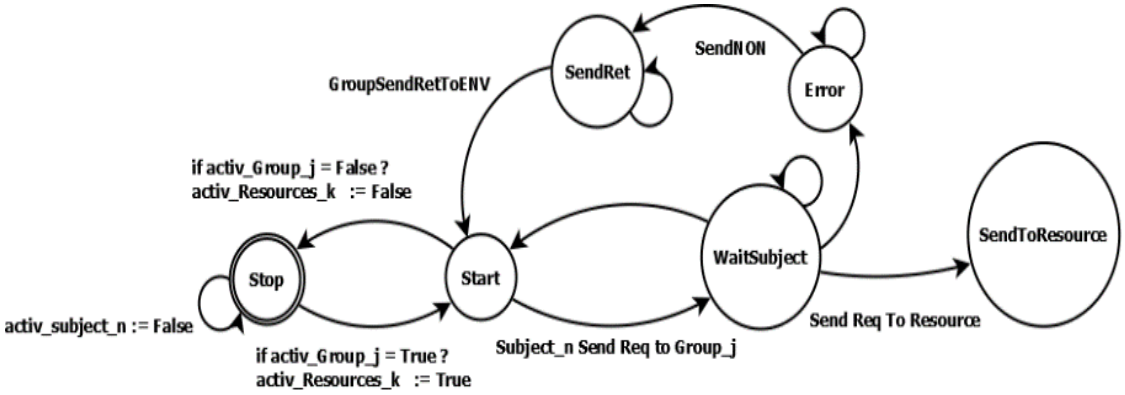

Figure 5.7 Automate Groupe.

5.4.2.4 Automate ressources

L'algorithme 5 décrit les étapes requises pour la construction d'automates « *Resources* » (Figure 5.8). Pour chaque Resource, un automate est construit. La variable *k* stocke le nombre de « *Resource* » *m* stocke le nombre de « *Actions* » *i* le nombre de « *Roles* » et *j* le nombre de « *Groups* » affectées à cette « Ressources ». Ensuite, l'automate est construit. Les différents tuples d'un automate Resource sont définis dans les lignes 8 à 31. Trois variables au minimum sont utilisées (*activ_Role_i et/ou activ_Group_j, activ_Resources_k , activ_Action_m*) pour identifier la « *Category* » (Rôle et/ou bien Groupe) et les « *Action* » qui dépendent de cette Resource.

Algorithme 5 : Construction de l'automate « *Resource* »

```
1   for each k ∈ Resources do
2     process Resourses_k (&activ_Resources_k   : read write bool,
3                          &activ_Action_m      : read write bool,
3                          &activ_Role_i        : read write bool,
3                          &activ_Group_j       : read write bool,
4                          &fifo_fromRoles_i    : read write t_fifo_req,
5                          &fifo_fromGroups_j   : read write t_fifo_req,
6                          &fifo_toActions      : read write t_fifo_ret)
7     is
8     statesStop, Start, WaitRole,WaitGroup, SendToAction,SendRet, Error
```

```
9     init  to Stop
10    from Stop
11          if (activ_resources_k = True) then activ_action_m := True; to Start end
12    from Start
13    if (activ_resources_k = False) then activ_action_m := False; to Stop end
14    from Start /* Roles_i Send Req to Resourses_k */
15          case (empty fifo_fromRoles_i) of false -> null end case;
16          reqFromRoles_i:= first fifo_fromRoles_i;
17          fifo_fromRoles_i:= dequeue (fifo_fromRoles_i); to WaitRole
18    from Start /* Groups_j Send Req to Resourses_k */
19          case (empty fifo_fromGroups_j) of false -> null end case;
20          reqFromGroups_j:= first fifo_fromGroups_j;
21          fifo_fromGroups_j:= dequeue (fifo_fromGroups_j); to WaitGroup
22    from WaitRole //------- send req to Action_m -------
23          if (full fifo_toAction_m ) then loop end;
24          fifo_toAction_m:=enqueue (fifo_toAction_m, reqFromRole_i);to
SendToAction
25    from WaitRole  ret := hasRightResourses_k (req) to Error
26    from WaitGroup //------- send req to Action_m -------
27    if (full fifo_toAction_m ) then loop end;
28        fifo_toAction_m:=enqueue(fifo_toAction_m, reqFromGroups_j); to
SendToAction
29    from WaitgGroup  ret := hasRightResourses_k (req)  to Error
30    from Error loop
31    from Error ret := RET_NON /*@ Send NON*/ to SendRet
32  end for
```

Lorsqu'un rôle ou un groupe est dans l'état « *Start* », les ressources qui lui sont associées sont rendues disponibles. Mais dans la plupart des situations, il est souhaitable que lorsque le rôle ou le groupe est actif, toutes les ressources associées soient également disponibles pour les utilisateurs du rôle ou du groupe. Avec la première opération d'activation de rôle ou groupe, les ressources associées peuvent être accessibles par les utilisateurs de ce rôle ou groupe, et lorsque tous les utilisateurs désactivent le rôle, les ressources ne sont plus accessibles. Pour cela deux emplacements « Stop » et « Start » sont créés. La transition de « Stop » à « Start » est étiquetée avec deux variables partagées (*if (activ_resources_k = True)*) et (*activ_action_m:=True*) et la transition de « Start » à «Stop» est étiquetée avec (*activ_action_m := False*).

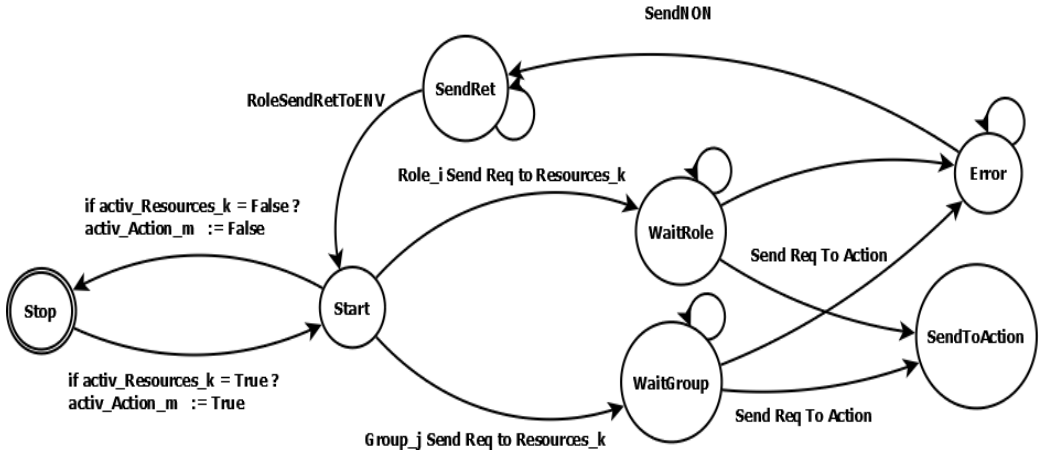

Figure 5.8. Automate Ressource.

5.4.2.5 Automate action

L'algorithme 6 est utilisé pour construire l'automate Action (Figure 5.9). Pour chaque « *Action* » il y a un automate Fiacre correspondant. La variable k stocke le nombre d'automate « *Resources* » affectées à « *Action* ». Ensuite, l'automate est construit. Les différents tuples de l'automate Fiacre sont définis dans les lignes 6 à 19.

Lorsqu'une ressource est dans l'état « *Start* », les actions qui lui sont associées sont rendues disponibles. Mais dans la plupart des situations, il est souhaitable que lorsqu'une ressource soit active, toutes les actions associées soient également disponibles. Avec la première opération d'activation de ressource, les actions associées peuvent être accessibles par les utilisateurs de cette ressource, et lorsque tous les utilisateurs désactivent le rôle, les ressources ne sont plus accessibles et les actions aussi. Pour cela deux emplacements « *Stop* » et « *Start* » sont créés. La transition de « *Stop* » à « *Start* » est étiquetée avec la variable partagée (*activ_action_m*) et la transition de « *Start* » à « *Stop* » est étiquetée avec refus de (*activ_action_m*).

Algorithme 6. Construction de l'automate « Action »

```
1: for each k ∈ Resource do
2:     process Actions_m (&activ_Action_m  : read write bool,
3:                        &fifo_fromResourses_k  : read write t_fifo_req)
4:     is
5:     states Stop, Start, WaitResource, SendRet, Error
6:     init to Stop
7:     from Stop  if (activ_action_m = True) then to Start end
8:     from Start if (activ_action_m = False) then to Stop end
9:     from Start /* Resourses_k Send Req to Action_m */
10:        case (empty fifo_from Resourses_k) of false -> null end case;
```

```
11:          reqFromResourses_k:= first fifo_fromResourses_k;
12:          fifo_fromResourses_k := dequeue (fifo_fromResourses_k);
13:               to WaitResource
14:   from WaitResource //------- send ret to ENV -------
15:               ret := RET_OK  to SendRet
16:   from SendRet loop
17:   from WaitResource //------- send ret to ENV -------
18:               ret := RET_NON  to SendRet
19:   from Error loop
20: end for
```

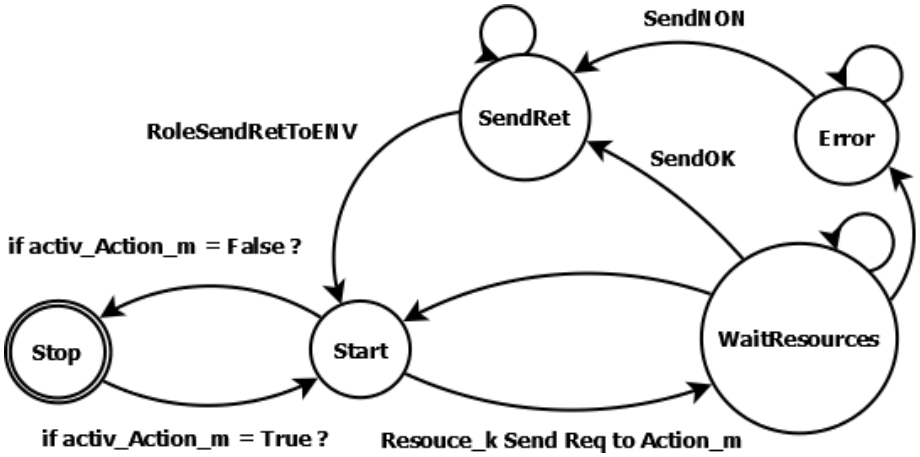

Figure 5.9 Automate Action.

5.4.3 Transformation de l'exemple Pol_1 en processus Fiacre

La Figure 5.10 illustre les automates Fiacre de notre exemple Pol_1 résultant de la traduction du méta-modèle source E-UACML.

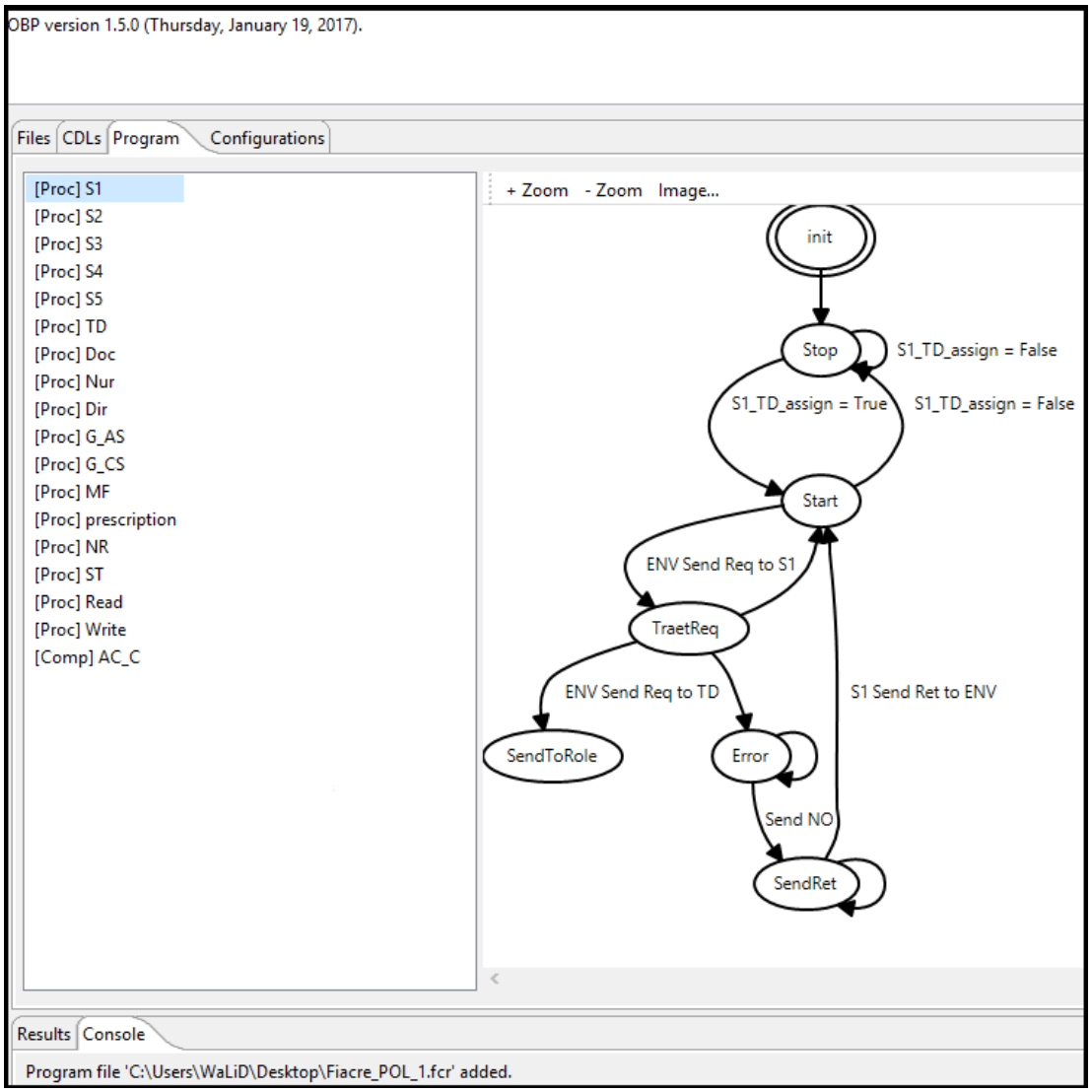

Figure 5.10 Aperçu des automates Fiacre de Pol_1 sur OBP [9].

Notre algorithme de transformation (Algorithme 1) génère dix-sept automates : (i) les automates subjects : « *S1* », « *S2* », « *S3* », « *S4* » et « *S5* ». (ii) les automate Roles : « *TD* », « *Doc* », « *Nur* » et « *Dir* ». (iii) les automates Groups « *G_AS* » et « *G_CS* ». (iv) les automates Ressources « *MF* », « Prescription », « *NR* » et « *ST* ». (v) et les automates Actions : « *Read* » et « *Write* ». Les processus représentant les éléments de E-UACML, les contraintes OCL et l'interprétation de ces contraintes sont systématiquement instanciés dans un composant Fiacre appelé *AC_C*, et spécifié en tant qu'entités indépendantes via l'opérateur « || ». Les processus de contrainte et les processus fonctionnels sont tous synchronisés via leurs ports de communication.

5.5 Spécification des propriétés de contrôle d'accès à l'aide de CDL : AC_CDL

Afin de vérifier les exigences exprimées pour un modèle de contrôle d'accès, il est nécessaire de les spécifier comme des propriétés formelles qui peuvent être interprétées par le vérificateur de modèle ciblé. En outre, pour valider les exigences, l'environnement dans lequel le système doit évoluer peut nécessiter d'être décrit. Dans notre approche, nous utilisons le langage CDL pour (i) exprimer les propriétés de sécurité en tant qu'automate d'observateur, et (ii) spécifier l'interaction entre l'environnement et le modèle du système. Dans cette étude, nous exprimons des propriétés CDL en suivant deux objectifs complémentaires : (i) pour vérifier que la mise en œuvre des contraintes OCL est correcte, (ii) pour s'assurer que les parties fonctionnelles du modèle sont correctement mises en œuvre.

5.5.1 Spécification du contexte (scénarios)

Lors des vérifications, les états du modèle AC_Fiacre sont explorés par simulation. Lors de ces explorations, nous appliquons des scénarios qui simulent le comportement de l'environnement qui interagit avec la politique de contrôle d'accès (par envoi de requête) où nous décrivons les interactions entre la politique de contrôle d'accès et l'environnement. Leurs comportements sont décrits comme expliqués dans [112], [133]. Les scénarios définissent les interactions d'acteurs légitimes qui communiquent avec le modèle, mais également le comportement d'éventuels attaquants. Les scénarios sont modélisés comme des graphes acycliques comprenant des opérations d'envois et de réception de messages vers et du modèle d'architecture à valider. Les diagrammes de scénarios sont organisés avec des séquences et des alternatives. Chaque scénario est entièrement décrit par des diagrammes de séquence (Figure 5.11).

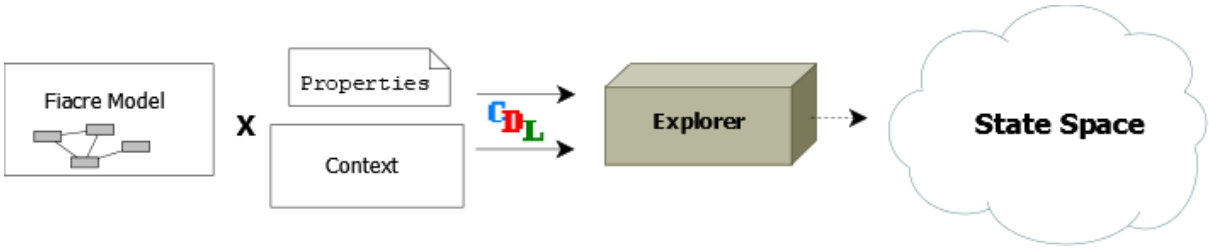

Figure 5.11 Exemple de contextes (scenario CDL).

Un programme CDL peut comporter plusieurs scénarios. Un scénario peut référencer une ou des propriétés et un ou des invariants sur les prédicats. Par exemple, le scénario Pol_1_cdl ci-dessous référence les propriétés pty_R1, pty_R2, pty_R3 et pty_4.

Sur Pol_1, si un sujet suppose transmettre ses informations pour accéder à une ressource, nous commençons par la définition des constantes cst_R1, cst_R2, cst_R3 et cst_R4, par des variables structurées, dans le modèle Fiacre de la manière suivante :

const cst_R1 : T_req is {subjects = S1, roles = TD, groupes= CS, Resources = MF, actions = R }
const cst_R2 : T_req is {subjects = UNKNOW, roles = TD, groupes= CS, Resources = MF, actions= R }
const cst_R3 : T_req is {subjects = S1, roles = ER1, groupes= CS, Resources = MF, actions = W }
const cst_R4 : T_req is {subjects = S1, roles = TD, groupes= CS, Resources = ST, actions = R }

Ensuite nous décrivons ces interactions (émission asynchrone du message « *cst_R1* » « *cst_Regle2* » « *cst_R3* » et « *cst_R4* » en provenance de l'environnement vers l'instance 1 du processus « *Subject* ») avec CDL par (event) comme suit :

event evt_send_R1 is {send cst_R1 from {env}1 to {Subject}1}
event evt_send _ R2 is {send cst_R2 from {env}1 to {Subject}1}
event evt_send _R3 is {send cst_R3 from {env}1 to {Subject}1}
event evt_send _ R4 is {send cst_R4 from {env}1 to {Subject}1}

Enfin nous spécifions le retour à l'environnement (émission asynchrone du message OK ou NO en provenance de l'instance 1 du processus « Action » vers l'environnement) comme suit:

event evt_send_OK is { send OK from {Action}1 to {env}1}
event evt_send_NO is { send NO from {Action}1 to {env}1}

Nous pouvons regrouper des ensembles de requêtes par des activités « *activity* » pour construire des séquences (avec l'opérateur « ; ») ou des alternatives (avec l'opérateur "[]") de requêtes. Des exemples d'activités act_clt1 et act_clt2 sont illustrés ci-dessous :

activity act_ret is { event evt_recv_OK [] event evt_recv_NON }
activity act_ R1 is { event evt_ R1; act_ret }
activity act_ R2 is { event evt_ R2; act_ret }
activity act_ R3 is { event evt_ R3; act_ret }
activity act_ R4 is { event evt_ R4; act_ret }

Les événements evt_recv_OK et evt_recv_NON spécifient les réceptions d'un marqueur de fin des activités act_ R1, act_ R2, act_ R3, act_ R4.
Le contexte est finalement décrit de la manière suivante :

cdl cdl_test_Pol_1 ***is***
{properties pty_ R1, pty_ R2, pty_ R3, pty_ R4
main is { act_ R1 || act_ R2 || act_ R3 || act_ R4 }}

5.5.2 Spécification des propriétés CDL pour Pol_1

Nous illustrons ici les spécifications de certaines propriétés associées aux contraintes incluses dans notre exemple Pol_1.

Reprenons la règle R2 de l'exemple Pol_1 : le rôle d'urgence ER1 est assigné pendent 30 minutes à un utilisateur (Exigence 1) uniquement lorsque ce dernier est dans l'emplacement « *Emergency_room* » (Exigence 2). Dans notre cas simplifié, les exigences 1 et 2 sont spécifiées en langage CDL suivant les principes : Une propriété CDL *prop*, de type observateur, fait référence à des occurrences d'événements *evt*. La propriété *property prop is : start − −// evt / − > reject* spécifie que l'observateur est mis en défaut (faux) si l'occurrence de l'événement *evt* est détectée lors de l'exploration du modèle. L'événement *evt* peut être défini par un changement d'état d'un prédicat *pred* comme par exemple : *event evt is : pred becomes true*. Enfin, le prédicat *pred* est défini de la façon suivante : *predicate pred is : x = 1* qui spécifie que *pred* est évalué à vrai si la variable x est égale à la valeur 1. Selon cette syntaxe, les exigences de notre modèle sont spécifiées comme suit :

Nous utilisons une horloge C (Propriété temporisée) Ou *c := -1* permet de dévalider l'horloge :

Listing 2. CDL pour Exigence 1

```
property time_R2 is
{ clock c;
      start  -- // evt_start / c := 0 ->wait;
      // reçu start entre 8 et 16 ?
      wait  -- c > t+30   //   / c :=-1 ->reject;
      wait  -- c < t   // evt_stop / c :=-1 ->reject;
     wait  --     // evt_stop  / c :=0 ->start
}
```

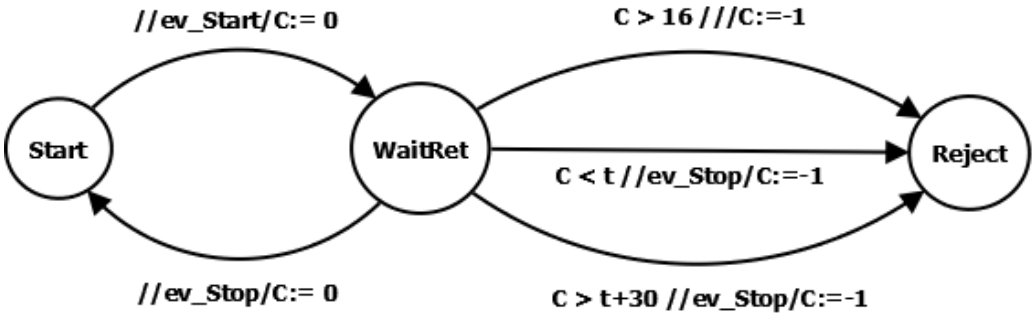

Figure 5.12 Automate observateur pour exigence1.

Listing 3.CDL pour Exigence 2

```
predicate pred2    is { {R}1: emplacement = Emergency_room }
event evt2         is { pred2 becomes true }
property prop2  is { start -- / / evt2 / -> reject }
```

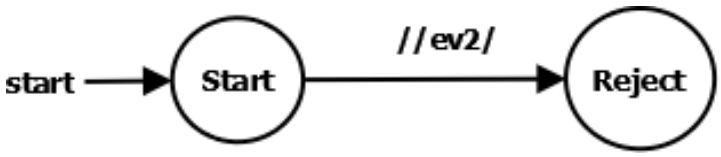

Figure 5.13 Automate observateur pour exigence 2.

Les propriétés, à vérifier sur notre exemple Pol_1, concernent l'acquittement d'une opération de lecture ou d'écriture d'une ressource détenue par un subject. La propriété pty_R1 exprime formellement que l'envoi (*send*), par l'entité subject, d'une requête portant la constante *cst_R1* (section 5.5.1), est acceptée ou refusée. Si la requête est acceptée, la réponse porte l'événement *evt_send_OK*. Si elle est refusée, elle porte l'événement *evt_send_NO* défini précédemment en section 5.5.1.

```
property  pty_R1
is
  {  start      -- / / evt_send_R1/  -> waitRet;
     waitRet  -- / / evt_send_OK     /  -> start;
//----------  errors ----------
     waitRet  -- / / evt_send_NON  /  -> reject
}
```

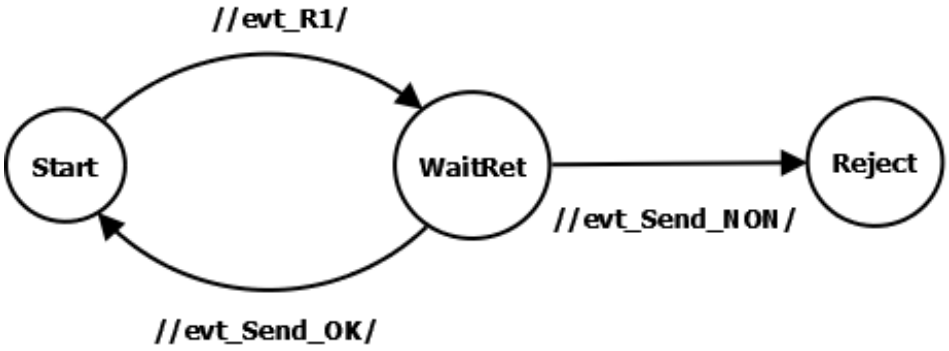

Figure 5.14 Automate observateur de la propriété R1.

```
property  pty_R3
 is
{       start      --  // evt_send_R3/  -> waitRet;
       waitRet   --  // evt_send_OK    /  -> succes;
         //--------------  errors ----------------------
      waitRet  --  // evt_send_NON  /  -> reject}
```

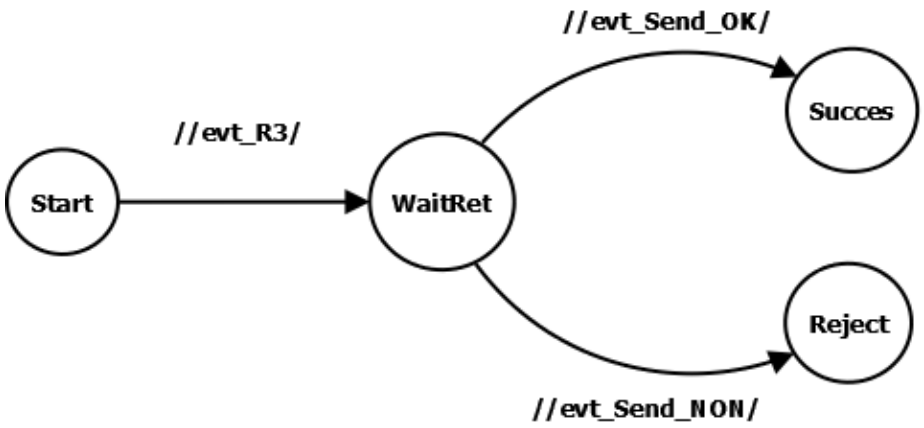

Figure 5.15 Automate observateur de la propriété R3.

```
property  pty_R4

is
{       start     --  // evt_send_R4/  -> waitRet;
       waitRet --  // evt_send_OK      /  ->succes;
                  //--------------  errors ----------------------
       waitRet --  // evt_send_NON  /  ->reject }
```

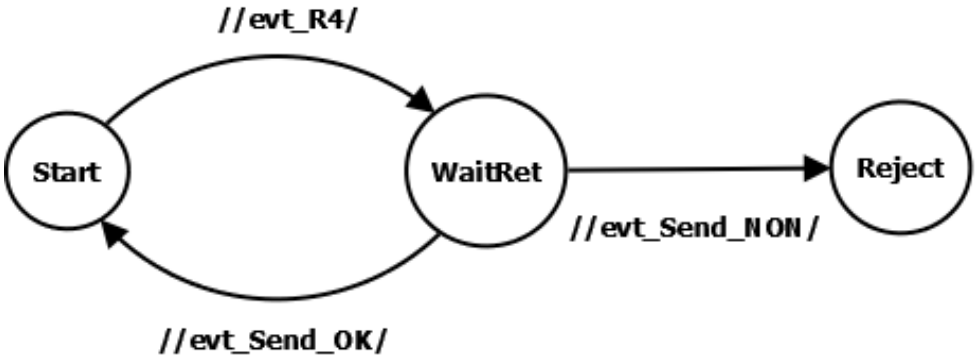

Figure 5.16 Automate observateur de la propriété R4.

5.6 Résultats des vérifications du modèle E-UACML (Pol_1) avec l'outil OBP

Nous avons exploré le modèle d'exigences de E-UACML en utilisant la méthode du model-checking par OBP Explorer.

Nous mettons en œuvre cette technique d'exploration des modèles et de vérification des propriétés sur la première architecture simple (Pol_1). Cette politique est présentée sous une forme simple.

Subjects	Nombre de rôles	Nombre de Groupes	Taille de FIFO	State	Transitions	Temps	Observation
S1	02	01	5	309 120	3 558 880	17.215 seconds	Exploration complète
S2	01	01	5	309 120	3 546 448	15.228 seconds	Exploration complète
S3	01	01	5	307 104	3 534 312	14.898 seconds	Exploration complète
S4	00	01	5	309 120	3 557 712	14.775 seconds	Exploration complète
S5	01	01	5	309 120	3 544 318	14.798 Seconds	Exploration complète
UNKNOW	05	2	5	309 120	35 558 712	17.320 Seconds	Exploration complète

Table 5.2 Explorations OBP de l'exemple Pol_1.

Pour les mesures consignées dans cette table, nous faisons varier le nombre de rôles et groupes pour chaque utilisateur, ainsi que le nombre de ressource et d'action pour chaque groupe et rôle.

Sur cet exemple simple, nous indiquons, dans table 5.2 les résultats des explorations de la politique Pol_1. La table 5.2 indique la complexité en nombre de configurations et de transitions du graphe SdT (système de transitions) de comportement généré selon les différents scénarios impliqués. Ces ensembles d'interaction sont caractéristiques du trafic entre l'environnement et la politique de contrôle d'accès.

Avec CDL, nous avons spécifié des observateurs pour exprimer et vérifier une gamme de propriétés correspondant aux exigences des règles de la politique Pol_1 comme indiqué dans la Section 5.5. Nous avons spécifié des diagrammes de scénarios AC_CDL et à partir de ces diagrammes, OBP génère un graphe acyclique (appelé graphe de contexte) qui représente toutes les interactions possibles entre le modèle et l'environnement (Figure 5.17).

Une attaque simple est ajoutée et qui consiste à représenter des scénarios avec des interactions d'entités non légitimes de l'environnement (attaquants). Nous considérons deux types d'attaquants qui sont susceptibles d'émettre des requêtes vers la politique de contrôle d'accès. (i) Les requêtes générées par un attaquant inconnu : *{subject = {UNKONWN}, Roles = {TD}, groupes = {CS}, resources = {MF}, actions = {R}}* (Figure 5.17). Un message de réponse (*NO*) est renvoyé à l'environnement. (ii) Les requêtes générées par un attaquant supposé être interne à la politique de sécurité : *{subject = {S1}, Roles = {TD}, groupes = {CS}, ressources = {ST}, actions = {R}}* (Figure 5.17). Un message de réponse (*NO*) est renvoyé à l'environnement.

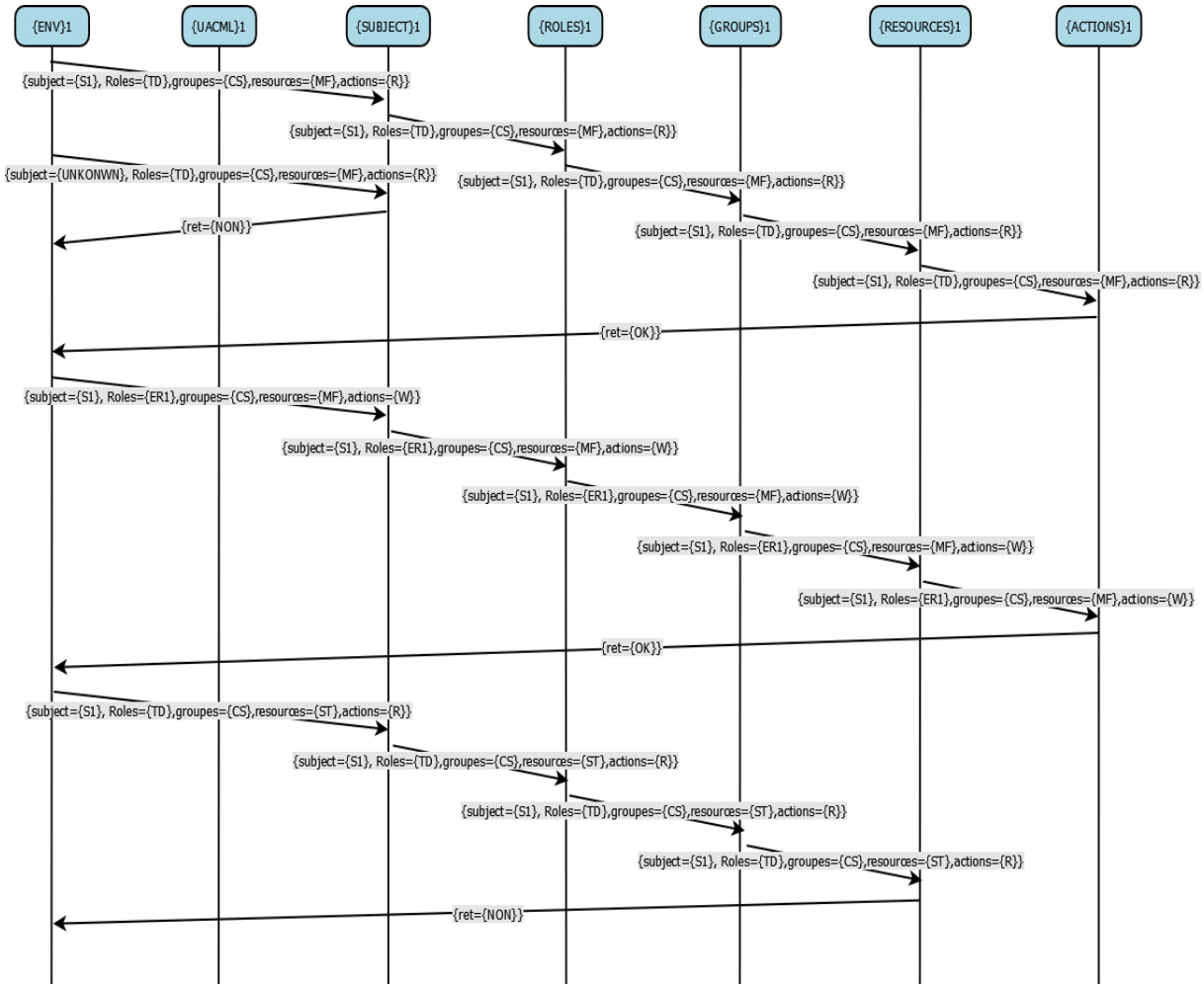

Figure 5.17 Représentation de la trace d'exploration sous la forme d'un diagramme de séquence.

Le nombre de configurations et de transitions LTS générées lors de l'exploration par OBP, est de taille raisonnable. Par exemple, pour les règles R1, R2, R3, R4 et R5 la taille des Fifo étant de 5, le nombre de configurations générées lors de l'exploration du modèle de chaque subject est entre 307 104 et 309 120 et le nombre de transitions est entre 3 534 312 et 3 558 880.

5.6.1 Résultats de l'exploration avec la technique de partitionnement de scenarios

La table 5.3 présente les résultats du modèle avec une taille de FIFO égale à 10. Nous avons augmenté la taille de la file partagée entre l'environnement et les automates Fiacre qui est passée de 5 à 10 (Ceci correspond au nombre d'information transmis par chaque automate Subject).

Split	Subject	Nombre de rôle	Nombre de groupe	Taille de FIFO	Transition	State	Sous contexte
Non	S1	02	02	10	-	Explosion	
Oui	S1	02	02	10	315 364 947	27 231 482	4

Table 5.3 Résultats de l'exploration avec la technique de partitionnement de scénarios.

Prenons le Subject S1 de la table 5.2 avec la taille de la file est égale à 5, le nombre de configurations explorées est donc 309 120 et le nombre de transition est de 3 558 880. Pour ce cas d'étude, la complexité d'exploration est de taille raisonnable.

Par contre le subject S1 (table 5.3) qui a 02 rôles et 02 groupes et une taille de file égale à 10, nous remarquons une explosion d'états (table 5.3) à cause de la limitation d'espace mémoire. Si nous augmentons la taille de la FIFO, nous remarquons une explosion dans le nombre de comportements. Dans ce cas, l'analyse des propriétés ne peut être complète.

Dans le cas de l'explosion des états, nous exploitons l'opération SPLIT qui est intégrée à l'outils OBP pour partitionner les interactions des scénarios de l'environnement avec le modèle. Cette technique permet aux modèles possédant un grand nombre d'états d'être exploité. Par exemple, dans le cas d'un automate Subject qui transmet 10 informations d'accès à une ressource, nous trouvons que l'exploration ne se termine pas. Ainsi, après l'application du partitionnement des contextes, OBP partitionne en 4 sous-contextes et l'analyse se termine normalement.

Dans la table 5.3 nous obtenons après le SPLIT un nombre de configurations explorés (cumulatives) de 27 231 482, et un nombre de transitions de 315 364 947.

Ces résultats nous montrent le cas de l'explosion combinatoire intrinsèquement associée avec des modèles de technique d'exploration, l'outil OBP peut fournir une vérification de correction de propriétés, même avec une machine ayant une mémoire limitée de taille (08 GB). Si le modèle a un nombre de comportement compatible avec la recherche exhaustive, les résultats de vérification peuvent être obtenus sans utiliser la technique de partitionnement implémentée en OBP.

Tous les objectifs que nous nous sommes fixés dans notre approche sont atteints à cette étape. Nous avons créé un cadre pour la modélisation des systèmes de contrôle d'accès hybrides. Cependant notre outil dans la version actuelle est expérimental. Certains éléments comme l'expression d'une hiérarchie ne sont pas pris en compte. Aussi dans la

traduction vers Fiacre nous avons ajouté uniquement les événements qui sont sous forme de constructeurs. En plus, La question du passage à l'échelle et l'application de notre méthode à un domaine réel fera l'objet d'au travail qui va suivre.

5.7 Conclusion

A l'issue de notre étude sur les méthodes de formalisation et de vérification du contrôle d'accès, nous remarquons que les méthodes sémantiques (Model-checking) sont utilisées pour s'assurer qu'une politique est sûre. Toutefois, certaines propositions ne répondent pas à tous les objectifs de sécurité et peu de travaux prennent en compte le facteur temps sachant la complexité de la vérification qui en découle. Néanmoins, nous pensons qu'une approche de vérification nécessite le choix d'un formalisme adapté à l'expression des aspects de sécurité. La problématique porte ainsi, sur la modélisation d'une part et la vérification d'autre part. De ce fait, notre objectif était de créer un cadre pour la spécification des systèmes de contrôle d'accès. Nous avons utilisé une méthode de spécification semi-formelle appelée UACML accompagnée par un modèle de contraintes spatio-temporel du type STRBAC et une méthode de spécification formelle à l'aide du langage Fiacre.

Notre architecture présente les modèles et les outils nécessaires au processus de vérification formelle des modèles de contrôle d'accès en passant par un processus de transformation de E-UACML vers Fiacre pour obtenir AC_Fiacre. A partir de AC_Fiacre et AC_CDL, OBP procède à la vérification. Notre méthodologie trace les étapes nécessaires à la vérification du méta-modèle E-UACML qui permet de réduire l'écart entre les modèles semi-formels pour décrire les comportements du système à valider et les langages formels utilisés en entrée du processus de model-checking afin d'exploiter ce que ces modèles permettent d'exprimer.

PARTIE IV: CONCLUSION ET PERSPECTIVES

Chapitre 6: Conclusion et Perspective

SOMMAIRE

6 Conclusion et pespectives

6.1 Conclusion générale

L'objectif de notre thèse est de créer un cadre pour la spécification des systèmes de contrôle d'accès hybrides. Pour atteindre cet objectif, nous avons utilisé une méthode de spécification semi-formelle appelée E-UACML et une méthode de spécification formelle appelée Model-cheking.

U-EACML est une méthode qui nous permet de spécifier sous forme de diagrammes UML, les besoins en contrôle d'accès d'un système. Le model-cheking permet quant à lui de spécifier formellement les exigences d'un système et de prouver que ces exigences répondent bien aux attentes.

Pour élaborer notre processus de modélisation, nous avons combiné ces deux méthodes de modélisation afin de créer des modèles de contrôle d'accès à la fois simples et sûrs par construction. Notre langage E-UACML nous a permis de créer des modèles simples et facilement compréhensibles grâce à sa notation graphique. Le model-cheking nous a permis de spécifier des opérations d'accès et de prouver que ces opérations ne contredisent pas l'ensemble des propriétés établies dans les modèles E-UACML.

La nécessité de trouver des techniques pour modéliser des systèmes de contrôle d'accès hybrides découle du fait que de plus en plus d'organisations ont besoin de mettre en œuvre des modèles de contrôle d'accès basés sur la combinaison des concepts de différents modèles de contrôle d'accès. La politique de sécurité Pol_1, que nous avons traité dans les chapitres précédents en est un exemple. Pour mettre en œuvre cette politique, nous avons fait appel aux concepts rôle, rôle d'urgence, groupe de sujets groupe de ressources et au contrainte saptio-temporelle.

Puisque ces différents concepts n'existent pas tous dans un même modèle de contrôle d'accès connu, nous sommes obligés de créer un nouveau modèle qui regroupe ces concepts et qui répond aux besoins de la politique à implémenter.

Des situations comme celles qui ont conduit à la création de la politique Pol_1 peuvent découler d'une fusion d'entreprise, du changement du mode de fonctionnement d'une entreprise, etc.

6.2 Perspectives

Nous avons pu à travers ce travail atteindre nos objectifs fixés, cependant un certain nombre d'améliorations peuvent être envisagé.

Durant notre travail, nous avons été confrontés à cette vaste problématique où beaucoup de paramètres sont à prendre en compte pour répondre au besoin de la formalisation des politiques de contrôle d'accès.

La chaine de transformation doit être toujours en amélioration pour, d'un côté, lever certaines restrictions posées sur le langage d'entrée et, de l'autre, permettre de cibler d'autres outils de vérification. A cet effet des améliorations seraient:

- Elargir l'approche à d'autres types de contraintes temporelles comme par exemple, le deadline, période, et contrainte du temps physique.

- Etendre le champ de vérification par l'ajout des éléments de control d'accès capable d'être transformer en langage Fiacre.

D'autre perspectives à ce travail concernent plusieurs axes à prendre en compte dans différentes directions. Ils concernent la prise en compte des politiques de contrôle d'accès qui peuvent être complexes, être définies de manière dynamique, porter sur des niveaux différents des architectures. Ces politiques doivent reposer sur une formalisation indispensable à partir de laquelle, un processus de génération de modèle d'architecture peut être envisagé.

C'est perspectives concernent les aspects de formalisation de la politique, des exigences, des mécanismes de sécurité candidats, mais aussi le choix des implantations. Nous en avons testé certains mais il serait judicieux d'entamer une approche comparative plus structurée sur les bases de l'approche que nous proposons. Des critères seraient à identifier pour mener cette comparaison. Ils concerneraient, à première vue, la robustesse aux attaques et la performance des architectures obtenues.

Un autre élément intéressant pour étendre nos travaux serait d'élaborer un langage textuel avec une sémantique formelle, qui permettrait de générer le modèle graphique que nous avons étudié dans cette thèse. L'objectif serait d'exprimer dans ce langage textuel des politiques qui soient des interprétations exactes du texte initial de la politique. Dans ce cas, l'utilisateur n'a plus à interpréter la politique avant de la modéliser. Il réécrit simplement cette politique dans un autre langage textuel et obtient la formalisation demandée.

Abréviations

ABAC Contrôle d'accès basé sur les attributs
TCSEC Trusted Computer System Evaluation Criteria
AC contrôle d'accès.
ACL listes de contrôle d'accès
ACM Matrice de contrôle d'accès
BLP Modèle Bell-LaPadula
CDL Context Description Language.
CTL Computation tree logic.
DAC Discretionary Access Control
DoD Le Département de la Défense des États-Unis
E-RBAC Emergency role-based access control
E-UACML
E-UACML Emergency Unified Access Control Modeling Language
HRU Modèle de Harrison-Ruzzo-Ullman
IDM Ingénierie Dirigée par les Modèles
LTL Linear temporal logic.
MAC Mandatory Access Control
MDA L'architecture dirigée par les modèles
MDS Model Driven Security
MIC le contrôle d'intégrité obligatoire
MLS Sécurité multi niveau
MLS Multi-Level Security
Nc niveaux de confiance
NRD non-répudiation de remise.
NRR non-répudiation de la réception
NRS non-répudiation de la soumission
OBP Observer-Based Prover.
PA d'autorisation
PoLP principe du moindre privilège
RBAC Role-Based Access Control
SoD La séparation des tâches,
STRBAC Spatial Temporal Role Based Access Control
UA D'utilisateur
UACML Unified Access Control Modeling Language
UML Unified Modeling Language

7 Référence

[1] P. Samarati et S. C. de Vimercati, « Access control: Policies, models, and mechanisms », in *International School on Foundations of Security Analysis and Design*, 2000, p. 137-196.

[2] R. S. Sandhu et P. Samarati, « Access control: principle and practice », *IEEE Commun. Mag.*, vol. 32, nº 9, p. 40-48, 1994.

[3] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, et C. E. Youman, « Role-based access control models », *Computer*, vol. 29, nº 2, p. 38-47, 1996.

[4] « MDA | Object Management Group ». http://www.omg.org/mda/ (consulté le 14 mars 2018).

[5] T. Lodderstedt, D. Basin, et J. Doser, « SecureUML: A UML-based modeling language for model-driven security », in *International Conference on the Unified Modeling Language*, 2002, p. 426-441.

[6] J. Jürjens, « UMLsec: Extending UML for secure systems development », in *International Conference on The Unified Modeling Language*, 2002, p. 412-425.

[7] N. Slimani, H. Khambhammettu, K. Adi, et L. Logrippo, « UACML: Unified access control modeling language », in *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, 2011, p. 1-8.

[8] F. Nazerian, H. Motameni, et H. Nematzadeh, « Emergency role-based access control (E-RBAC) and analysis of model specifications with alloy », *J. Inf. Secur. Appl.*, vol. 45, p. 131-142, avr. 2019, doi: 10.1016/j.jisa.2019.01.008.

[9] M. W. Krakallah et S. Nait-Bahloul, « Formalisation of access control based on observers automata », *Int. J. Syst. Syst. Eng.*, vol. 11, nº 2, p. 170-197, janv. 2021, doi: 10.1504/IJSSE.2021.116045.

[10] B. Berthomieu *et al.*, « Fiacre: an Intermediate Language for Model Verification in the Topcased Environment », Toulouse, France, janv. 2008. [En ligne]. Disponible sur: https://hal.inria.fr/inria-00262442

[11] M. W. Krakallah et S. Nait-Bahloul, « Methodological Sketch of Observer-Based Access Control », in *Proceedings of the 2nd international Conference on Big Data, Cloud and Applications*, 2017, ACM, p.62.

[12] R. J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley Publishing, 2008.

[13] G. Saunders, M. Hitchens, et V. Varadharajan, « Role-Based Access Control and the Access Control Matrix », in *Information and Communications Security*, Berlin, Heidelberg, 2003, p. 145-157. doi: 10.1007/978-3-540-39927-8_14.

[14] N. C. S. Center, « A guide to understanding audit in trusted systems », Technical Report NCSC-TG-001, National Computer Security Center, 1988.

[15] F. David et D. Ferraiolo, « Richard Kuhn », 1992.

[16] J. Daemen et V. Rijmen, « The design of Rijndael: AES-the advanced encryption standard Springer Science & Business Media », 2013.

[17] R. L. Rivest, A. Shamir, et L. Adleman, « A method for obtaining digital signatures and public-key cryptosystems », *Commun. ACM*, vol. 21, nº 2, p. 120-126, 1978.

[18] P. C. Kocher, « Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems », in *Annual International Cryptology Conference*, 1996, p. 104-113.

[19] D. M. Ritchie et K. Thompson, « The UNIX time-sharing system », *Bell Syst. Tech. J.*, vol. 57, nº 6, p. 1905-1929, 1978.

[20] L. Lamport, « Password authentication with insecure communication », *Commun. ACM*, vol. 24, nº 11, p. 770-772, 1981.

[21] S. S. Hamilton, M. C. Carlisle, et J. A. Hamilton, « A global look at authentication », in *2007 IEEE SMC Information Assurance and Security Workshop*, 2007, p. 1-8.
[22] J. Wayman, A. Jain, D. Maltoni, et D. Maio, « An introduction to biometric authentication systems », in *Biometric Systems*, Springer, 2005, p. 1-20.
[23] D. Weinshall, « Cognitive authentication schemes safe against spyware », in *2006 IEEE Symposium on Security and Privacy (S&P'06)*, 2006, p. 6-pp.
[24] C.-H. Jiang, S. Shieh, et J.-C. Liu, « Keystroke statistical learning model for web authentication », in *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, 2007, p. 359-361.
[25] S. Kremer, O. Markowitch, et J. Zhou, « An intensive survey of fair non-repudiation protocols », *Comput. Commun.*, vol. 25, nº 17, p. 1606-1621, 2002.
[26] J. Zhou et D. Gollman, « A fair non-repudiation protocol », in *Proceedings 1996 IEEE Symposium on Security and Privacy*, 1996, p. 55-61.
[27] L. Brandeis et S. Warren, « The right to privacy », *Harv. Law Rev.*, vol. 4, nº 5, p. 193-220, 1890.
[28] R. Clarke, « Introduction to Dataveillance and Information Privacy, and Definitions and Terms.[Online] Available: http://www. anu. edu. au/people/Roger. Clarke/DV », *Intro Html Priv*, 1999.
[29] M. Langheinrich, « Privacy by design—principles of privacy-aware ubiquitous systems », in *International conference on Ubiquitous Computing*, 2001, p. 273-291.
[30] J. Rosen, « The right to be forgotten », *Stan Rev Online*, vol. 64, p. 88, 2011.
[31] J. H. Saltzer, « Protection and the control of information sharing in Multics », *Commun. ACM*, vol. 17, nº 7, p. 388-402, 1974.
[32] V. D. Gligor, S. I. Gavrila, et D. Ferraiolo, « On the formal definition of separation-of-duty policies and their composition », in *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No. 98CB36186)*, 1998, p. 172-183.
[33] L. Qiu, Y. Zhang, F. Wang, M. Kyung, et H. R. Mahajan, « Trusted computer system evaluation criteria », 1985.
[34] B. W. Lampson, « Protection », *ACM SIGOPS Oper. Syst. Rev.*, vol. 8, nº 1, p. 18-24, 1974.
[35] M. A. Harrison, W. L. Ruzzo, et J. D. Ullman, « Protection in operating systems », *Commun. ACM*, vol. 19, nº 8, p. 461-471, 1976.
[36] J. H. Saltzer et M. D. Schroeder, « The protection of information in computer systems », *Proc. IEEE*, vol. 63, nº 9, p. 1278-1308, 1975.
[37] G. S. Graham et P. J. Denning, « Protection: principles and practice », in *Proceedings of the May 16-18, 1972, spring joint computer conference*, 1971, p. 417-429.
[38] J. B. Dennis et E. C. Van Horn, « Programming semantics for multiprogrammed computations », *Commun. ACM*, vol. 9, nº 3, p. 143-155, 1966.
[39] M. Selimi et F. Freitag, « Tahoe-lafs distributed storage service in community network clouds », in *2014 IEEE Fourth International Conference on Big Data and Cloud Computing*, 2014, p. 17-24.
[40] M. S. Miller, M. Samuel, B. Laurie, I. Awad, et M. Stay, « Safe active content in sanitized JavaScript », *Google Inc Tech Rep*, 2008.
[41] A. Mettler, D. A. Wagner, et T. Close, « Joe-E: A Security-Oriented Subset of Java. », in *NDSS*, 2010, vol. 10, p. 357-374.
[42] M. S. Miller, K.-P. Yee, et J. Shapiro, « Capability myths demolished », Technical Report SRL2003-02, Johns Hopkins University Systems Research …, 2003.
[43] E. J. Coyne, « Role engineering », in *Proceedings of the first ACM Workshop on Role-based access control*, 1996, p. 4-es.

[44] S. Smalley, « Which operating system access control technique will provide the greatest overall benefit to users? », in *Proceedings of the sixth ACM symposium on Access control models and technologies*, 2001, p. 147-148.
[45] N. NIST, « Interagency Report 7316 Access to Access Contol Systems, Vincent C », *Hu Al*, p. 18, 2006.
[46] D. McCullough, « Specifications for multi-level security and a hook-up », in *1987 IEEE Symposium on Security and Privacy*, 1987, p. 161-161.
[47] D. E. Bell et L. J. LaPadula, « Secure computer systems: Mathematical foundations », MITRE CORP BEDFORD MA, 1973.
[48] K. J. Biba, « Integrity considerations for secure computer systems », MITRE CORP BEDFORD MA, 1977.
[49] J. McLean, « A comment on the 'basic security theorem'of Bell and LaPadula », *Inf. Process. Lett.*, vol. 20, nº 2, p. 67-70, 1985.
[50] M. Bishop, « Introduction to computer security », 2005.
[51] D. F. Brewer et M. J. Nash, « The Chinese Wall Security Policy. », in *IEEE symposium on security and privacy*, 1989, vol. 1989, p. 206.
[52] R. Ausanka-Crues et H. Mudd, « Methods for Access Control : Advances and Limitations », 2006. https://www.semanticscholar.org/paper/Methods-for-Access-Control-%3A-Advances-and-Ausanka-Crues-Mudd/6192f0308dc8d7782b55a0557dfb66f323638853 (consulté le 4 novembre 2021).
[53] V. C. Hu, D. R. Kuhn, et T. Xie, « Property verification for access control models via model-checking », 2008.
[54] A. Gouglidis, I. Mavridis, et V. C. Hu, « Security policy verification for multi-domains in cloud systems », *Int. J. Inf. Secur.*, vol. 13, nº 2, p. 97-111, 2014.
[55] X. Jin, R. Krishnan, et R. Sandhu, « A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC », in *Data and Applications Security and Privacy XXVI*, Berlin, Heidelberg, 2012, p. 41-55. doi: 10.1007/978-3-642-31540-4_4.
[56] V. Hu *et al.*, « Guide to attribute based access control (ABAC) definition and considerations », *Natl. Inst. Stand. Technol. Spec. Publ.*, p. 162-800, janv. 2014.
[57] L. Wang, D. Wijesekera, et S. Jajodia, « A logic-based framework for attribute based access control », in *Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, New York, NY, USA, oct. 2004, p. 45-55. doi: 10.1145/1029133.1029140.
[58] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, et T. Freeman, « Attribute Based Access Control for Grid Computing ».
[59] A. Corradi, R. Montanari, et D. Tibaldi, « Context-Based Access Control Management in Ubiquitous Environments », in *Proceedings of the Network Computing and Applications, Third IEEE International Symposium*, USA, août 2004, p. 253-260.
[60] D. Kulkarni et 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*, New York, NY, USA, juin 2008, p. 113-122. doi: 10.1145/1377836.1377854.
[61] E. Bertino, P. A. Bonatti, et E. Ferrari, « TRBAC: A Temporal Role-based Access Control Model », *ACM Trans Inf Syst Secur*, vol. 4, nº 3, p. 191-233, août 2001, doi: 10.1145/501978.501979.
[62] M. Kumar et R. E. Newman, « STRBAC-An approach towards spatio-temporal role-based access control. », *Commun. Netw. Inf. Secur.*, vol. 155, 2006.
[63] G. Zhao, D. Chadwick, et S. Otenko, « Obligations for role based access control », in *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, 2007, vol. 1, p. 424-431.

[64] A. J. W. Mayer, « The architecture of the Burroughs B5000: 20 years later and still ahead of the times? », *ACM SIGARCH Comput. Archit. News*, vol. 10, nº 4, p. 3-10, juin 1982, doi: 10.1145/641542.641543.
[65] R. S. Fabry, « Capability-based addressing », *Commun. ACM*, vol. 17, nº 7, p. 403-412, juill. 1974, doi: 10.1145/361011.361070.
[66] M. Houdek, F. Soltis, et R. L. Hoffman, « IBM System/38 support for capability-based addressing », 1981.
[67] T. Rentsch, « Object oriented programming », *ACM SIGPLAN Not.*, vol. 17, nº 9, p. 51-57, sept. 1982, doi: 10.1145/947955.947961.
[68] H. M. Levy, *Capability-Based Computer Systems*. Digital Press, 2014.
[69] T. Jaeger, R. Sailer, et X. Zhang, « Analyzing integrity protection in the SELinux example policy », in *Proceedings of the 12th conference on USENIX Security Symposium - Volume 12*, USA, août 2003, p. 5.
[70] M. Bauer, « Paranoid penguin: an introduction to Novell AppArmor », *undefined*, 2006, Consulté le: 4 novembre 2021. [En ligne]. Disponible sur: https://www.semanticscholar.org/paper/Paranoid-penguin%3A-an-introduction-to-Novell-Bauer/ffb585fa9fcfcf80ada7f8a5091755718c729f36
[71] C. Schaufler, « Smack in embedded computing », janv. 2008.
[72] lastnameholiu, « Mandatory Integrity Control - Win32 apps ». https://docs.microsoft.com/en-us/windows/win32/secauthz/mandatory-integrity-control (consulté le 4 novembre 2021).
[73] C. Vance, T. C. Miller, R. Dekelbaum, et A. Reisse, « Security-Enhanced Darwin: Porting SELinux to Mac OS X », *undefined*, 2007, Consulté le: 4 novembre 2021. [En ligne]. Disponible sur: https://www.semanticscholar.org/paper/Security-Enhanced-Darwin%3A-Porting-SELinux-to-Mac-OS-Vance-Miller/7e4cf65628a3936c1c09fcff80e7fadf46dbee1a
[74] Z. Wilcox-O'Hearn et B. Warner, « Tahoe: the least-authority filesystem », in *Proceedings of the 4th ACM international workshop on Storage security and survivability*, New York, NY, USA, oct. 2008, p. 21-26. doi: 10.1145/1456469.1456474.
[75] J. S. Shapiro, J. M. Smith, et D. J. Farber, « EROS: a fast capability system », *ACM SIGOPS Oper. Syst. Rev.*, vol. 33, nº 5, p. 170-185, déc. 1999, doi: 10.1145/319344.319163.
[76] J. S. Shapiro, E. Northup, M. S. Doerrie, S. Sridhar, N. H. Walfield, et M. Brinkmann, « Coyotos microkernel specification », *EROS Group LLC 05 Ed.*, p. 24, 2007.
[77] J. Joshi, A. Ghafoor, W. G. Aref, et E. H. Spafford, « Digital government security infrastructure design challenges », *Computer*, vol. 34, nº 2, p. 66-72, févr. 2001, doi: 10.1109/2.901169.
[78] D. Basin, J. Doser, et T. Lodderstedt, « Model driven security for process-oriented systems », in *Proceedings of the eighth ACM symposium on Access control models and technologies*, 2003, p. 100-109.
[79] S. Chehida, « Approche de spécification et validation formelles de politiques RBAC au niveau des processus métiers », *Approch. Formelles Dans L'Assistance Au Dev. Logiciels*, p. 27, 2016.
[80] O. M. Group, « Unified modeling language », *Httpwww Uml Org*, 2001.
[81] N. Russell, W. Van der Aalst, A. Ter Hofstede, et P. Wohed, « On the suitability of UML 2.0 activity diagrams for business process modelling », *Concept. Model. 2006 Proc. APCCM2006*, p. 95-104, 2006.
[82] A. Rodríguez, E. Fernández-Medina, J. Trujillo, et M. Piattini, « Secure business process model specification through a UML 2.0 activity diagram profile », *Decis. Support Syst.*, vol. 51, nº 3, p. 446-465, 2011.

[83] S. Nurcan, G. Grosz, et C. Souveyet, « Describing business processes with a guided use case approach », in *International Conference on Advanced Information Systems Engineering*, 1998, p. 339-362.
[84] D. Lubke, K. Schneider, et M. Weidlich, « Visualizing use case sets as BPMN processes », in *2008 Requirements Engineering Visualization*, 2008, p. 21-25.
[85] K. Alghathbar et D. Wijesekera, « authUML: A Three-phased Framework to Analyze Access Control Specifications in Use Cases », in *Proceedings of the 2003 ACM Workshop on Formal Methods in Security Engineering*, New York, NY, USA, 2003, p. 77-86. doi: 10.1145/1035429.1035438.
[86] S. Barker, « The next 700 access control models or a unifying meta-model? », in *Proceedings of the 14th ACM symposium on Access control models and technologies*, 2009, p. 187-196.
[87] « home | OBP2 ». http://www.obpcdl.org/ (consulté le 5 avril 2020).
[88] P. Dhaussy et J.-C. Roger, « Cdl (context description language): syntax and semantics », *Rapp. Tech. ENSTA-Bretagne*, vol. 37, 2011.
[89] A. Pnueli, « The temporal logic of programs », *18th Annu. Symp. Found. Comput. Sci. Sfcs 1977*, 1977, doi: 10.1109/SFCS.1977.32.
[90] A. Pnueli, « The temporal semantics of concurrent programs », *Theor. Comput. Sci.*, vol. 13, n° 1, Art. n° 1, 1981.
[91] E. M. Clarke et E. A. Emerson, « Design and synthesis of synchronization skeletons using branching time temporal logic », in *Logics of Programs*, Berlin, Heidelberg, 1982, p. 52-71. doi: 10.1007/BFb0025774.
[92] E. Emerson et J. Y. Halpern, « Decision Procedures and Expressiveness in the Temporal Logic of Branching Time », *J Comput Syst Sci*, 1985, doi: 10.1016/0022-0000(85)90001-7.
[93] S. Bardin, « Introduction au Model Checking », *CEA LIST Saf. Softw. Lab.*, 2008.
[94] G. J. Holzmann, « The model-checker SPIN », *IEEE Trans. Softw. Eng.*, vol. 23, n° 5, p. 279-295, mai 1997, doi: 10.1109/32.588521.
[95] « NuSMV home page ». https://nusmv.fbk.eu/ (consulté le 8 juin 2022).
[96] S. Santiago, S. Escobar, C. Meadows, et J. Meseguer, « A formal definition of protocol indistinguishability and its verification using Maude-NPA », in *International Workshop on Security and Trust Management*, 2014, p. 162-177.
[97] G. Behrmann, A. David, et K. G. Larsen, « A Tutorial on <Emphasis Type="SmallCaps">Uppaal</Emphasis> », in *Formal Methods for the Design of Real-Time Systems*, Springer, Berlin, Heidelberg, 2004, p. 200-236. doi: 10.1007/978-3-540-30080-9_7.
[98] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, et Y. Zhu, « Bounded model-checking », 2003.
[99] D. Jackson, « Alloy: a lightweight object modelling notation », *ACM Trans. Softw. Eng. Methodol. TOSEM*, vol. 11, n° 2, p. 256-290, 2002.
[100] B. Shafiq, J. B. Joshi, et A. Ghafoor, « Petri-net based modeling for verification of RBAC policies », *CERIAS Purdue Univ. Tech Rep*, vol. 33, p. 2002, 2002.
[101] H. Rakkay, « Approches formelles pour la modélisation et la vérification du contrôle d'accès et des contraintes temporelles dans les systèmes d'information », PhD Thesis, École Polytechnique de Montréal, 2009.
[102] T. Ahmed et A. R. Tripathi, « Static Verification of Security Requirements in Role Based CSCW Systems », in *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, New York, NY, USA, 2003, p. 196-203. doi: 10.1145/775412.775438.

[103] F. Hansen et V. Oleshchuk, « Conformance Checking of RBAC Policy and its Implementation », in *Information Security Practice and Experience*, 2005, p. 144-155.
[104] F. B. Schneider, « Enforceable security policies », *ACM Trans. Inf. Syst. Secur. TISSEC*, vol. 3, nº 1, p. 30-50, 2000.
[105] S. Mondal et S. Sural, « A verification framework for temporal RBAC with Role Hierarchy (Short Paper) », in *International Conference on Information Systems Security*, 2008, p. 140-147.
[106] M. E. Jiague, « Approches formelles de mise en oeuvre de politiques de contrôle d'accès pour des applications basées sur une architecture orientée services », PhD Thesis, Université Paris-Est, 2012.
[107] E. Geepalla, B. Bordbar, et K. Okano, « Verification of spatio-temporal role based access control using timed automata », in *Networked Embedded Systems for Every Application (NESEA), 2012 IEEE 3rd International Conference on*, 2012, p. 1-6.
[108] G. Yunchuan, Y. Lihua, et L. Chao, « Automatically verifying STRAC policy », in *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, 2014, p. 141-142.
[109] Y. Aoki et S. Matsuura, « Verifying security requirements using model-checking technique for UML-based requirements specification », in *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET)*, août 2014, p. 18-25. doi: 10.1109/RET.2014.6908674.
[110] M. Cheminod, L. Durante, L. Seno, et A. Valenzano, « Semiautomated Verification of Access Control Implementation in Industrial Networked Systems », *IEEE Trans. Ind. Inform.*, vol. 11, nº 6, p. 1388-1399, déc. 2015, doi: 10.1109/TII.2015.2489181.
[111] P. Vasilikos, F. Nielson, et H. R. Nielson, « Time Dependent Policy-Based Access Control », in *24th International Symposium on Temporal Representation and Reasoning (TIME 2017)*, Dagstuhl, Germany, 2017, vol. 90, p. 21:1-21:18. doi: 10.4230/LIPIcs.TIME.2017.21.
[112] P. Dhaussy, J. C. Roger, et F. Boniol, « Reducing State Explosion with Context Modeling for Model-Checking », in *2011 IEEE 13th International Symposium on High-Assurance Systems Engineering*, nov. 2011, p. 130-137. doi: 10.1109/HASE.2011.24.
[113] J. Whittle, « Specifying Precise Use Cases with Use Case Charts », in *Satellite Events at the MoDELS 2005 Conference*, 2006, p. 290-301.
[114] B. Westphal, « lsc. sty–Typesetting Live Sequence Charts », 2006.
[115] H. Sotharith, « Transformation de modèles UML vers des programmes Fiacre », *ENSTA Bretagnedumas-00725248*.
[116] B. H. Cheng et J. M. Atlee, « Research directions in requirements engineering », *Future Softw. Eng. FOSE07*, p. 285-303, 2007.
[117] A. Van Lamsweerde, *Requirements engineering: From system goals to UML models to software*, vol. 10. Chichester, UK: John Wiley & Sons, 2009.
[118] T. Yue, L. C. Briand, et Y. Labiche, « A systematic review of transformation approaches between user requirements and analysis models », *Requir. Eng.*, vol. 16, nº 2, p. 75-99, 2011.
[119] « IEEE Standard for Property Specification Language (PSL) », *IEEE Std 1850-2010 Revis. IEEE Std 1850-2005*, p. 1-182, avr. 2010, doi: 10.1109/IEEESTD.2010.5446004.
[120] J. B. Joshi, E. Bertino, et A. Ghafoor, « Temporal hierarchies and inheritance semantics for GTRBAC », in *Proceedings of the seventh ACM symposium on Access control models and technologies*, 2002, p. 74-83.
[121] P. Epstein et R. Sandhu, « Towards a UML based approach to role engineering », in *Proceedings of the fourth ACM workshop on Role-based access control*, 1999, p. 135-143.

[122] M. E. Shin et G.-J. Ahn, « UML-based representation of role-based access control », in *Proceedings IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000)*, 2000, p. 195-200.
[123] D. Basin, J. Doser, et T. Lodderstedt, « Model driven security: From UML models to access control infrastructures », *ACM Trans. Softw. Eng. Methodol. TOSEM*, vol. 15, nº 1, p. 39-91, 2006.
[124] J. A. Pavlich-Mariscal, S. A. Demurjian, et L. D. Michel, « A framework of composable access control features: Preserving separation of access control concerns from models to code », *Comput. Secur.*, vol. 29, nº 3, p. 350-379, 2010.
[125] I. Ray, N. Li, D.-K. Kim, et R. France, « Using parameterized UML to specify and compose access control models », in *Working Conference on Integrity and Internal Control in Information Systems*, 2003, p. 49-65.
[126] T. Doan, S. Demurjian, T. C. Ting, et A. Ketterl, « MAC and UML for secure software design », in *Proceedings of the 2004 ACM workshop on Formal Methods in Security Engineering*, 2004, p. 75-85.
[127] B. Maboudou, « Un environnement pour la modélisation des systèmes de contrôle d'accès », masters, Université du Québec en Outaouais, Gatineau, 2015. Consulté le: 17 juillet 2018. [En ligne]. Disponible sur: http://di.uqo.ca/724/
[128] N. Menad, P. Dhaussy, Z. Drey, et R. Mekki, « Towards a transformation approach of timed uml marte specifications for observer-based formal verification », *Comput. Inform.*, vol. 35, nº 2, p. 338-368, 2016.
[129] F. Mallet, « Clock constraint specification language: specifying clock constraints with UML/MARTE », *Innov. Syst. Softw. Eng.*, vol. 4, nº 3, p. 309-314, 2008.
[130] C. André, « Syntax and Semantics of the Clock Constraint Specification Language (CCSL) », INRIA, report, 2009. Consulté le: 13 février 2018. [En ligne]. Disponible sur: https://hal.inria.fr/inria-00384077/document
[131] F. Jouault, C. Teodorov, J. Delatour, L. Le Roux, et P. Dhaussy, « Transformation de modèles UML vers Fiacre, via les langages intermédiaires tUML et ABCD », *Génie Logiciel*, vol. 109, p. xx, juin 2014.
[132] S. Mondal, S. Sural, et V. Atluri, « Security analysis of GTRBAC and its variants using model-checking », *Comput. Secur.*, vol. 30, nº 2, p. 128-147, mars 2011, doi: 10.1016/j.cose.2010.09.002.
[133] P. Dhaussy, F. Boniol, J.-C. Roger, et L. Leroux, « Improving Model-Checking with Context Modelling », *Adv Soft Eng*, vol. 2012, p. 9:9-9:9, janv. 2012, doi: 10.1155/2012/547157.

8 Annexe

Traduction automate Subject1 en code Fiacre

```
516  //================================================
517  //================================================
518  //================================================
519  // utilisateur S1 active le role TD et le groupe CS
520  //================================================
521  //================================================
522  //================================================
523
524  process S1 ( &activ_s1        : read write bool,
525               &activ_TD        : read write bool,
526               &activ_CS        : read write bool,
527               &fifo_fromEnv    : read write t_fifo_req,
528               &fifo_toRole_TD  : read write t_fifo_req,
529               &fifo_toGroup_CS : read write t_fifo_req)
530   is
531   states Stop, Start, TraiteReq, SendToRol,SendToGroupe ,SendRet, Error
532   init to Stop
533   from Stop activ_s1 := False to Stop
534   from Stop /*Active Rrole TD */   activ_s1 := True; activ_TD := True; to Start
535   from Stop /*Active Goupe CS*/   activ_s1 := True; activ_CS := True; to Start
536   from Start /* Desactive Role TD*/ activ_s1 := False; activ_TD := False; to Stop
537   from Start /* Desactive Groupe CS*/ activ_s1 := False; activ_CS := False; to Stop
538
539   from Start /* ENV Send Req to S1*/
540   case (empty fifo_fromEnv) of false -> null end case;
541   reqFromEnv := first fifo_fromEnv;
542   fifo_fromEnv := dequeue (fifo_fromEnv); to TraiteReq
543   from TraiteReq //------- send req to Role TD -------
544   if (full fifo_toRole_TD ) then loop end;
545   fifo_toRole_TD := enqueue (fifo_toRole_TD, reqFromEnv); to SendToRol
546   from TraiteReq //------- send req to Group -------
547   if (full fifo_toGroup_CS ) then loop end;
548   fifo_toGroup_CS := enqueue (fifo_toGroup_CS, reqFromEnv); to
549   SendToGroup
550   from TraiteReq ret := hasRight_S1 (req) to Error
551   from Error //------- send ret to ENV -------
552   ret := RET_NON to SendRet
553   from Error loop
554   end for
```

Traduction de l’automate Subject 2 en code Fiacre

```
//=============================================
//=============================================
//=============================================
// utilisateur S2 active le role TD et le groupe CS
//=============================================
//=============================================
//=============================================

process S2 ( &activ_s2        : read write bool,
             &activ_TD        : read write bool,
             &activ_CS        : read write bool,
             &fifo_fromEnv    : read write t_fifo_req,
             &fifo_toRole_TD  : read write t_fifo_req,
             &fifo_toGroup_CS : read write t_fifo_req)
is
states Stop, Start, TraiteReq, SendToRol,SendToGroupe ,SendRet, Error
init to Stop
from Stop activ_s2 := False to Stop
from Stop /*Active Rrole TD */   activ_s2 := True; activ_TD := True; to Start
from Stop /*Active Goupe CS*/   activ_s2 := True; activ_CS := True; to Start
from Start /* Desactive Role TD*/ activ_s2 := False; activ_TD := False; to Stop
from Start /* Desactive Groupe CS*/ activ_s2 := False; activ_CS := False; to Stop

from Start /* ENV Send Req to S2*/
case (empty fifo_fromEnv) of false -> null end case;
reqFromEnv := first fifo_fromEnv;
fifo_fromEnv := dequeue (fifo_fromEnv); to TraiteReq
from TraiteReq //------- send req to Role TD -------
if (full fifo_toRole_TD ) then loop end;
fifo_toRole_TD := enqueue (fifo_toRole_TD, reqFromEnv); to SendToRol
from TraiteReq //------- send req to Group -------
if (full fifo_toGroup_CS ) then loop end;
fifo_toGroup_CS := enqueue (fifo_toGroup_CS, reqFromEnv); to
SendToGroup
from TraiteReq ret := hasRight_S2 (req) to Error
from Error //------- send ret to ENV -------
ret := RET_NON to SendRet
from Error loop
end for
```

Traduction de l’automate Subject 3 en code Fiacre

```
599  //=============================================
600  //=============================================
601  //=============================================
602  //utilisateur S3 active le role Nur et le groupe CS
603  //=============================================
604  //=============================================
605  //=============================================
606
607  process S3 ( &activ_s3        : read write bool,
608               &activ_Nur       : read write bool,
609               &activ_CS        : read write bool,
610               &fifo_fromEnv    : read write t_fifo_req,
611               &fifo_toRole_Nur  : read write t_fifo_req,
612               &fifo_toGroup_CS : read write t_fifo_req)
613  is
614  states Stop, Start, TraiteReq, SendToRol,SendToGroupe ,SendRet, Error
615  init to Stop
616  from Stop activ_s3 := False to Stop
617  from Stop /*Active Rrole Nur */     activ_s3 := True;  activ_Nur := True; to Start
618  from Stop /*Active Goupe CS*/       activ_s3 := True;  activ_CS := True; to Start
619  from Start /* Desactive Role Nur*/  activ_s3 := False; activ_Nur := False; to Stop
620  from Start /* Desactive Groupe CS*/ activ_s3 := False; activ_CS := False; to Stop
621
622  from Start /* ENV Send Req to S3*/
623  case (empty fifo_fromEnv) of false -> null end case;
624  reqFromEnv := first fifo_fromEnv;
625  fifo_fromEnv := dequeue (fifo_fromEnv); to TraiteReq
626  from TraiteReq //------- send req to Role Nur -------
627  if (full fifo_toRole_Nur ) then loop end;
628  fifo_toRole_Nur := enqueue (fifo_toRole_Nur, reqFromEnv); to SendToRol
629  from TraiteReq //------- send req to Group -------
630  if (full fifo_toGroup_CS ) then loop end;
631  fifo_toGroup_CS := enqueue (fifo_toGroup_CS, reqFromEnv); to
632  SendToGroup
633  from TraiteReq ret := hasRight_S3 (req) to Error
634  from Error //------- send ret to ENV -------
635  ret := RET_NON to SendRet
636  from Error loop
637  end for
```

Traduction de l’automate Subject 4 en code Fiacre

```
642  //================================================
643  //================================================
644  //================================================
645  // utilisateur S4 active le role Doc et le groupe CS
646  //================================================
647  //================================================
648  //================================================
649
650  process S4 ( &activ_s4        : read write bool,
651               &activ_Doc       : read write bool,
652               &activ_CS        : read write bool,
653               &fifo_fromEnv    : read write t_fifo_req,
654               &fifo_toRole_Doc  : read write t_fifo_req,
655               &fifo_toGroup_CS : read write t_fifo_req)
656   is
657   states Stop, Start, TraiteReq, SendToRol,SendToGroupe ,SendRet, Error
658   init to Stop
659   from Stop activ_s4 := False to Stop
660   from Stop /*Active Rrole Doc */   activ_s4 := True; activ_Doc := True; to Start
661   from Stop /*Active Goupe CS*/   activ_s4 := True; activ_CS := True; to Start
662   from Start /* Desactive Role Doc*/ activ_s4 := False; activ_Doc := False; to Stop
663   from Start /* Desactive Groupe CS*/ activ_s4 := False; activ_CS := False; to Stop
664
665   from Start /* ENV Send Req to S4*/
666   case (empty fifo_fromEnv) of false -> null end case;
667   reqFromEnv := first fifo_fromEnv;
668   fifo_fromEnv := dequeue (fifo_fromEnv); to TraiteReq
669   from TraiteReq //------- send req to Role Doc -------
670   if (full fifo_toRole_Doc ) then loop end;
671   fifo_toRole_Doc := enqueue (fifo_toRole_Doc, reqFromEnv); to SendToRol
672   from TraiteReq //------- send req to Group -------
673   if (full fifo_toGroup_CS ) then loop end;
674   fifo_toGroup_CS := enqueue (fifo_toGroup_CS, reqFromEnv); to
675   SendToGroup
676   from TraiteReq ret := hasRight_S4 (req) to Error
677   from Error //------- send ret to ENV -------
678   ret := RET_NON to SendRet
679   from Error loop
680   end for
```

Traduction de l’automate Subject 5 en code Fiacre

```
684  //=============================================
685  //=============================================
686  //=============================================
687  // utilisateur S5 active le role Dir et le groupe AS
688  //=============================================
689  //=============================================
690  //=============================================
691
692  process S5 ( &activ_s5        : read write bool,
693               &activ_Dir       : read write bool,
694               &activ_AS        : read write bool,
695               &fifo_fromEnv    : read write t_fifo_req,
696               &fifo_toRole_Dir  : read write t_fifo_req,
697               &fifo_toGroup_AS : read write t_fifo_req)
698  is
699  states Stop, Start, TraiteReq, SendToRol,SendToGroupe ,SendRet, Error
700  init to Stop
701  from Stop activ_s5 := False to Stop
702  from Stop /*Active Rrole Dir */   activ_s5 := True; activ_Dir := True; to Start
703  from Stop /*Active Goupe AS*/   activ_s5 := True; activ_AS := True; to Start
704  from Start /* Desactive Role Dir*/ activ_s5 := False; activ_Dir := False; to Stop
705  from Start /* Desactive Groupe AS*/ activ_s5 := False; activ_AS := False; to Stop
706
707  from Start /* ENV Send Req to S5*/
708  case (empty fifo_fromEnv) of false -> null end case;
709  reqFromEnv := first fifo_fromEnv;
710  fifo_fromEnv := dequeue (fifo_fromEnv); to TraiteReq
711  from TraiteReq //------- send req to Role Dir -------
712  if (full fifo_toRole_Dir ) then loop end;
713  fifo_toRole_Dir := enqueue (fifo_toRole_Dir, reqFromEnv); to SendToRol
714  from TraiteReq //------- send req to Group -------
715  if (full fifo_toGroup_AS ) then loop end;
716  fifo_toGroup_AS := enqueue (fifo_toGroup_AS, reqFromEnv); to
717  SendToGroup
718  from TraiteReq ret := hasRight_S5 (req) to Error
719  from Error //------- send ret to ENV -------
720  ret := RET_NON to SendRet
721  from Error loop
722  end for
```

Traduction Automate Role (Doc) en code Fiacre

```
725  //=================================================
726  //=================================================
727  //=================================================
728  //  le role DOC qui active la resource medical_file, prescription et nurse_report
729  //=================================================
730  //=================================================
731  //=================================================
732
733
734  process DOC (  &activ_DOC : read write bool,
735                 &activ_medical_file : read write bool,
736                 &activ_prescription : read write bool,
737                 &activ_nurse_report     : read write bool,
738                 &fifo_fromS4 : read write t_fifo_req,
739                 &fifo_toMedical_file : read write t_fifo_ret,
740                 &fifo_toPrescriptio : read write t_fifo_ret
741                 &fifo_toNurse_report : read write t_fifo_ret )
742  is
743   states Stop, Start, WaitSubject, SendToResou,SendRet, Error
744  init to Stop
745   from Stop
746         if (activ_DOC = True) then activ_medical_file :=True;
747                                    activ_prescription :=True;
748                                    activ_nurse_report      :=True;
749   to Start end
750    from Start
751   if (activ_DOC = False) then activ_medical_file :=False;
752                               activ_prescription :=False;
753                               activ_nurse_report      :=False;
754
755   to Stop end
756   from Start /* S4 Send Req to DOC */
757       case (empty fifo_fromS4) of false -> null end case;
758              reqFromS4 := first fifo_fromS4;
759              fifo_fromS4 := dequeue (fifo_fromS4);
760   to WaitSubject
761   from WaitSubject //------- send req to medical_file -------
762           if (full fifo_toMedical_file ) then loop end;
763              fifo_toMedical_file := enqueue (fifo_toMedical_file, reqFromS4);
764  to SendToResou
765   from WaitSubject //------- send req to prescription -------
766           if (full fifo_toPrescription ) then loop end;
767              fifo_toPrescription := enqueue (fifo_toPrescription, reqFromS4);
768  to SendToResou
769  from WaitSubject //------- send req to nurse_report -------
770           if (full fifo_toServeu ) then loop end;
771              fifo_toServeu := enqueue (fifo_toServeu, reqFromS4);
772  to SendToResou
773  from WaitSubject ret := hasRight_DOC (req) to Error
774  from Error loop
775  from Error ret := RET_NON /*@ Send NON*/ to SendRet
```

Traduction automate role (TD) en code Fiacre

```
781 //=============================================================
782 //=============================================================
783 //=============================================================
784 //  le role TD qui active la resource medical_file, prescription et nurse_report
785 //=============================================================
786 //=============================================================
787 //=============================================================
788 process TD (   &activ_TD : read write bool,
789                &activ_medical_file : read write bool,
790                &activ_prescription : read write bool,
791                &activ_nurse_report     : read write bool,
792                &fifo_fromS1 : read write t_fifo_req,
793                &fifo_fromS2 : read write t_fifo_req,
794                &fifo_toMedical_file : read write t_fifo_ret,
795                &fifo_toPrescriptio : read write t_fifo_ret
796                &fifo_toNurse_report : read write t_fifo_ret )
797 is
798  states Stop, Start, WaitSubject, SendToResou,SendRet, Error
799 init to Stop
800  from Stop
801         if (activ_TD = True) then activ_medical_file :=True;
802                                   activ_prescription :=True;
803                                   activ_nurse_report      :=True;
804  to Start end
805  from Start
806  if (activ_TD = False) then activ_medical_file :=False;
807                             activ_prescription :=False;
808                             activ_nurse_report     :=False;
809  to Stop end
810  from Start /* S1 Send Req to TD */
811      case (empty fifo_fromS4) of false -> null end case;
812             reqFromS1 := first fifo_fromS1;
813             fifo_fromS1 := dequeue (fifo_fromS1);
814  to WaitSubject
815  from Start /* S2 Send Req to TD */
816      case (empty fifo_fromS4) of false -> null end case;
817             reqFromS2 := first fifo_fromS2;
818             fifo_fromS2 := dequeue (fifo_fromS2);
819  to WaitSubject
820 from WaitSubject //------- send req to medical_file -------
821          if (full fifo_toMedical_file ) then loop end;
822             fifo_toMedical_file := enqueue (fifo_toMedical_file, reqFromS4);
823 to SendToResou
824  from WaitSubject //------- send req to prescription -------
825          if (full fifo_toPrescription ) then loop end;
826             fifo_toPrescription := enqueue (fifo_toPrescription, reqFromS4);
827 to SendToResou
828 from WaitSubject //------- send req to nurse_report -------
829          if (full fifo_toServeu ) then loop end;
830             fifo_toServeu := enqueue (fifo_toServeu, reqFromS4);
831 to SendToResou
832 from WaitSubject ret := hasRight_TD (req) to Error
833 from Error loop
834 from Error ret := RET_NON /*@ Send NON*/ to SendRet
```

Traduction automate role (Nur) en code Fiacre

```
837  //=============================================================
838  //=============================================================
839  //=============================================================
840  //  le role Nur qui active la resource nurse_report
841  //=============================================================
842  //=============================================================
843  //=============================================================
844
845
846  process Nur (  &activ_Nur : read write bool,
847                 &activ_nurse_report     : read write bool,
848                 &fifo_fromS3 : read write t_fifo_req,
849                 &fifo_toNurse_report : read write t_fifo_ret )
850  is
851   states Stop, Start, WaitSubject, SendToResou,SendRet, Error
852  init to Stop
853   from Stop
854         if (activ_Nur = True) then  activ_nurse_report :=True;
855
856   to Start end
857
858   from Start
859   if (activ_Nur = False) then  activ_nurse_report :=False;
860
861   to Stop end
862
863   from Start /* S3 Send Req to Nur */
864       case (empty fifo_fromS4) of false -> null end case;
865              reqFromS3 := first fifo_fromS3;
866              fifo_fromS3 := dequeue (fifo_fromS3);
867   to WaitSubject
868
869
870  from WaitSubject //------- send req to nurse_report -------
871          if (full fifo_toServeu ) then loop end;
872             fifo_toServeu := enqueue (fifo_toServeu, reqFromS3);
873  to SendToResou
874  from WaitSubject ret := hasRight_Nur (req) to Error
875  from Error loop
876  from Error ret := RET_NON /*@ Send NON*/ to SendRet
```

Traduction automate role (Dir) en code Fiacre

```
879  //=============================================================
880  //=============================================================
881  //=============================================================
882  //  le role Dir qui active la resource staff_table
883  //=============================================================
884  //=============================================================
885  //=============================================================
886
887
888  process Dir (  &activ_Dir : read write bool,
889                 &activ_staff_table      : read write bool,
890                 &fifo_fromS5 : read write t_fifo_req,
891                 &fifo_toNstaff_table : read write t_fifo_ret )
892  is
893   states Stop, Start, WaitSubject, SendToResou,SendRet, Error
894  init to Stop
895   from Stop
896        if (activ_Dir = True) then  activ_staff_table :=True;
897
898   to Start end
899
900   from Start
901   if (activ_Dir = False) then  activ_staff_table :=False;
902
903   to Stop end
904
905  from Start /* S5 Send Req to Dir */
906      case (empty fifo_fromS4) of false -> null end case;
907            reqFromS5 := first fifo_fromS5;
908            fifo_fromS5 := dequeue (fifo_fromS5);
909   to WaitSubject
910
911
912  from WaitSubject //------- send req to staff_table -------
913         if (full fifo_toServeu ) then loop end;
914            fifo_toServeu := enqueue (fifo_toServeu, reqFromS5);
915   to SendToResou
916   from WaitSubject ret := hasRight_Dir (req) to Error
917   from Error loop
918   from Error ret := RET_NON /*@ Send NON*/ to SendRet
```

Traduction automate groupe (CS) en code Fiacre

```
//=====================================================
//=====================================================
//  le groupe cs
//=====================================================
//=====================================================
//=====================================================

process CS (   &activ_CS : read write bool,
               &activ_medical_file : read write bool,
               &activ_prescription : read write bool,
               &activ_nurse_report     : read write bool,
               &fifo_fromS1 : read write t_fifo_req,
               &fifo_fromS2 : read write t_fifo_req,
               &fifo_fromS3 : read write t_fifo_req,
               &fifo_fromS4 : read write t_fifo_req,
               &fifo_toMedical_file : read write t_fifo_ret,
               &fifo_toPrescriptio : read write t_fifo_ret
               &fifo_toNurse_report : read write t_fifo_ret )
is
 states Stop, Start, WaitSubject, SendToResou,SendRet, Error
init to Stop
 from Stop
        if (activ_CS = True) then activ_medical_file :=True;
                                  activ_prescription :=True;
                                  activ_nurse_report      :=True;

 to Start end
  from Start
 if (activ_CS = False) then activ_medical_file :=False;
                            activ_prescription :=False;
                            activ_nurse_report     :=False;
 to Stop end
 from Start /* S1 Send Req to CS */
    case (empty fifo_fromS4) of false -> null end case;
           reqFromS1 := first fifo_fromS4;
           fifo_fromS1 := dequeue (fifo_fromS1);
 to WaitSubject
 from Start /* S2 Send Req to CS */
    case (empty fifo_fromS4) of false -> null end case;
           reqFromS2 := first fifo_fromS4;
           fifo_fromS2 := dequeue (fifo_fromS2);
 to WaitSubject
  from Start /* S3 Send Req to CS */
    case (empty fifo_fromS4) of false -> null end case;
           reqFromS3 := first fifo_fromS4;
           fifo_fromS4 := dequeue (fifo_fromS3);
 to WaitSubject
  from Start /* S4 Send Req to CS */
    case (empty fifo_fromS4) of false -> null end case;
           reqFromS4 := first fifo_fromS4;
           fifo_fromS4 := dequeue (fifo_fromS4);
 to WaitSubject
 from WaitSubject //------- send req to medical_file -------
        if (full fifo_toMedical_file ) then loop end;
           fifo_toMedical_file := enqueue (fifo_toMedical_file, reqFromS4);
 to SendToResou
 from WaitSubject //------- send req to prescription -------
        if (full fifo_toPrescription ) then loop end;
           fifo_toPrescription := enqueue (fifo_toPrescription, reqFromS4);
 to SendToResou
 from WaitSubject //------- send req to nurse_report -------
        if (full fifo_toServeu ) then loop end;
           fifo_toServeu := enqueue (fifo_toServeu, reqFromS4);
 to SendToResou
 from WaitSubject ret := hasRight_CS (req) to Error
 from Error loop
 from Error ret := RET_NON /*@ Send NON*/ to SendRet
```

Traduction automate groupe (AS) en code Fiacre

```
991  //=============================================
992  //=============================================
993  //=============================================
994  //le groupe AS
995  //=============================================
996  //=============================================
997  //=============================================
998
999
1000 process AS (  &activ_AS : read write bool,
1001                &activ_staff_table      : read write bool,
1002                &fifo_fromS5 : read write t_fifo_req,
1003                &fifo_toNstaff_table : read write t_fifo_ret )
1004 is
1005  states Stop, Start, WaitSubject, SendToResou,SendRet, Error
1006 init to Stop
1007  from Stop
1008       if (activ_AS = True) then  activ_staff_table :=True;
1009
1010  to Start end
1011
1012  from Start
1013  if (activ_AS = False) then  activ_staff_table :=False;
1014
1015  to Stop end
1016
1017  from Start /* S5 Send Req to AS */
1018     case (empty fifo_fromS5) of false -> null end case;
1019          reqFromS5 := first fifo_fromS5;
1020          fifo_fromS5 := dequeue (fifo_fromS5);
1021  to WaitSubject
1022
1023
1024 from WaitSubject //------- send req to staff_table -------
1025       if (full fifo_toServeu ) then loop end;
1026          fifo_toServeu := enqueue (fifo_toServeu, reqFromS5);
1027 to SendToResou
1028 from WaitSubject ret := hasRight_AS (req) to Error
1029 from Error loop
1030 from Error ret := RET_NON /*@ Send NON*/ to SendRet
```

Traduction automate ressource (medical_file) en code Fiacre

```
1033  //================================================
1034  //================================================
1035  //================================================
1036  //  la resourse medical_file qui active l'action Read et Write
1037  //================================================
1038  //================================================
1039  //================================================
1040
1041  process medical_file (&activ_medical_file : read write bool,
1042                        &activ_Read : read write bool,
1043                        &activ_Write : read write bool,
1044                        &fifo_fromDoc : read write t_fifo_req,
1045                        &fifo_fromTD : read write t_fifo_req,
1046                        &fifo_fromCS : read write t_fifo_req,
1047                        &fifo_toRead : read write t_fifo_ret)
1048  is
1049  states Stop, Start, WaitRole,WaitGroup, SendToAction,SendRet, Error
1050  init to Stop
1051  from Stop
1052  if (activ_medical_file = True) then activ_Read := True;
1053                                      activ_Write :=True;
1054  to Start end
1055  from Start
1056  if (activ_medical_file = False) then activ_Read := False;
1057                                       activ_Write :=False;
1058  to Stop end
1059  from Start /* DOC Send Req to medical_file */
1060  case (empty fifo_fromDoc) of false -> null end case;
1061  reqFromDoc:= first fifo_fromDoc;
1062  fifo_fromDoc:= dequeue (fifo_fromDoc); to WaitRole
1063  from Start /* TD Send Req to medical_file */
1064  case (empty fifo_fromTD) of false -> null end case;
1065  reqFromTD:= first fifo_fromTD;
1066  fifo_fromTD:= dequeue (fifo_fromTD); to WaitRole
1067  from Start /* CS Send Req to medical_file */
1068  case (empty fifo_fromCS) of false -> null end case;
1069  reqFromCS:= first fifo_fromCS;
1070  fifo_fromCS:= dequeue (fifo_fromCS); to WaitGroup
1071  from WaitRole //------- send req to Read -------
1072  if (full fifo_toRead ) then loop end;
1073  fifo_toRead:=enqueue (fifo_toRead, reqFromRole_i); to SendToAction
1074  from WaitRole ret := hasRightMedical_file (req) to Error
1075  from WaitGroup //------- send req to Read -------
1076  if (full fifo_toRead ) then loop end;
1077  fifo_toRead:=enqueue(fifo_toRead, reqFromGroups_j); to SendToAction
1078  from WaitgGroup ret := hasRightMedical_file (req) to Error
1079  from Error loop
1080  from Error ret := RET_NON /*@ Send NON*/ to SendRet
```

Traduction automate ressource (prescription) en code Fiacre

```
1085  //  la resourse prescription qui active l'action Read et Write
1086  //================================================
1087  //================================================
1088  //================================================
1089
1090  process prescription (&activ_prescription : read write bool,
1091                        &activ_Read : read write bool,
1092                        &activ_Write : read write bool,
1093                        &fifo_fromDoc : read write t_fifo_req,
1094                        &fifo_fromTD : read write t_fifo_req,
1095                        &fifo_fromCS : read write t_fifo_req,
1096                        &fifo_toRead : read write t_fifo_ret)
1097  is
1098  states Stop, Start, WaitRole,WaitGroup, SendToAction,SendRet, Error
1099  init to Stop
1100  from Stop
1101  if (activ_prescription = True) then activ_Read := True;
1102                                      activ_Write :=True;
1103  to Start end
1104  from Start
1105  if (activ_prescription = False) then activ_Read := False;
1106                                       activ_Write :=False;
1107  to Stop end
1108  from Start /* DOC Send Req to prescription */
1109  case (empty fifo_fromDoc) of false -> null end case;
1110  reqFromDoc:= first fifo_fromDoc;
1111  fifo_fromDoc:= dequeue (fifo_fromDoc); to WaitRole
1112
1113  from Start /* TD Send Req to prescription */
1114  case (empty fifo_fromTD) of false -> null end case;
1115  reqFromTD:= first fifo_fromTD;
1116  fifo_fromTD:= dequeue (fifo_fromTD); to WaitRole
1117  from Start /* CS Send Req to prescription */
1118  case (empty fifo_fromCS) of false -> null end case;
1119  reqFromCS:= first fifo_fromCS;
1120  fifo_fromCS:= dequeue (fifo_fromCS); to WaitGroup
1121  from WaitRole //------- send req to Read -------
1122  if (full fifo_toRead ) then loop end;
1123  fifo_toRead:=enqueue (fifo_toRead, reqFromRole_i); to SendToAction
1124  from WaitRole ret := hasRightPrescription (req) to Error
1125  from WaitGroup //------- send req to Read -------
1126  if (full fifo_toRead ) then loop end;
1127  fifo_toRead:=enqueue(fifo_toRead, reqFromGroups_j); to SendToAction
1128  from WaitgGroup ret := hasRightPrescription (req) to Error
1129  from Error loop
1130  from Error ret := RET_NON /*@ Send NON*/ to SendRet
```

Traduction automate ressource (nurse_repot) en code Fiacre

```
1132  //=============================================
1133  //=============================================
1134  //=============================================
1135  //  la resourse nurse_report qui active L'action Read et Write
1136  //=============================================
1137  //=============================================
1138  //=============================================
1139
1140  process nurse_report (&activ_nurse_report : read write bool,
1141                        &activ_Read : read write bool,
1142                        &activ_Write : read write bool,
1143                        &fifo_fromDoc : read write t_fifo_req,
1144                        &fifo_fromTD : read write t_fifo_req,
1145                        &fifo_fromCS : read write t_fifo_req,
1146                        &fifo_toRead : read write t_fifo_ret)
1147  is
1148  states Stop, Start, WaitRole,WaitGroup, SendToAction,SendRet, Error
1149  init to Stop
1150  from Stop
1151  if (activ_nurse_report = True) then activ_Read := True;
1152                                      activ_Write :=True;
1153  to Start end
1154  from Start
1155  if (activ_nurse_report = False) then activ_Read := False;
1156                                       activ_Write :=False;
1157  to Stop end
1158  from Start /* DOC Send Req to nurse_report */
1159  case (empty fifo_fromDoc) of false -> null end case;
1160  reqFromDoc:= first fifo_fromDoc;
1161  fifo_fromDoc:= dequeue (fifo_fromDoc); to WaitRole
1162  from Start /* TD Send Req to nurse_report */
1163  case (empty fifo_fromTD) of false -> null end case;
1164  reqFromTD:= first fifo_fromTD;
1165  fifo_fromTD:= dequeue (fifo_fromTD); to WaitRole
1166  from Start /* CS Send Req to nurse_report */
1167  case (empty fifo_fromCS) of false -> null end case;
1168  reqFromCS:= first fifo_fromCS;
1169  fifo_fromCS:= dequeue (fifo_fromCS); to WaitGroup
1170  from WaitRole //------- send req to Read -------
1171  if (full fifo_toRead ) then loop end;
1172  fifo_toRead:=enqueue (fifo_toRead, fifo_fromDoc); to SendToAction
1173  from WaitRole ret := hasRightPrescription (req) to Error
1174  from WaitRole
1175  if (full fifo_toRead ) then loop end;
1176  fifo_toRead:=enqueue (fifo_toRead, fifo_fromTD); to SendToAction
1177  from WaitRole ret := hasRightPrescription (req) to Error
1178  from WaitGroup //------- send req to Read -------
1179  if (full fifo_toRead ) then loop end;
1180  fifo_toRead:=enqueue(fifo_toRead, fifo_fromCS); to SendToAction
1181  from WaitgGroup ret := hasRightPrescription (req) to Error
1182  from WaitRole //------- send req to Write -------
1183  if (full activ_Write ) then loop end;
1184  activ_Write:=enqueue (activ_Write, fifo_fromDoc); to SendToAction
1185  from WaitRole ret := hasRightPrescription (req) to Error
1186  from WaitRole
1187  if (full fifo_toRead ) then loop end;
1188  fifo_toRead:=enqueue (fifo_toRead, fifo_fromTD); to SendToAction
1189  from WaitRole ret := hasRightPrescription (req) to Error
1190  from WaitGroup //------- send req to Read -------
1191  if (full activ_Write ) then loop end;
1192  activ_Write:=enqueue(activ_Write, fifo_fromCS); to SendToAction
1193  from WaitgGroup ret := hasRightPrescription (req) to Error
1194  from Error loop
1195  from Error ret := RET_NON /*@ Send NON*/ to SendRet
```

Traduction automate ressource (staff_table) en code Fiacre

```
1197  //=============================================
1198  //=============================================
1199  //=============================================
1200  //  la resourse staff_table qui active l'action Read et Write
1201  //=============================================
1202  //=============================================
1203  //=============================================
1204
1205  process staff_table (&activ_nurse_report : read write bool,
1206                       &activ_Read : read write bool,
1207                       &activ_Write : read write bool,
1208                       &fifo_fromDir : read write t_fifo_req,
1209                       &fifo_fromAS : read write t_fifo_req,
1210                       &fifo_toRead : read write t_fifo_ret,
1211                       &activ_Write : read write t_fifo_ret)
1212  is
1213  states Stop, Start, WaitRole,WaitGroup, SendToAction,SendRet, Error
1214  init to Stop
1215  from Stop
1216  if (activ_nurse_report = True) then activ_Read := True;
1217                                      activ_Write :=True;
1218  to Start end
1219  from Start
1220  if (activ_nurse_report = False) then activ_Read := False;
1221                                       activ_Write :=False;
1222  to Stop end
1223  from Start /* Dir Send Req to staff_table */
1224  case (empty fifo_fromDir) of false -> null end case;
1225  reqFromDir:= first fifo_fromDir;
1226  fifo_fromDir:= dequeue (fifo_fromDir); to WaitRole
1227  from WaitRole //------- send req to Read -------
1228  if (full fifo_toRead ) then loop end;
1229  fifo_toRead:=enqueue (fifo_toRead, reqFromRole_i); to SendToAction
1230  from WaitRole ret := hasRightPrescription (req) to Error
1231  from WaitGroup //------- send req to Read -------
1232  if (full fifo_toRead ) then loop end;
1233  fifo_toRead:=enqueue(fifo_toRead, fifo_fromAS); to SendToAction
1234  from WaitgGroup ret := hasRightPrescription (req) to Error
1235  from WaitRole //------- send req to Write -------
1236  if (full activ_Write ) then loop end;
1237  activ_Write:=enqueue (activ_Write, reqFromRole_i); to SendToAction
1238  from WaitRole ret := hasRightPrescription (req) to Error
1239  from WaitGroup //------- send req to Write -------
1240  if (full activ_Write ) then loop end;
1241  activ_Write:=enqueue(activ_Write, fifo_fromAS); to SendToAction
1242  from WaitgGroup ret := hasRightPrescription (req) to Error
1243  from Error loop
1244  from Error ret := RET_NON /*@ Send NON*/ to SendRet
```

Traduction automate action (read) en code Fiacre

```
1246  //================================================
1247  //================================================
1248  //================================================
1249  //  l'action read
1250  //================================================
1251  //================================================
1252  //================================================
1253
1254
1255  process Read (&activ_Read  : read write bool,
1256                &fifo_fromMedical_file : read write t_fifo_req,
1257                &fifo_fromPescription : read write t_fifo_req,
1258                &fifo_fromNurse_report : read write t_fifo_req,
1259                &fifo_fromStaff_table : read write t_fifo_req
1260                )
1261  is
1262  states Stop, Start, WaitResource, SendRet, Error
1263  init to Stop
1264  from Stop if (activ_Read = True) then to Start end
1265  from Start if (activ_Read = False) then to Stop end
1266
1267  from Start /* medical_file Send Req to Read */
1268  case (empty fifo_fromMedical_file) of false -> null end case;
1269  fifo_fromMedical_file:= first fifo_fromMedical_file;
1270  fifo_fromMedical_file := dequeue (fifo_fromMedical_file);
1271  to WaitResource
1272
1273  from Start /* prescription Send Req to Read */
1274  case (empty fifo_fromPescription) of false -> null end case;
1275  fifo_fromPescription:= first fifo_fromPescription;
1276  fifo_fromPescription := dequeue (fifo_fromPescription);
1277  to WaitResource
1278
1279  from Start /* nurse_report Send Req to Read */
1280  case (empty fifo_fromNurse_report) of false -> null end case;
1281  fifo_fromNurse_report:= first fifo_fromNurse_report;
1282  fifo_fromNurse_report := dequeue (fifo_fromNurse_report);
1283  to WaitResource
1284
1285  from Start /* staff_table Send Req to Read */
1286  case (empty fifo_fromStaff_table) of false -> null end case;
1287  fifo_fromStaff_table:= first fifo_fromStaff_table;
1288  fifo_fromStaff_table := dequeue (fifo_fromStaff_table);
1289  to WaitResource
1290
1291  from WaitResource //------- send ret to ENV -------
1292  ret := RET_OK to SendRet
1293  from SendRet loop
1294  from WaitResource //------- send ret to ENV -------
1295  ret := RET_NON to SendRet
1296  from Error loop
```

Traduction automate action (write) en code Fiacre

```
1298  //================================================
1299  //================================================
1300  //================================================
1301  //  l'action write
1302  //================================================
1303  //================================================
1304  //================================================
1305
1306
1307  process Write (&activ_Write  : read write bool,
1308                 &fifo_fromMedical_file : read write t_fifo_req,
1309                 &fifo_fromPescription : read write t_fifo_req,
1310                 &fifo_fromNurse_report : read write t_fifo_req,
1311                 &fifo_fromStaff_table : read write t_fifo_req
1312                 )
1313  is
1314  states Stop, Start, WaitResource, SendRet, Error
1315  init to Stop
1316  from Stop if (activ_Write = True) then to Start end
1317  from Start if (activ_Write = False) then to Stop end
1318
1319  from Start /* medical_file Send Req to Write */
1320  case (empty fifo_fromMedical_file) of false -> null end case;
1321  fifo_fromMedical_file:= first fifo_fromMedical_file;
1322  fifo_fromMedical_file := dequeue (fifo_fromMedical_file);
1323  to WaitResource
1324
1325  from Start /* prescription Send Req to Write */
1326  case (empty fifo_fromPescription) of false -> null end case;
1327  fifo_fromPescription:= first fifo_fromPescription;
1328  fifo_fromPescription := dequeue (fifo_fromPescription);
1329  to WaitResource
1330
1331  from Start /* nurse_report Send Req to Write */
1332  case (empty fifo_fromNurse_report) of false -> null end case;
1333  fifo_fromNurse_report:= first fifo_fromNurse_report;
1334  fifo_fromNurse_report := dequeue (fifo_fromNurse_report);
1335  to WaitResource
1336
1337  from Start /* staff_table Send Req to Write */
1338  case (empty fifo_fromStaff_table) of false -> null end case;
1339  fifo_fromStaff_table:= first fifo_fromStaff_table;
1340  fifo_fromStaff_table := dequeue (fifo_fromStaff_table);
1341  to WaitResource
1342
1343  from WaitResource //------- send ret to ENV -------
1344  ret := RET_OK to SendRet
1345  from SendRet loop
1346  from WaitResource //------- send ret to ENV -------
1347  ret := RET_NON to SendRet
1348  from Error loop
```

Component de POL_1

```
1350
1351  //=======================================================
1352  //=======================================================
1353  //==================== Component ========================
1354  //=======================================================
1355  //=======================================================
1356
1357
1358  component AC_POL_1 is
1359  var
1360
1361  activ_s1    :  bool,
1362  activ_s2    :  bool,
1363  activ_s3    :  bool,
1364  activ_s4    :  bool,
1365  activ_s5    :  bool,
1366
1367  activ_Doc   :  bool,
1368  activ_TD    :  bool,
1369  activ_Nur   :  bool,
1370  activ_Dir   :  bool,
1371
1372  activ_CS    :  bool,
1373  activ_AS    :  bool,
1374
1375  activ_medical_file  :  bool,
1376  activ_prescription  :  bool,
1377  activ_nurse_report  :  bool,
1378  activ_staff_table   :  bool,
1379
1380  activ_Read   :  bool,
1381  activ_Write  :  bool,
1382
1383  fifo_fromS1 :  t_fifo_req,
1384  fifo_fromS2 :  t_fifo_req,
1385  fifo_fromS3 :  t_fifo_req,
1386  fifo_fromS4 :  t_fifo_req,
1387  fifo_fromS5 :  t_fifo_req,
1388
1389  fifo_fromDoc :  t_fifo_req,
1390  fifo_fromTD  :  t_fifo_req,
1391  fifo_fromDir :  t_fifo_req,
1392  fifo_fromNur :  t_fifo_req,
1393
1394  fifo_fromCS :  t_fifo_req,
1395  fifo_fromAS :  t_fifo_req,
1396
1397  fifo_fromMedical_file :  t_fifo_req,
1398  fifo_fromPescription  :  t_fifo_req,
1399  fifo_fromNurse_report :  t_fifo_req,
1400  fifo_fromStaff_table  :  t_fifo_req,
1401
1402  fifo_toDoc :  t_fifo_ret,
1403  fifo_toTD  :  t_fifo_ret,
1404  fifo_toDir :  t_fifo_ret,
1405  fifo_toNur :  t_fifo_ret,
1406
1407  fifo_toCS   :  t_fifo_ret,
1408  fifo_fromAS :  t_fifo_ret,
1409
1410  fifo_toMedical_file :  t_fifo_ret,
1411  fifo_toPescription  :  t_fifo_ret,
1412  fifo_toNurse_report :  t_fifo_ret,
1413  fifo_toStaff_table  :  t_fifo_ret,
1414
1415  fifo_toRead   :t_fifo_ret
1416
1417  init
1418
1419  activ_s1          := false;
1420  activ_s2          := false;
1421  activ_s3          := false;
1422  activ_s4          := false;
1423  activ_s5          := false;
1424
1425  activ_Doc    := false;
1426  activ_TD     := false;
1427  activ_Nur    := false;
1428  activ_Dir    := false;
1429
1430  activ_CS     := false;
1431  activ_AS     := false;
1432
1433  activ_medical_file  := false;
1434  activ_prescription  := false;,
1435  activ_nurse_report  := false;,
1436  activ_staff_table   := false;
1437
1438  activ_Read    := false;
1439  activ_Write   := false;
1440
1441  par
1442       S1 (&activ_s1, &activ_TD, &activ_CS, &fifo_fromEnv, &fifo_toRole_TD, &fifo_toGroup_CS)
1443      //  S2 ( &activ_s2, &activ_TD, &activ_CS, &fifo_fromEnv, &fifo_toRole_TD,&fifo_toGroup_CS)
1444      //  S3 ( &activ_s3, &activ_Nur, &activ_CS, &fifo_fromEnv, &fifo_toRole_Nur, &fifo_toGroup_CS)
1445      //  S4 ( &activ_s4l, &activ_Doc, &activ_CS, &fifo_fromEnv, &fifo_toRole_Doc, &fifo_toGroup_CS)
1446      //  S5 ( &activ_s5, &activ_Dir, &activ_AS, &fifo_fromEnv, &fifo_toRole_Dir, &fifo_toGroup_AS )
1447
1448      //  DOC (&activ_DOC, &activ_medical_file, &activ_prescription, &activ_nurse_report, &fifo_fromS4,
1449           &fifo_toMedical_file, &fifo_toPrescriptio,&fifo_toNurse_report)
1450      //  TD  (&activ_TD, &activ_medical_file, &activ_prescription, &activ_nurse_report, &fifo_fromS1,
1451           &fifo_fromS2, &fifo_toMedical_file, &fifo_toPrescriptio, &fifo_toNurse_report)
1452      //  Nur (&activ_Nur,&activ_nurse_report, &fifo_fromS3, &fifo_toNurse_report )
1453      //  Dir (&activ_Dir, &activ_staff_table, &fifo_fromS5, &fifo_toNstaff_tabl)
1454
1455      //  CS  (&activ_CS, &activ_medical_file, &activ_prescription, &activ_nurse_report, &fifo_fromS1, &fifo_fromS2,
1456           &fifo_fromS3, &fifo_fromS4, &fifo_toMedical_file, &fifo_toPrescriptio, &fifo_toNurse_report)
1457      //  AS  (&activ_AS,&activ_staff_table,&fifo_fromS5,&fifo_toNstaff_table )
1458
1459      //  medical_file (&activ_medical_file, &activ_Read, &activ_Write, &fifo_fromDoc, &fifo_fromTD, &fifo_fromCS, &f
1460      //  prescription (&activ_prescription, &activ_Read, &activ_Write, &fifo_fromDoc, &fifo_fromTD, &fifo_fromCS, &f
1461      //  nurse_report (&activ_nurse_report, &activ_Read, &activ_Write, &fifo_fromDoc, &fifo_fromTD, &fifo_fromCS, &f
1462      //  staff_table  (&activ_nurse_report, &activ_Read, &activ_Write, &fifo_fromDir, &fifo_fromAS, &fifo_toRead, &a
1463
1464      //  Read  (&activ_Read, &fifo_fromMedical_file, &fifo_fromPescription, &fifo_fromNurse_report, &fifo_fromStaff_
1465      //  Write (&activ_Write, &fifo_fromMedical_file, &fifo_fromPescription, &fifo_fromNurse_report, &fifo_fromStaff
1466  end par
1467
1468  AC_POL_1
```

9 Annexe : Publications

Les publications rédigées durant ce travail sont listées ci-dessous.

- M. W. Krakallah et S. Nait-Bahloul, « Methodological Sketch of Observer-Based Access Control », in *Proceedings of the 2nd international Conference on Big Data, Cloud and Applications*, 2017, p. 62.

- M. W. Krakallah et S. Nait-Bahloul, « Formalisation of access control based on observers automata », *Int. J. Syst. Syst. Eng.*, vol. 11, n° 2, p. 170-197, janv. 2021, doi: 10.1504/IJSSE.2021.116045.