Protection obligatoire des serveurs d'applications Web : application aux processus métiers

Maxime Fonda

UNIVERSITÉ D'ORLÉANS

ÉCOLE DOCTORALE MATHEMATIQUES, INFORMATIQUE, PHYSIQUE THEORIQUE et INGENIERIE DES SYSTEMES

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLÉANS

THÈSE présentée par :

Maxime FONDA

soutenue le : **21 Mai 2014**

pour obtenir le grade de : **Docteur de l'université d'Orléans**

Discipline : **INFORMATIQUE**

Protection obligatoire des serveurs d'applications Web :

Application aux processus métiers.

THÈSE dirigée par :

Christian TOINARD	Professeur des Universités, INSA Centre Val-de-Loire

RAPPORTEURS :

Fabrice BOUQUET	Professeur des Universités, LIFC, INRIA Nancy Grand-Est
Franck BARBIER	Professeur des Universités, Université de Pau

JURY :

Jean-Michel COUVREUR	Professeur des Universités, LIFO, **Président du jury**
Franck BARBIER	Professeur des Universités, Université de Pau
Fabrice BOUQUET	Professeur des Universités, LIFC, INRIA Nancy Grand-Est
Stéphane MOINARD	Directeur technique, QualNet
Christian TOINARD	Professeur des Universités, INSA Centre Val-de-Loire

Remerciements

Je remercie M. Fabrice Bouquet et M. Franck Barbier pour avoir accepté d'évaluer mes travaux en étant rapporteurs de cette thèse.

Je remercie également M. Jean-Michel Couvreur pour avoir accepté d'être président de mon jury de thèse.

Je tiens à remercier mon directeur de thèse, M. Christian Toinard pour m'avoir offert la possibilité de réaliser cette thèse et pour son soutien et son aide durant la durée de mes recherches. Je remercie également la région Centre et la société QualNet pour avoir financé cette thèse et m'avoir ainsi permis de réaliser ces travaux de recherche.

Je tiens à remercier tous les membres de l'équipe Sécurité et Distribution des Système pour les nombreuses discussions enrichissantes et pour l'aide qu'ils m'ont fourni durant cette thèse. Je remercie aussi l'ENSI de Bourges pour m'avoir accueilli dans ses locaux et m'avoir offert un cadre de travail agréable.

J'aimerai également remercier tous les membres de la société QualNet pour m'avoir accueilli, et particulièrement M. Stéphane Moinard pour m'avoir encadré durant ces trois ans et faire partie de mon jury de thèse. Nos discussions ont grandement contribué à cette thèse et m'ont beaucoup appris sur le monde de l'entreprise.

Merci aux thésards de l'Ensi de Bourges, Damien et Pierre. Trois ans ont passé, à travailler sur nos thèses respectives et à se soutenir mutuellement. Je vous remercie pour les nombreux bons moments passés ensembles et vous souhaite bon courage pour mener à bien vos thèses.

À mes amis, Bérengère, Cécile, Élodie, Julien, Natalia, Thierry, Yann,... et tous ceux que j'oublie, merci pour tout !

Mes dernières pensées iront vers ma famille, qui a toujours cru en moi et a su me soutenir dans les moments difficiles.

Table des matières

Table des figures

Table des algorithmes

Table des tableaux

Table des listings

Table des définitions

Glossaire des sigles

.Net	Le framework de développement informatique proposé par Microsoft	page 65
BIBA	Modèle d'intégrité conçu par K. J. Biba	page 25
BLP	Modèle de confidentialité Bell-LaPadula, nommé d'après ses auteurs D. E. Bell et L. J. La Padula	page 25
BPEL	Business Process Execution Language : langage de programmation destiné à l'exécution des procédures d'entreprise	page 30
BPMN	Business Process Model and Notation : notation graphique standardisée pour modéliser des procédures d'entreprise ou des processus métier	page 30
CIL	Common Intermediate Language : code intermédiaire utilisé par le framework .Net	page 27
CLR	Common Language Runtime : environnement d'exécution fournit par le .NET Framework qui exécute le code et offre des services qui simplifient le processus de développement	page 26
DAC	Discretionary Access Control : Controle d'accès discrétionnaire	page 24
DTE	Domain and Type Enforcement : Modèle de protection associant des domaines aux sujets et des types aux objets	page 25
GED	Gestion électronique de documents	page 17
HRU	Modèle théorique de DAC établi par Harrison, Ruzzo et Ullman	page 24
MAC	Mandatory Access Control : Controle d'accès obligatoire	page 24
RBAC	Role-Based Access Control : Contrôle d'accès à base de rôles	page 26

SaaS	Software as a Service : modèle d'exploitation commerciale des logiciels dans lequel ceux-ci sont installés sur des serveurs distants	page 17
SELinux	Security-Enhanced Linux : mécanisme de contrôle d'accès mandataire	page 26
SOA	Service Oriented Architecture : une architecture logicielle s'appuyant sur un ensemble de services simples	page 29
XACML	eXtensible Access Control Markup Language : Langage à balise extensible pour la configuration du contrôle d'accès	page 29
XPDL	(XML Process Definition Language : langage dérivé du XML de définition de processus métier	page 30

Chapitre 1

Introduction

1.1 Contexte et objectifs

Le besoin des entreprises en terme de sécurité des systèmes d'information prend de l'ampleur de jour en jour. Les cas d'attaque de serveurs web sont quotidiens. Ces serveurs sont particulièrement vulnérables puisqu'ils permettent d'exécuter des applications tierces qui peuvent être utilisées pour exécuter du code malveillant sur le serveur. Le risque sur ces serveurs est d'autant plus important qu'ils sont utilisés pour héberger des applications très diverses avec des objectifs incompatibles. La nécessité de posséder des mécanismes de protection sur ces serveurs qui puissent gérer différents domaines avec des politiques spécifiques apparait évidente. L'utilisation croissante d'applications de type *Software as a Service* (SaaS) ou *Cloud* augmente le besoin en terme de sécurité des applications hébergées. Les attaques visant à voler des informations, qu'elles proviennent de l'extérieur de l'entreprise, ou de l'intérieur par le biais d'utilisateurs malveillants, sont de plus en plus importantes et doivent être évitées à tout prix, car la responsabilité juridique des entreprises proposant et hébergeant des applications de ce type peut être engagée.

Pour une défense en profondeur efficace des systèmes d'informations (cf. figure 1.1), il est nécessaire de protéger aussi bien les systèmes d'exploitation que les intergiciels et les applications Web s'exécutant sur ces systèmes d'exploitation.

Un grand nombre de modèles et de mécanismes de protection existent au niveau des systèmes d'exploitation. Nous pouvons distinguer deux grandes classes de modèles de contrôle d'accès : les modèles de contrôle d'accès discrétionnaire (DAC) dans lequel les utilisateurs finaux définissent eux-mêmes les droits d'accès à leur ressources, et les modèles de contrôle d'accès obligatoire visant à définir des objectifs de sécurité et les garantir. Les mécanismes de contrôle d'accès discrétionnaire ont prouvé leur inefficacité à protéger efficacement un système d'information ([TCSEC, 1985, Ferraiolo et Kuhn, 1992, Loscocco *et al.*, 1998]). Seuls les mécanismes de contrôle d'accès obligatoire peuvent protéger efficacement un système. En terme de mécanismes de contrôle d'accès obligatoire, nous pouvons par exemple citer SELinux pour les systèmes d'exploitation Linux ou MIC pour les systèmes d'exploitations Windows. Cependant, ceux-ci ne sont pas efficaces pour exprimer des besoins de sécurité pour des applications Web et pour leur protection, car ils s'appliquent à un niveau trop bas.

Cette thèse a été effectuée au sein de la société QualNet, spécialisée dans les applications de *gestion électronique de documents* (GED) et de gestion des processus métiers (workflows) basées des applications Web sur des environnements Microsoft. Celle-ci propose notamment l'hébergement en mode SaaS de ces applications, et avec plus de 350 clients et 300 000 utilisateurs à travers le monde, les éventualités d'accès non autorisés et

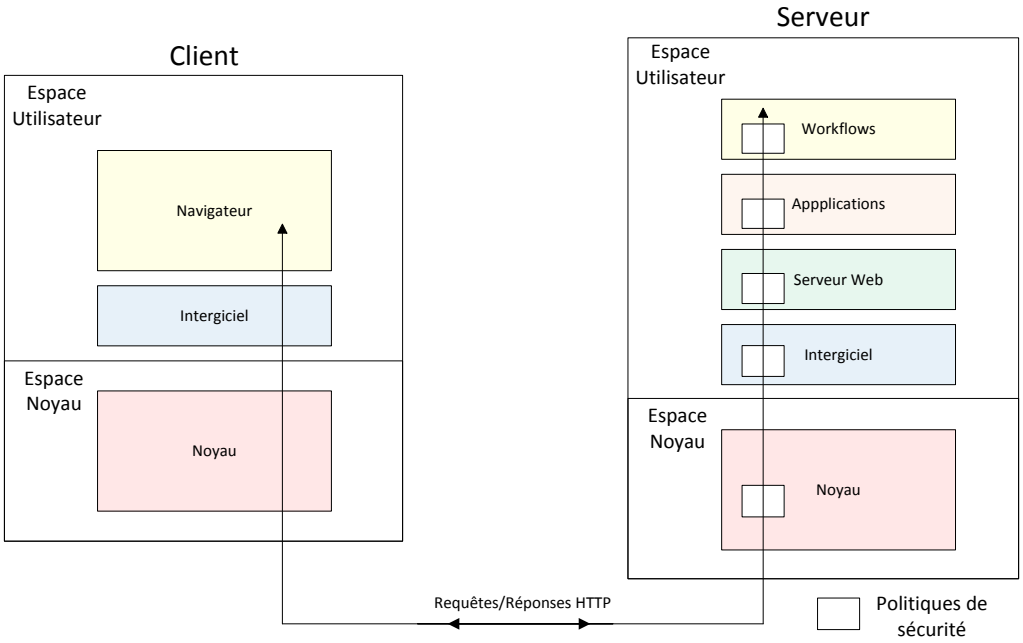

FIGURE 1.1 – Défense en profondeur des serveurs d'applications Web.

de vol d'informations sont croissantes. La nécessité de protéger efficacement les applications éditées par QualNet apparait donc essentielle.

C'est pourquoi nous proposons, dans le cadre de cette thèse, de définir, modéliser et implémenter une nouvelle approche de protection obligatoire pour les serveurs d'applications Web. Ce travail propose une nouvelle architecture de contrôle d'accès répondant à des objectifs précis.

Tout d'abord, notre architecture doit protéger efficacement les serveurs d'applications Web en proposant une méthode de contrôle d'accès obligatoire dynamique. Le contrôle d'accès doit être indépendant des types de serveurs et des technologies, supportant ainsi l'ouverture et l'extensibilité du Web. Elle doit faciliter l'expression des politiques de sécurité requises. De plus, et en premier lieu, elle doit simplifier l'administration en automatisant complètement le calcul des politiques de sécurité. Elle doit prendre en compte les environnements de workflows reposant sur le Web.

Bien que l'approche doit supporter tout type d'environnement Web, elle doit être utilisable et intégrée au sein des environnements QualNet. Du point de vue serveur Web, celle-ci doit à minima être applicable aux serveurs Web Microsoft, c'est-à-dire au serveur Web *Internet Information Services* (IIS) et au framework *.Net*. Cependant, notre solution doit être indépendante de l'intergiciel utilisé et des langages de programmation, aussi bien du point de vue client que serveur Web. Au niveau des workflows, notre protection obligatoire doit à minima pouvoir être appliquée au système de workflows de QualNet *Intraqual Dynamic*. Si l'approche doit être indépendante des normes existantes, elle doit cependant s'adapter à celles-ci et s'appliquer à un contexte plus large que les environnements Microsoft. Elle doit notamment s'adapter à différents types d'applications et de workflows en étant suffisamment extensible pour exprimer différents modèles d'applications tel que SOA et de workflows tel que XPDL par exemple.

Afin de simplifier l'administration de la sécurité au sein des serveurs d'applications Web, elle ne doit demander aucune configuration spécifique au niveau intergiciel ou serveur Web mais automatiser complètement le calcul des politiques de sécurité. Elle doit aussi

proposer des moyens permettant de tester et vérifier simplement la mise en œuvre des politiques de sécurité calculées.

1.2 Apports de la thèse

Cette thèse propose une approche orientée serveur d'application Web, alors que l'essentiel des solutions existantes concernent plutôt des aspects systèmes ou organisationnels de plus haut niveau.

Nous proposons un modèle abstrait permettant de modéliser tous types d'applications Web. La modélisation des applications Web est un élément essentiel à la définition future des objectifs de sécurité qui sont nécessaires à la protection de ces applications. Les modèles d'applications Web existants, comme par exemple *Service Oriented Architecture* (SOA) peuvent être représentés par notre modèle abstrait d'application.

Nous définissons un langage de protection dédié permettant d'exprimer les besoins en terme de contrôle d'accès au sein d'un serveur d'application Web et s'appuyant sur le modèle d'application que nous proposons. Ce langage permet de représenter tous types d'autorisations d'un sujet (une entité active du système) sur un objet c'est-à-dire une ressource hébergée sur un serveur d'applications Web. Il prend en compte le caractère extensible et versatile du Web en permettant de définir les différents attributs permettant de caractériser efficacement des sujets, des objets et les actions concernées.

Nous proposons une architecture de protection obligatoire permettant d'appliquer de manière efficace et sûre les besoins en terme contrôle d'accès qui sont exprimés dans notre langage de protection. Cette protection obligatoire est dynamique car elle permet de définir à chaque requête entrante sur un serveur d'applications web les sujets, les actions et les objets associés à cette requête sous la forme de règles potentielles d'accès. Elle est également indépendante des applications protégées car elle s'appuie notamment sur un adaptateur applicatif qui permet d'interfacer notre protection avec n'importe quel type d'application web.

Enfin, nous proposons une méthode de calcul automatisé des politiques de sécurité s'appuyant sur notre modèle d'application. Cette méthode permet d'obtenir, à partir du modèle d'une application web, la politique de sécurité complète associée à cette application. Les politiques de sécurité obtenues sont directement utilisables par notre protection obligatoire. Cette méthode facilite l'administration de la politique de sécurité en permettant de calculer la politique de sécurité pour l'application à protéger.

Les difficultés pour calculer une politique de sécurité pour les systèmes de contrôle d'accès obligatoire sont le principal frein à la mise en place de tels systèmes. En effet, une politique de sécurité adaptée peut contenir des milliers de règles d'autorisation, ce qui rend leur définition fastidieuse voire impossible si elles doivent être définies manuellement. Ainsi, grâce à cette méthode de calcul automatisé des politiques de sécurité, la problématique de l'obtention de politiques de sécurité est ici résolue. De plus, nous proposons une méthode de test de cette politique, et un moyen de vérification de la sûreté de cette politique, ce qui en soit est un problème tout aussi difficile que celui du calcul de la politique requise.

1.3 Plan du mémoire

La suite du document s'articule en quatre chapitres. Le chapitre 2 présente l'ensemble des travaux de la littérature dans le domaine de la protection des systèmes, et notamment des applications web et des workflows. Il conclut sur la nécessité d'un modèle de protection

dédié aux serveurs d'applications Web s'appuyant sur une modélisation des applications Web, et de la nécessité de proposer une méthode efficace de calcul des politiques de sécurité pour automatiser l'administration de la protection.

Le chapitre 3 propose une modélisation abstraite des applications Web prenant en compte tous types de ressources avec une finesse et une extensibilité suffisante. Cette modélisation abstraite des applications Web nous permet de proposer d'une part une protection obligatoire dynamique des serveurs d'applications et le langage associé d'expression des politiques de sécurité et d'autre part une approche de calcul automatisé de ces politiques de sécurité. Ce modèle d'application Web n'est pas lié à une technologie particulière et permet de formaliser aisément tous types d'application hébergée sur différents types de serveurs Web. Le modèle de protection obligatoire proposé est également indépendant du type de serveur Web et des technologies utilisées. De plus, nous proposons un modèle générique de workflows qui, tout en pouvant s'adapter aux différents standards et technologies, se projette automatiquement et efficacement sur notre modèle abstrait d'applications Web. Nous définissons enfin une méthode de test et de vérification des politiques de sécurité calculées automatiquement. Ainsi, non seulement nous simplifions l'administration de la protection, mais nous testons et vérifions automatiquement son bon fonctionnement.

Le chapitre 4 décrit l'implémentation de cette protection obligatoire sur des environnements Web basés sur les technologies Web Microsoft (serveur Web IIS, Framework .Net). Cette implémentation reste indépendante des applications Web protégées car elle repose sur l'utilisation d'un adaptateur applicatif pour s'interfacer avec n'importe quelle application, et peut donc notamment fonctionner sur les applications de Workflows développées par QualNet mais également toute autre application. Il décrit également l'implémentation des méthodes de calcul automatisé et de test des politiques de sécurité proposée dans le chapitre 3.

Enfin, les expérimentations présentées dans le chapitre 5 montrent que ces implémentations sont fonctionnelles et parfaitement intégrées aux applications QualNet. Ce chapitre étudie également l'impact de la mise en place de notre protection obligatoire, notamment sur le temps de traitement des requêtes par une application. La surcharge imposée par l'utilisation de notre protection obligatoire reste faible vis-à-vis de la taille des environnements protégés et des politiques de sécurités appliquées.

Chapitre 2

État de l'art

Les problématiques de contrôle d'accès sont présentes à tous les niveaux des systèmes d'information. Ce chapitre résume les travaux menés dans le domaine de la protection en se centrant plus particulièrement sur le contexte de l'étude à savoir les environnements Web Microsoft. Nous présentons d'abord les notions de politiques et de propriétés de sécurité. Nous détaillons ensuite les principaux modèles de contrôle d'accès. Nous poursuivons par la description de la mise en œuvre d'un contrôle d'accès au niveau système, intergiciel, Web et workflows. Une discussion établira le manque de modèles efficaces concernant le contrôle d'accès pour le Web et les workflows et montrera la nécessité de nouveaux modèles utilisables en grandeur réelle notamment sur les environnements Web de Microsoft.

2.1 Politiques de sécurité

Une politique de sécurité permet de définir les objectifs de sécurité attendus sur le système. Un exemple d'objectif est la confidentialité, garantissant qu'un domaine non privilégié ne puisse lire les informations issues d'un domaine privilégié. Les propriétés de sécurité sont regroupées en trois catégories : confidentialité, intégrité et disponibilité [TCSEC, 1985]. La confidentialité concerne les accès en lecture et l'intégrité les accès en écriture. La disponibilité concerne essentiellement les temps d'accès aux services.

Définition 1 (Politique de sécurité)**.**

Une politique de sécurité est une déclaration qui partitionne les états d'un système en un ensemble d'états autorisés (ou sûrs) et un ensemble d'états non-autorisés (ou non-sûrs).

Une politique de sécurité explicite les comportements autorisés ou ceux interdits. Elle définit les états sûrs et les états non sûrs.

Définition 2 (Système sûr)**.**

Un système sûr est un système qui, partant d'un état sûr, ne pourra jamais entrer dans un état non-sûr.

Généralement, les politiques considèrent implicitement que tout ce qui n'est pas autorisé est interdit. Cependant, dans certains cas, il peut être utile d'expliciter ce qui est interdit. Il est donc souvent nécessaire de pouvoir tester et vérifier que le système satisfait bien la politique de sécurité et reste bien dans un état sûr.

2.1.1 Propriétés de sécurité générales

Les trois propriétés d'intégrité, de confidentialité et de disponibilité sont considérées différemment suivant le contexte auquel elles s'appliquent. Leur représentation est liée aux besoins des utilisateurs, des services et des lois en vigueur. Différentes normes s'intéressent à la définition de ces propriétés [ITSEC, 1991, TCSEC, 1985]. Bien que de nombreuses variantes de ces propriétés [Briffaut, 2007] existent dans la littérature, il est possible d'en donner des définitions générales.

Confidentialité

Selon [Bishop, 2003], la confidentialité "prévient la divulgation d'informations non autorisées". Plus précisément, elle peut être définie comme suit :

Définition 3 (Confidentialité)**.**

Soit I de l'information et soit X un ensemble d'entités non autorisées à accéder à I. La propriété de confidentialité de X envers I est respectée si aucun membre de X ne peut obtenir de l'information de I.

La propriété de confidentialité implique que l'information ne doit pas être divulguée à certaines entités, mais lisible par les autres.

Intégrité

La propriété d'intégrité prévient une modification non autorisée d'une information. Plus précisément, la propriété d'intégrité se définit comme suit :

Définition 4 (Intégrité)**.**

Soit X un ensemble d'entités et soit I de l'information ou une ressource. La propriété d'intégrité de X envers I est respectée si aucun membre de X ne peut modifier I.

Les utilisateurs autorisés à modifier une information ou une ressource sont généralement définis de manière explicite.

2.1.2 Autres formulations des propriétés de sécurité

Il existe une autre façon de définir des propriétés de confidentialité ou d'intégrité, en considérant que celles-ci correspondent à limiter les privilèges des utilisateurs, des machines ou des processus. Ainsi, nous trouvons souvent évoqué les principes du moindre privilège et de la séparation des privilèges.

Moindre privilège

Ce principe peut ainsi être défini comme suit :

Définition 5 (Moindre privilège)**.**

Soit X une entité, P un ensemble de privilèges assignés à X et T un ensemble de tâches attribuées à X. Alors, le principe du moindre privilège sur X pour les tâches T est respecté si tous les privilèges de P sont nécessaires pour réaliser T.

[Saltzer et Schroeder, 1975] ont défini cette notion. En pratique, ce principe revient à limiter les accès au moyen d'une politique. Par exemple, une politique au niveau du système d'exploitation peut limiter les privilèges des processus [Lampson, 1973].

Séparation des privilèges

Selon [Clark et Wilson, 1987, Sandhu, 1990], cette notion se définit comme suit :

Définition 6 (Séparation des privilèges)**.**

Soit O un objet, P un ensemble de privilèges associés à O et X un ensemble d'entités. Alors, le principe de séparation de privilège implique que les privilèges P sur O soient distribués sur l'ensemble des utilisateurs X.

Ce principe permet notamment d'associer des droits différents aux processus devant créer et exécuter des scripts ou des binaires exécutables.

Ces deux principes permettent d'empêcher des violations de confidentialité et d'intégrité en allouant des privilèges clairement définis et distincts aux différentes entités actives du système.

2.2 Modèles de contrôle d'accès

La sécurité des système d'information repose sur la garantie de propriétés de sécurité fondamentales, dont notamment la *confidentialité*, l'*intégrité* et la *disponibilité*. Le contrôle d'accès permet de garantir certaines de ces propriétés par la définition et l'implantation d'une politique de sécurité. Un système d'information peut être vu comme un ensemble de sujets (c'est-à-dire une entité active du système) et d'objets (une entité passive) qui interagissent ensemble afin d'effectuer des actions qui modifieront ou non l'état du système. Une opération effectuée par un sujet sur un objet peut donc être représentée par un triplet *(sujet,action, objet)*.

Il existe deux modèles principaux de contrôle d'accès, que nous détaillerons dans des sous parties distinctes :

- Le **contrôle d'accès discrétionnaire** dans lequel ce sont les utilisateurs qui attribuent les permissions sur les ressources qui leurs appartiennent. Il s'agit du mécanisme utilisé dans les systèmes d'exploitation traditionnels (Linux, Windows, MacOS). Ce modèle est faible en terme de sécurité.

– Le **contrôle d'accès obligatoire** dans lequel l'attribution des permissions est gérée par une entité tierce et dans lequel les utilisateurs du système ne peuvent en aucun cas intervenir dans l'attribution des permissions.

De plus, ces deux modèles utilisent souvent la notion de rôles afin de factoriser les règles. Un rôle représente un ensemble de droits d'accès. Les utilisateurs se voient attribués certains rôles et héritent ainsi des droits d'accès associés à ces rôles, ce qui factorise les règles des différents utilisateurs.

2.2.1 Contrôle d'accès discrétionnaire

Le contrôle d'accès discrétionnaire (ou *DAC*) est le modèle de contrôle d'accès de base utilisé par l'ensemble des systèmes d'exploitation actuel. Dans ce modèle de contrôle d'accès, c'est l'utilisateur propriétaire de ses ressources qui décide des droits d'accès accordés aux autres utilisateurs sur ses ressources.

Une première formalisation de ce modèle de contrôle d'accès est proposé par l'auteur de [Lampson, 1969]. Celui-ci y propose notamment les définitions de capacités et d' *Acces Control List* (ou ACL). Lampson propose de placer dans une matrice l'ensemble des domaines de protection qui représentent des contextes d'exécution (les sujets) sur les lignes et l'ensemble des objets sur les colonnes. Ces définitions sont affinées dans [Lampson, 1971] où l'auteur propose la modélisation des accès sous forme de *matrice d'accès*.

Le modèle *HRU* ([Harrison *et al.*, 1976]) propose une amélioration du modèle de contrôle d'accès discrétionnaire de Lampson. Dans ce modèle, les auteurs proposent une description des politiques de contrôle d'accès discrétionnaire. Une matrice P représente l'ensemble des droits d'accès des sujets sur des objets. Chaque sujet possède des droits d'accès sur des objets, mais les sujets sont eux-mêmes des objets et peuvent modifier la matrice de contrôle d'accès dans le but d'ajouter ou supprimer des objets, mais également modifier les droits accordés à un autre sujet.

Cependant, HRU établit qu'on ne peut garantir de sécurité dès lors que l'on peut modifier librement les règles.

Sandhu introduit dans [Sandhu, 1992] le modèle *Typed Acces Matrix* (ou *TAM*) dans lequel l'auteur propose une extension du modèle HRU en introduisant la notion de typage fort. Cette notion de typage fort est une extension des travaux précédents sur *SPM* (Sandhu's Schematic Protection Model) de [Sandhu, 1988]. Cependant, il ne résout pas l'impossibilité de garantir des propriétés de confidentialité ou d'intégrité avec l'approche discrétionnaire.

2.2.2 Contrôle d'accès obligatoire

A l'inverse, les modèles de contrôle d'accès obligatoire reposent sur le principe que la politique de sécurité appliquée sur un système ne doit pas pouvoir être modifiée par les utilisateurs du système. Pour cela, Anderson propose dans [Anderson, 1980] d'utiliser un *moniteur de référence*. Ce moniteur de référence doit permettre d'appliquer sur le système une politique de sécurité (modélisée sous la force d'une matrice fixe) qui détaille explicitement tous les accès autorisés ou interdits au sein du système. Avec cette approche, il est possible de garantir des propriétés de confidentialité et d'intégrité. Sur cette base, différents modèles ont été proposés.

Bell-Lapadula

Le modèle de *Bell-Lapadula* (BLP), défini dans [Bell et LaPadula, 1973] formalise la propriété de confidentialité des données dans les milieux militaires. Ce modèle associe la notion de *label* aux sujets et objets. Un label représente un niveau de sécurité contenant deux types d'identifiant de sécurité : un niveau hiérarchique et un identifiant de catégorie. Ces niveaux de sécurité permettent de garantir deux règles :

- **No Read Up** : Cette propriété assure qu'un sujet qui demande un accès en lecture sur un objet du système possède un niveau de sécurité supérieur ou égal à celui de l'objet.
- **No Write Down** : Cette propriété assure qu'un sujet qui demande à modifier un objet du système possède un niveau de sécurité inférieur ou égal à celui de l'objet.

Biba

Le modèle Biba [Biba, 1977] formalise le dual de Bell et Lapadula pour garantir la propriété d'intégrité d'un système. Dans ce modèle, chaque sujet et objet possède un label. Deux propriétés sont ainsi garanties :

- **No Read Down** : Cette propriété assure qu'un sujet qui demande un accès en lecture sur un objet du système possède un niveau d'intégrité inférieur ou égal à celui de l'objet.
- **No Write Up** : Cette propriété assure qu'un sujet qui demande un accès en écriture sur un objet du système possède un niveau d'intégrité supérieur ou égal à celui de l'objet.

Ces propriétés permettent d'éviter les transferts d'informations d'un niveau d'intégrité bas vers un niveau d'intégrité haut ce qui entrainerait une compromission de l'intégrité du niveau haut.

Domain and Type Enforcement

Le modèle *Domain and Type Enforcement* (DTE) défini dans [Boebert et Kain, 1985] est un modèle de contrôle d'accès basé sur une abstraction des ressources du système et créé spécifiquement pour l'écriture de politiques de sécurité. À la différence des modèles BLP et Biba qui supportent seulement deux cas particuliers de *confidentialité* et d'*intégrité*, le modèle DTE couvre un plus large ensemble de propriétés. Dans ce modèle, les notions de sujets et d'objets sont remplacées par celles de *domain* et de *type* dans le but de distinguer les entités actives et passives du système. Ce modèle autorise les interactions entre un domaine et un type, mais également les interactions entre domaines. Dans ce modèle, il est nécessaire de définir de manière explicite tous les accès autorisés aux différents domaines. Cela permet une application efficace du principe de moindre privilège car les accès d'un domaine peuvent être restreints aux ressources qui lui sont nécessaires.

2.3 Implantation des mécanismes de contrôle d'accès

Dans cette partie, nous nous intéressons aux mécanismes de contrôle d'accès obligatoires existants aux différents niveaux, à savoir système d'exploitation, intergiciel, Web et workflows. Cependant, nous nous intéressons plus particulièrement au cas des environnements Microsoft puisqu'ils correspondent à notre cible expérimentale.

2.3.1 Contrôle d'accès au niveau système d'exploitation

À partir des systèmes d'exploitation *Vista*, Microsoft a implanté un modèle de protection obligatoire basé sur le dépôt du brevet [Ward et Hamblin, 2006]. Cette implantation, nommée *Mandatory Integrity Control* (MIC), est mise en place pour protéger le système et les fichiers système des attaques visant à porter atteinte a leur intégrité. Dans ce modèle, inspiré du modèle Biba, chaque entité du système possède un niveau d'intégrité [Microsoft, 2013c]. Il existe par défaut cinq niveaux d'intégrité :

- **Untrusted** : niveau de non confiance, notamment pour les sessions invitées.
- **Low** : niveau utilisé pour les processus fortement exposés aux attaques externes.
- **Medium** : niveau par défaut pour les utilisateurs authentifiés.
- **High** : niveau réservé à l'administrateur du système.
- **System** : niveau réserve au système.

Un sujet désirant modifier un objet doit posséder un niveau d'intégrité supérieur ou égal à celui de l'objet. Il s'agit de la propriété *No Write Up* du modèle Biba. Contrairement au modèle Biba, la propriété *No Read Down* n'est pas implantée.

Microsoft a également implanté une notion de labels sur certains objets du système, qui agissent de façon complémentaires avec les niveaux d'intégrité. Ainsi, trois propriétés sont garanties :

- **No Read Up** : les sujets possédant un label *strictement* supérieur à celui de la cible ne peuvent pas lire l'objet cible. Il s'agit d'une application restreinte du modèle BLP.
- **No Write Up** : les sujets ayant un label *strictement* inférieur à celui de l'objet ne peuvent pas le modifier.
- **No Execute Up** : empêche l'exécution d'objets par les sujets ayant un niveau d'intégrité bas.

Des exceptions existent cependant : les utilisateurs autorisés à effectuer des tâches d'administration peuvent modifier leur niveau d'intégrité dans le but de réaliser des actions privilégiées.

À la différence des implantations de modèles de contrôle d'accès obligatoire sous Linux tels que *SELinux* ou *grsecurity* [Splengler, 2002], le contrôle d'accès obligatoire MIC est effectué avant le contrôle d'accès discrétionnaire. Cela permet à un utilisateur élevant son niveau d'intégrité de contourner les droits d'accès classiques.

SELinux est un modèle de contrôle d'accès obligatoire créé par la NSA et intégré nativement dans les noyaux Linux sous la forme d'un *Linux Security Module* (LSM) [Wright *et al.*, 2002], qui sont des interfaces de sécurité implantées dans le noyau et permettant de détourner de manière légitime le flux d'exécution pour y appliquer un mécanisme de contrôle d'accès. Il s'appuie sur le modèle *Domain and Type Enforcement* et son architecture est basée sur FLASK [Spencer *et al.*, 1999].

Grsecurity est un mécanisme de contrôle d'accès obligatoire qui, à la différence de SELinux, n'utilise pas les LSM mais est disponible sous la forme d'un patch noyau. Celui-ci est basé sur les modèles *RBAC* et *Trusted Path Execution* (TPE).

À la différence du modèle MIC de Microsoft, ces approches pour Unix nécessitent des politiques de sécurité fines mais offrent une meilleure garantie [Gros *et al.*, 2012].

2.3.2 Contrôle d'accès au niveau intergiciel

Common Language Runtime de .NET

Le Framework .Net fournit un environnement d'exécution de code nommé *Common Language Runtime* (CLR) [Microsoft, 2013b]. Les outils de développement compatibles

avec le CLR produisent un code source dans un langage intermédiaire appelé *Common Intermediate Language* (CIL, anciennement MSIL), sur le même principe que Java .

À la différence de Java, ce code intermédiaire n'est pas interprété [Lindholm *et al.*, 2013]. Celui-ci est compilé à la volée pour produire le code machine compatible avec la plateforme cible lors de l'exécution.

L'architecture .Net est essentiellement basée sur l'utilisation d'*assemblages*. Un assemblage représente une collection logique contenant le code source et les ressources d'une application. Il contient également un manifeste, c'est-à-dire une description du code et des ressources de cet assemblage.

Mécanismes de sécurité de .NET

Jusqu'à la version 3.5 du framework .Net, le CLR assure la sécurité de chaque assemblage par une approche obligatoire. L'administrateur doit définir, pour chaque assemblage, la stratégie de sécurité associée à cet assemblage, c'est-à-dire les privilèges qui lui sont attribués [Platt, 2001]. Ces règles sont définies au sein d'un fichier XML au niveau de la machine. L'administrateur définit les permissions attribuées aux assemblages en créant des groupes de privilèges accordés ou interdits comme un tout. Même si la configuration par défaut propose plusieurs groupes de permissions et qu'il est possible pour l'administrateur d'en créer des nouveaux, en pratique ce système n'est jamais utilisé par les administrateurs car il est extrèmement complexe. De plus, il n'existe pas d'outils permettant de définir de manière simple les groupes et les permissions à accorder aux assemblages. Ainsi, ce mécanisme fut abandonné par Microsoft pour les versions 4.0 et supérieures du framework .Net.

Dans la version 4.0 du Framework .Net, Microsoft a introduit un nouveau modèle de protection appelé *transparence de niveau 2* [Microsoft, 2013a]. La transparence est un mécanisme permettant de distinguer le code s'exécutant dans le cadre de l'application et le code qui s'exécute dans le cadre de l'infrastructure. Elle établit un cloisonnement entre le code qui peut réaliser des actions privilégiées (code critique) et le code qui ne le peut pas (code transparent).

La transparence de niveau 2 repose sur trois composants [Microsoft, 2013a] :

- **Le code transparent** peut appeler du code transparent ou du code non critique. Il peut être de confiance totale ou partielle.
- **Le code critique sécurisé** a un niveau de confiance totale mais peut être appelé par du code transparent. Il expose au code transparent une surface limitée. Les vérifications d'exactitude et de sécurité ont lieu dans le code critique sécurisé.
- **Le code critique** Le code critique de sécurité peut appeler tout code et a un niveau de confiance totale, c'est-à-dire qu'il possède tous les droits mais il ne peut pas être appelé par du code transparent.

Les interactions entre les trois niveaux de confiance sont représentées dans la figure 2.1.

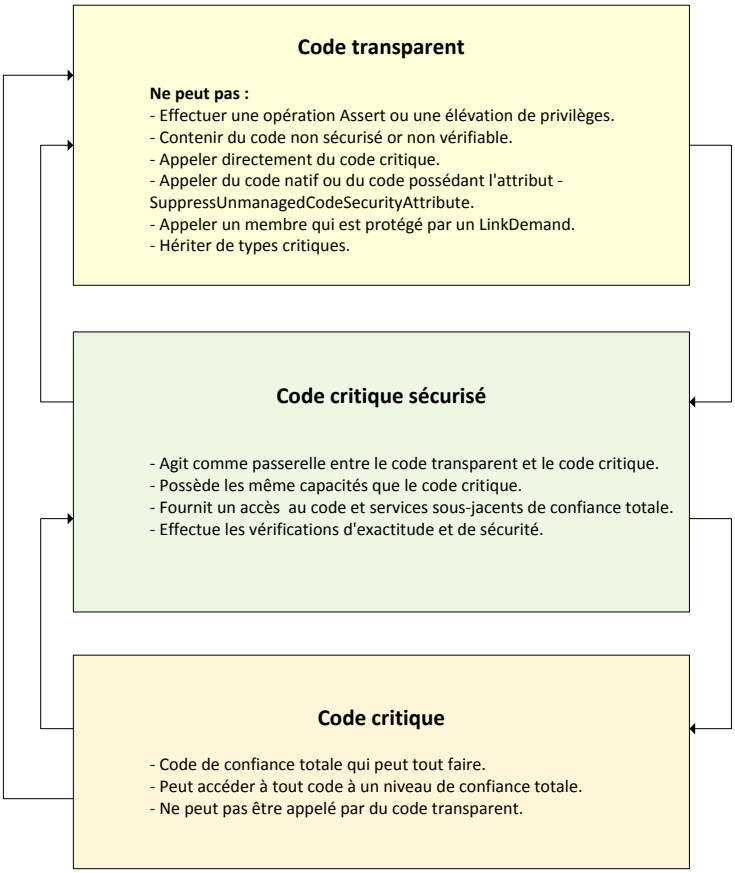

FIGURE 2.1 – Interactions entre les niveaux de confiance .Net.

À l'inverse de .Net, Java [Lindholm *et al.*, 2013] ne propose ni mécanisme de protection obligatoire tel que défini pour .Net 3.5, ni d'approche de cloisonnement par programmation tel que défini dans .NET 4.0. Pour Java, JAAS offre la possibilité aux programmeurs d'exprimer des règles de contrôle d'accès proches de .NET 3.5. Cependant, ce mécanisme est sous le contrôle des développeurs et peut être facilement contourné [Nair *et al.*, 2008, Venelle *et al.*, 2013].

2.3.3 Contrôle d'accès au niveau application Web

Différentes approches existent pour contrôler l'accès à une application hébergée sur un serveur Web.

Tout d'abord, les serveurs Web proposent des mécanismes d'administration pour authentifier les utilisateurs et restreindre l'accès à certaines ressources. Ces méthodes existent aussi bien pour les serveurs Unix [Bouchaudy, 2013] que Windows IIS [Thobois, 2013].

Une seconde méthode existe au niveau des environnements de programmation Web. Par exemple, avec les environnements Microsoft, un mécanisme d'autorisation d'accès aux ressources des applications Web existe. Celui-ci permet des règles d'autorisations ou de refus d'accès sur les pages aspx d'une application Web.

Cette configuration est effectuée au sein du fichier sous format XML *Web.config*. les éléments <allow> et <deny> permettent d'autoriser ou d'interdire les utilisateurs et les rôles pour les pages présentes dans le répertoire contrôlé par le fichier web.config. Des attributs spécifiques comme par exemple *verb* permettent d'autoriser la lecture d'une page mais d'interdire par exemple l'envoi d'un formulaire. L'attribut *location* permet de spécifier

un fichier en particulier. Lorsqu'aucune règle ne s'applique, la configuration par défaut qui autorise toutes les requêtes s'applique. Le listing 2.1 présente un exemple d'utilisation de ces règles d'autorisations.

Pour utiliser ce mécanisme d'autorisation, il est nécessaire d'implémenter et donc de développer pour chaque application un *Membership Provider*. Celui-ci sert d'interface entre le mécanisme de contrôle d'accès de .Net et l'application et permet à l'application de fournir au contrôleur le rôle associé à l'utilisateur. Il fournit un certain nombre de méthodes et de propriétés qui seront utilisés de manière transparente par le mécanisme de contrôle d'accès.

```
1 <?xml version="1.0"?>
2 <configuration>
3   ...
4  <authorization>
5    <allow users="Bob" roles="Doctors" />
6    <deny users="*" />
7  </authorization>
8 </configuration>
```

Listing 2.1 – Exemple de configuration d'autorisation .Net

Un certain nombre de limitations existe pour ces solutions. D'une part, elles nécessitent un effort de programmation pour la récupération des notions d'utilisateur et de rôles. D'autre part, elles nécessitent un effort d'administration ou de développement conséquent car toutes les règles d'autorisations doivent être définies manuellement. Enfin, elles ne constituent pas une réelle approche obligatoire puisqu'il n'y a pas de politique globale de protection, celle-ci étant définie par une multitude de fichiers de configuration disséminés sur l'ensemble du système de fichiers. En effet, comme le montre [Noseevich et Petukhov, 2011], il est nécessaire de tester les applications Web pour détecter les failles liées au contrôle d'accès discrétionnaire appliqué en l'absence de méthode obligatoire.

XACML

Le consortium OASIS, qui standardise des approches liées aux applications Web, propose la méthode de contrôle d'accès *XACML* [OASIS, 2013]. Cette méthode est tout à fait extensible. Elle peut être vue comme un méta-langage de contrôle d'accès qui peut être appliqué au niveau des serveurs web mais aussi plus largement par exemple pour contrôler les accès des processus aux fichiers. Ainsi, [Betgé-Brezetz *et al.*, 2013] montre une utilisation de XACML pour offrir un contrôle d'accès proche de SELinux. Cependant, le niveau de complexité et la verbosité des politiques XACML sont importants et nécessitent en pratique [Hu *et al.*, 2007, Masi *et al.*, 2012, Liu *et al.*, 2011, Kolovski *et al.*, 2007] des outils de vérifications syntaxiques et sémantiques complexes uniquement pour valider la conformité des fichiers de politique. En pratique, XACML est peu utilisé dans les environnements Web. Enfin, il n'y aucune méthode pour calculer automatiquement ces politiques [Masi *et al.*, 2012].

SOA

Service-Oriented Architecture (*SOA*) [OASIS, 2012] vise à pallier le manque de modèle pour spécifier les applications Web. Il propose principalement les notions d'application, services, fonctionnalités et ressources Web. Il ne présuppose pas d'outils pour la mise en œuvre. Ainsi, SOA peut reposer sur des services Web SOAP [Nielsen *et al.*, 2003], REST

[Fielding, 2000] voire des intergiciels indépendants du Web. Cependant, SOA n'adresse pas la façon de contrôler les accès et offre encore moins d'approche de calcul de politiques. SOA reporte le problème du contrôle d'accès sur des standards comme XACML [Kim, 2009].

OAuth

Le standard OAuth propose un protocole d'autorisation [Group, 2012] qui permet un échange de jetons d'accès entre un client et un serveur Web. Cependant, ce standard ne définit pas de langage de contrôle d'accès pour exprimer les règles et ne propose pas d'approche de contrôle d'accès obligatoire.

[Cherrueau *et al.*, 2013] vise à proposer une approche obligatoire liée à OAuth. Cependant, l'approche repose sur un langage qui traite essentiellement d'aspects cryptographiques [Dell'Amico *et al.*, 2012]. De plus, il ne propose aucune mise en œuvre pour des applications Web et ne permet pas de calculer automatiquement les politiques.

2.3.4 Contrôle d'accès au niveau Workflows

Nous allons tout d'abord décrire les normes existantes en matière de workflows, et nous montrerons que le contrôle d'accès dans les workflows repose essentiellement sur deux approches, l'une basée sur la classification des workflows en terme de sécurité, et l'autre sur des mises en œuvre de contrôle d'accès.

Standardisation

Différentes normes existent en matière de workflows. XPDL [WFMC, 2012] permet de décrire des workflows dans un format XML. BPMN [(OMG), 2011] offre une standardisation de la représentation graphique des éléments d'un workflow. BPEL [OASIS, 2007] est un langage d'exécution des workflows au moyen d'une orchestration de services Web SOAP. Aucun de ces standards n'adresse le problème du contrôle d'accès. Cependant, [Debricon *et al.*, 2009] propose de modéliser des scénarios de tests d'applications au moyen de BPMN et BPEL. Ces scénarios de tests pourraient éventuellement être projetés sur des modèles de contrôle d'accès si ceux-ci étaient disponibles dans les standards existants.

Classification des workflows

Dans [Cvrcek, 2000], l'auteur propose une méthode abstraite permettant de catégoriser les workflows en fonction de deux critères de sécurité :

- La sensibilité des données : Soit seules les données traitées par les workflows sont classifiées, soit la définition du workflow en lui-même est également classifiée.
- La sécurité du système d'information sur lequel les workflows s'exécutent.

Cette approche est une extension du modèle de classification des workflows présentée dans [Huang, 1998] qui définit quatre niveaux d'exécution des workflows en fonction du niveau de sécurité qu'il garantit. Cependant, l'approche est en pratique difficilement utilisable. En effet, le but de cette classification est d'appliquer aux workflows des modèles de contrôle d'accès obligatoire de type Biba et BLP, mais les auteurs ne proposent pas de mise en œuvre de ces modèles.

Dans [Olivier *et al.*, 1998], les auteurs proposent de séparer les besoins en termes de sécurité dans les workflows en trois catégories :

- **Niveau 1** : Contrôler les accès aux données utilisées pour chaque étapes d'un workflow.

- **Niveau 2** : Contrôle temporel des accès. Quand un utilisateur est autorisé à accéder aux données d'une étape d'un workflow, l'accès doit être autorisé uniquement pendant le traitement de l'étape.
- **Niveau 3** : Définitions de besoins de sécurité au niveau applicatif.

Ce cadre est ouvert et intéressant mais n'offre pas de réponse précise au problème du contrôle d'accès.

Mise en œuvre de contrôle d'accès

Un modèle de contrôle d'accès basé sur les rôles et dédié aux workflows est présenté dans [Wainer *et al.*, 2001]. Ce modèle *W-RBAC* s'appuie sur le modèle de définition des workflows pour définir des rôles associés aux utilisateurs lors de l'exécution des workflows. Il permet également de définir des contraintes statiques et dynamiques sur les rôles utilisés par les utilisateurs. Par exemple, une contrainte statique peut exprimer le fait qu'aucun utilisateur d'un workflow ne peut posséder le rôle de demandeur et d'approbateur d'un workflow. Au contraire, une contrainte dynamique permet d'exprimer le fait qu'un utilisateur ne peut être l'approbateur d'une demande qu'il a effectuée, mais peut être l'approbateur d'une demande effectuée par un autre utilisateur.

Ce modèle ne fonctionne que pour les workflows dont la structure est statique. Une amélioration pour les workflows adaptatifs, c'est-à-dire dont la structure du workflow est susceptible de varier au cours du temps est proposée dans [Leitner *et al.*, 2011]. Cependant, ces approches ne concernent pas les workflows basés sur le Web. Elles ne constituent pas des approches obligatoires et ne permettent pas de calculer les politiques de contrôle d'accès à partir du workflow.

Les autres solutions telles que [Koshutanski et Massacci, 2003, Atluri et Huang, 1996] proposent de décrire les règles d'accès en XACML ou à l'aide de réseaux de Pétri. Elles présentent donc les limitations liées à XACML, ne sont pas directement applicables à des environnements Web et n'offrent pas un calcul automatisé des politiques.

2.4 Discussion

L'état de l'art montre la nécessité d'un contrôle d'accès obligatoire pour garantir des propriétés de confidentialité et d'intégrité. Les méthodes fonctionnant purement au niveau système comme SELinux ou au niveau intergiciel comme .NET sont peu adaptées pour réaliser des contrôles au niveau des serveurs d'application Web car elles ne sont pas capables d'analyser les requêtes HTTP pour trouver la notion de sujet accédant à un objet de l'application Web. Par ailleurs, les contrôles réalisés par programmation (ASP.NET) au niveau des serveurs d'application Web sont fragiles. De plus, les méthodes reposant sur l'administration des serveurs Web (IIS, Apache, etc.) ne sont pas obligatoires et nécessitent les compétences de l'administrateur du serveur Web. La *complexité des standards existants* (XACML) ou leur limitation à un protocole d'échange de clé d'accès (OAuth), montrent l'absence d'un langage efficace pour exprimer des règles de contrôle d'accès pour le Web. De plus, ces méthodes ne permettent pas de calculer automatiquement les règles de contrôle d'accès requises par les applications Web hébergées sur le serveur.

La raison essentielle de ce manque d'approche de contrôle d'accès obligatoire pour le Web est que celui-ci ne propose pas de modèle d'application. Le standard SOA essaie de pallier à ce manque et propose un modèle d'architecture d'application. Cependant, ce standard ne définit pas de modèle de protection. De plus, il peut exister un large ensemble

d'autres modèles d'applications basés par exemple sur REST et sur une organisation des composants logiciels différente de SOA. Nous voyons donc se dégager la nécessité de proposer un modèle d'architecture d'application Web qui s'applique de façon souple au caractère flexible et extensible du Web. Ce modèle d'architecture, qui vise à décrire les objets de l'application, doit permettre de définir un modèle de contrôle d'accès avec des sujets, associés aux requêtes HTTP, accédant aux objets de l'application. De cette façon, nous pourrons calculer automatiquement la politique de contrôle d'accès.

De plus, nous avons vu que les modèles de Workflow ne propose pas non plus de modèle de protection. Les modèles de Workflows sont divers, standardisés (XPDL,BPMN) ou dédiés à un environnement de développement. Ainsi, nous voyons là aussi apparaître le besoin d'un modèle de Workflow qui soit suffisamment simple et flexible pour permettre de représenter ces différentes approches. Ce nouveau modèle de Worflow doit pouvoir être projetté automatiquement sur notre modèle d'application afin d'en déduire automatiquement les politiques de contrôle d'accès requises par les Workflow.

Les approches proposées doivent pouvoir s'appliquer de façon large et en premier lieu aux environnements Microsoft et de QualNet.

Chapitre 3

Formalisation du modèle de protection obligatoire pour les serveurs d'applications Web

Étant donné l'absence de modèle général d'application Web prenant en compte tout type d'architecture des composants logiciels présents au niveau du serveur Web, nous allons proposer un modèle permettant ce niveau de généralité. Notre **modèle d'application Web** permet de définir la notion d'objet structuré en trois niveaux (Applications, Entités d'applications et Ressources). Notre modèle utilise également la notion de *relation* pour définir l'organisation et la sémantique entre ces trois niveaux. Nous verrons que cette notion de relation facilite la prise en compte des différents types d'architectures Web.

Nous définissons ensuite une protection obligatoire dynamique qui supporte ce modèle d'applications Web et propose un langage dédié permettant d'exprimer des règles de contrôle d'accès. Elle permet de contrôler précisément les accès, correspondants aux **actions**, des clients, correspondants aux **sujets**, aux différentes ressources, correspondant aux **objets** que peut héberger un serveur d'applications Web. Ainsi, nous pouvons garantir des privilèges du type Sujet->Action->Objet où le sujet a le droit d'effectuer une action donnée sur un objet. Elle s'adapte au caractère extensible du Web en permettant de définir les différents attributs permettant de caractériser de façon adaptée les sujets, les objets et les actions.

Cette protection obligatoire inclut un moniteur de référence qui reçoit les requêtes HTTP et les compare à une politique de contrôle d'accès exprimée dans notre langage de protection dédié. Elle est adaptée au cas du Web en ce sens qu'elle permet de définir dynamiquement à la réception d'une requête Web, les sujets, les actions requises et les objets associés à la requête. Le moniteur de référence définit dynamiquement toutes les règles potentielles d'accès en fonction non seulement de la requête mais aussi de l'application considérée. Ainsi, il peut par exemple calculer dynamiquement les sujets (l'utilisateur et ses différents rôles), les actions ou privilèges requis (lecture, écriture, exécution, etc.) sur les différents objets tels que des ressources de type pages, session, composants logiciels, etc. Le moniteur de référence compare ensuite les règles potentielles à la politique de contrôle d'accès requise afin d'autoriser ou de refuser l'exécution de la requête.

L'approche de protection proposée permet également le calcul des politiques obligatoires avec deux niveaux d'abstraction.

Tout d'abord, nous proposons un **modèle spécifique d'application** qui est un raffinement du modèle général. Nous montrons que ce modèle spécifique permet de décrire

un large ensemble d'architectures, notamment SOA et les applications QualNet. De plus, il permet de définir un modèle de contrôle d'accès qui permet le calcul des politiques nécessaires. Ce modèle de contrôle d'accès correspond à l'ensemble des règles nécessaires à l'application, à savoir que tous les accès non présents dans le modèle sont interdits.

Ensuite, nous proposons un **modèle spécifique de workflow** qui permet de calculer le modèle spécifique d'application correspondant et de ce fait les politiques requises. L'intérêt est d'offrir un modèle de haut niveau qui facilite la description des workflows en restant simple et extensible à différentes normes telles que BPEL ou XPDL ou différents environnements de workflows tels que le logiciel de Qual'Net. De la même façon que pour le modèle spécifique d'application, nous montrons que différents logiciels de type workflows peuvent être décrits selon notre modèle spécifique de workflow. Les expérimentations montreront qu'il a ainsi permis d'automatiser complètement l'approche notamment pour le système de workflows de Qual'Net.

Enfin, nous proposons une méthode permettant de tester et de vérifier la sûreté de notre protection obligatoire. Cette méthode consiste à tester exhaustivement tous les accès légitimes et illégitimes vis-à-vis de la politique de sécurité et ainsi d'évaluer le taux de sûreté de la protection.

3.1 Modèle général des serveurs d'applications Web

La figure 3.1 présente notre modèle de représentation des serveurs d'applications Web.

Un premier objectif de ce modèle est de décrire ce que contient la requête permettant de caractériser le sujet (l'utilisateur, ses rôles, etc.), l'objet concerné par la requête c'est-à-dire l'application ou les entités d'application ainsi que les ressources requises et enfin le type d'action demandée.

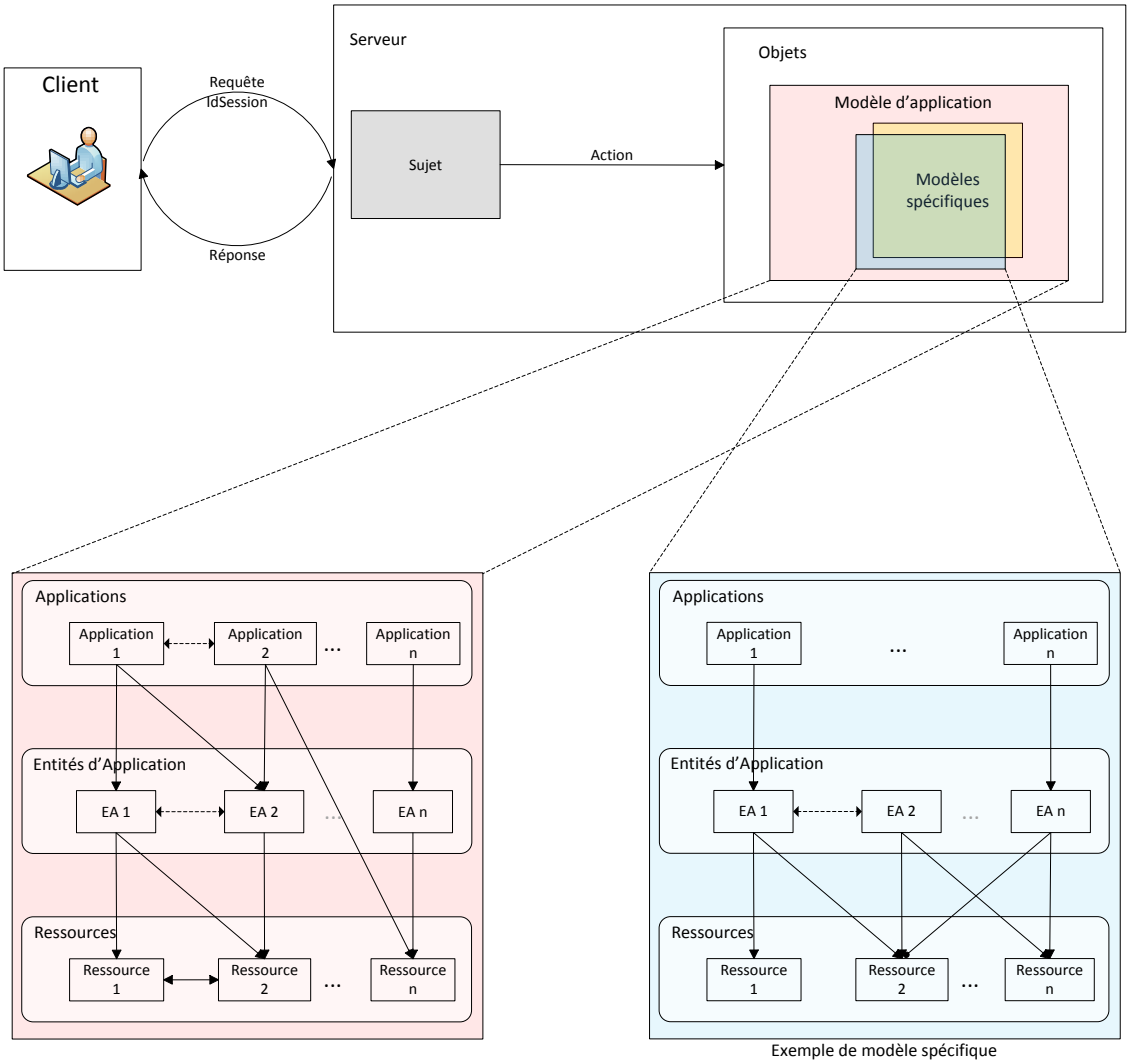

FIGURE 3.1 – Modèle général des serveurs d'applications Web.

Un second objectif est de permettre de prendre en compte la structuration d'un ensemble d'applications. D'une part, notre modèle général permet de décrire les liens entre applications. D'autre part, il permet de définir la structuration interne de chaque application indépendamment des autres applications en précisant les différentes entités d'application ou ressources qu'elle contient ou utilise. Ce modèle général d'application autorise tous types de relations entre applications, entités et ressources. Nous verrons que la formalisation est simplifiée car à ce niveau nous ne définissons que des ensembles correspondants à des paramètres pour les relations. La sémantique et le rôle précis de ces relations doit être définie au moyen d'un modèle spécifique.

Étant donné que nous voulons supporter différents types d'application Web, un troisième objectif pour notre modèle général est de permettre la projection de modèles spécifiques d'applications Web sur ce modèle général. Sur le même principe, des modèles spécifiques de workflow doivent pouvoir être projetés sur les modèles spécifiques d'application.

Un modèle spécifique d'applications Web définit précisément d'une part les relations entre les différents éléments applications, entités d'application et ressources, et d'autre part les actions des sujets nécessaires.

L'intérêt d'un modèle spécifique est de faciliter la définition des privilèges, c'est-à-dire les actions requises par un sujet donné sur les applications, les entités d'application et les ressources, puisque les relations sont clairement définies.

Nous proposerons donc un modèle spécifique de structuration d'application et nous montrerons que ce modèle spécifique se projette bien dans le modèle général. Nous proposerons enfin un modèle spécifique de workflow et nous montrerons que celui-ci se projette bien sur notre modèle spécifique d'application tout en supportant un large ensemble de solutions de workflow dont celle de Qual'Net.

Nous voyons déjà globalement que les différentes notions de sujet, d'objet, d'action et de modèle général d'application doivent être les plus extensibles possibles afin de supporter un large ensemble de modèles spécifiques d'applications et de Workflows.

3.1.1 Sujet d'une requête Web

Afin de caractériser le sujet nous devons nous intéresser à ce que contient la requête Web le désignant, puisque selon le principe client-serveur celui-ci ne peut-être défini que par la requête HTTP.

Identifiant de session

Le Web reposant sur un protocole (HTTP) en mode non connecté et sans état, les applications ou les serveurs Web mettent généralement en place un mécanisme d'authentification. L'authentification est hors du cadre de notre étude. Cependant, une fois l'authentification réalisée, l'application est contrainte de renvoyer un **identifiant de session** pour éviter à l'utilisateur d'avoir à s'authentifier à nouveau (Single Sign On).

En pratique, l'identifiant de session est le seul mécanisme permettant de faire du Single Sign On et ce mécanisme est donc présent dans quasiment toutes les applications (voir figure 3.2). Nous faisons donc l'hypothèse d'un identifiant de session ou d'un nom d'utilisateur dans la requête.

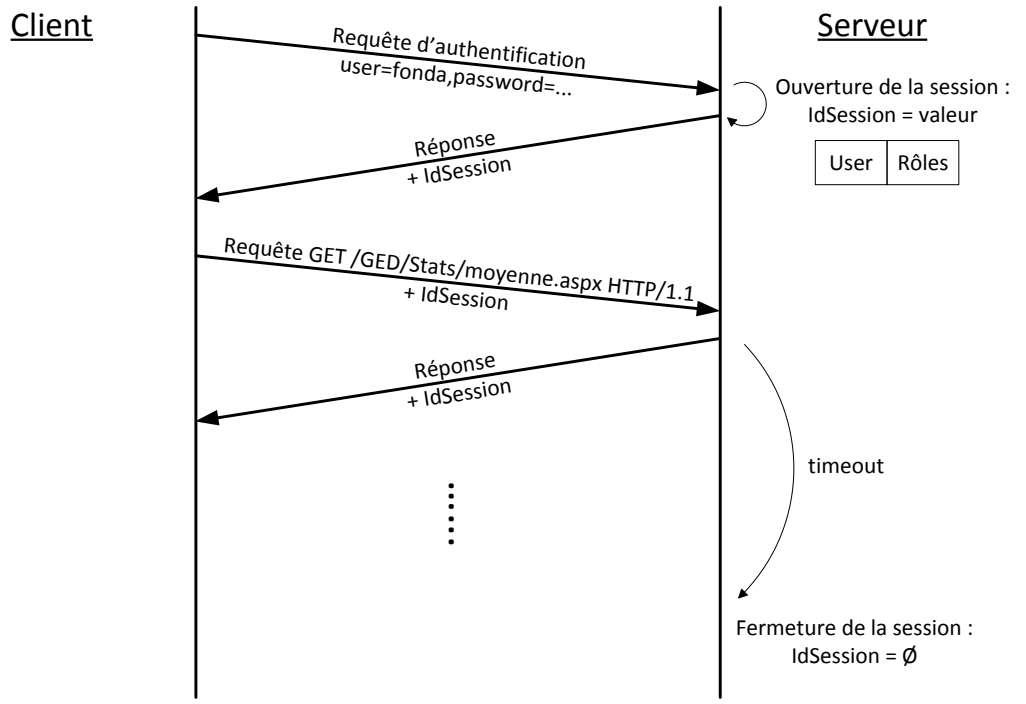

FIGURE 3.2 – Mécanisme de gestion des sessions.

Désignation du sujet

Nous verrons par la suite que nous prenons bien en considération le fait que la requête transporte soit directement l'utilisateur faisant la requête soit un identifiant de session qui permet de retrouver l'utilisateur concerné.

Nous le voyons en pratique, la requête permet de caractériser le sujet en tant qu'utilisateur. Mais le sujet doit pouvoir être défini par d'autres éléments de la requête. Par exemple, la localisation géographique (France, Chine, etc.) ou le type de machine cliente (Windows, Android, etc.).

De part le caractère versatile du Web, le sujet doit ainsi pouvoir être caractérisé par un ensemble d'attributs dont l'identité de l'utilisateur, voire ses rôles, sa localisation, etc. Notre modèle de protection doit supporter ces différents attributs en imposant éventuellement un ensemble prédéfini d'attributs tels l'utilisateur et le rôle. Le fait de ne pas limiter la notion de sujet à des attributs spécifiques offre un caractère d'extensibilité à notre modèle de protection qui correspond bien à la versatilité et l'adaptabilité du Web.

Si certains attributs désignant le sujet peuvent être présents directement dans la requête (par exemple l'identité de l'utilisateur), d'autres doivent pouvoir être calculés indirectement à partir de la requête sur le serveur (par exemple l'attribut de rôle). De façon plus générale, nous considérons que notre modèle de protection doit permettre de calculer indirectement tous les attributs qui permettent de désigner le sujet, ce qui impose un modèle de protection où le sujet est calculé dynamiquement.

3.1.2 Désignation de l'objet

L'objet est lui aussi désigné par la requête. Cependant, cet objet est hébergé par le serveur d'applications.

A la différence du sujet, la désignation de l'objet est dépendante de deux niveaux de structuration et des relations à l'intérieur et entre ces deux niveaux. Au niveau le plus haut, l'objet dépend de la façon dont les applications sont architecturées en terme d'entités d'application, c'est-à-dire de sous parties de l'application ayant des liens entre elles. Au niveau le plus bas, l'objet correspond à des ressources utilisées par ces entités d'application.

En effet, il n'existe pas dans le Web de notion d'objet même si certains standards tels que REST ou SOA introduisent ou reposent sur la notion d'objets. Cependant, il est indispensable d'introduire une notion d'objet incluant à la fois un niveau haut de structuration des applications et un niveau bas de ressources nécessaires. L'idée forte du modèle d'application proposé est de pouvoir désigner les objets en autorisant différents modèles de structuration d'application et différents types de ressources au niveau le plus bas.

Modèle général d'application

Nous proposons un modèle général pour les applications Web qui permet de désigner les objets en reposant sur trois niveaux (applications, entités d'application et ressources). Ce modèle décrit également les liens entre et à l'intérieur de ces niveaux. Pour cela, notre modèle fait apparaître la notion de *paramètres* permettant de définir des *relations* au niveau des modèles spécifiques d'applications entre un ensemble d'applications A, un ensemble d'entités d'application EA et un ensemble de ressources R. Ces relations correspondent à une sémantique que donne un modèle spécifique d'application à un ensemble de paramètres. Nous verrons des exemples de sémantiques par la suite. Les paramètres correspondent à une union d'un ensemble d'applications A, d'un ensemble d'entités d'application EA et d'un ensemble de ressources R. Ainsi, nous décrivons les liens fonctionnels et pouvons associer une sémantique à ces liens entre les applications, les entités de ces applications et les ressources. L'intérêt est de supporter différents modèles spécifiques d'application associant chacun une sémantique particulière aux paramètres. Ainsi, nous pouvons non seulement proposer des modèles spécifiques d'application comme SOA incluant les notions d'applications, de services et de fonctionnalités ou des modèles incluant seulement les notions d'applications et de composants, mais aussi définir une sémantique pour les liens entre ces différentes entités d'application par exemple une sémantique pour les relations entre applications, services et fonctionnalités de SOA.

Paramètres

Nous proposons ici une notion de paramètres correspondant à une liste non bornée d'applications, d'entités d'application et de ressources :

Définition 7 (Paramètres)**.**

Soit A l'ensemble des applications du serveur,
EA l'ensemble des entités d'application du serveur et
R l'ensemble des ressources du serveur.

$$Paramètres = \{A_1, \ldots, A_n, EA_1, \ldots, EA_m, R_1, \ldots, R_p\} \ /$$
$$\forall i \in [1 \ldots n], \forall j \in [1 \ldots m], \forall k \in [1 \ldots p], A_i \in A, EA_j \in EA, R_k \in R$$

Pour illustrer la notion de paramètres, prenons un exemple correspondant aux applications $\{Dynamic$ et $GED\}$ de QualNet et aux différentes entités d'application et ressources (figure 3.3). Sur cet exemple, nous pouvons notamment définir les paramètres suivants correspondants à trois relations (descendanteSimple, descendanteMultiRessources et descendanteMultiEntites) sur lesquelles nous reviendrons :

$$params_{descendanteSimple} = \{Dynamic, WF1, apercu.aspx\}$$
$$params_{descendanteMultiRessources} = \{GED, Redaction, saisie.aspx, apercu.aspx\}$$
$$params_{descendanteMultiEntites} = \{Dynamic, WF1, Stats, moyenne.aspx, apercu.aspx\}$$

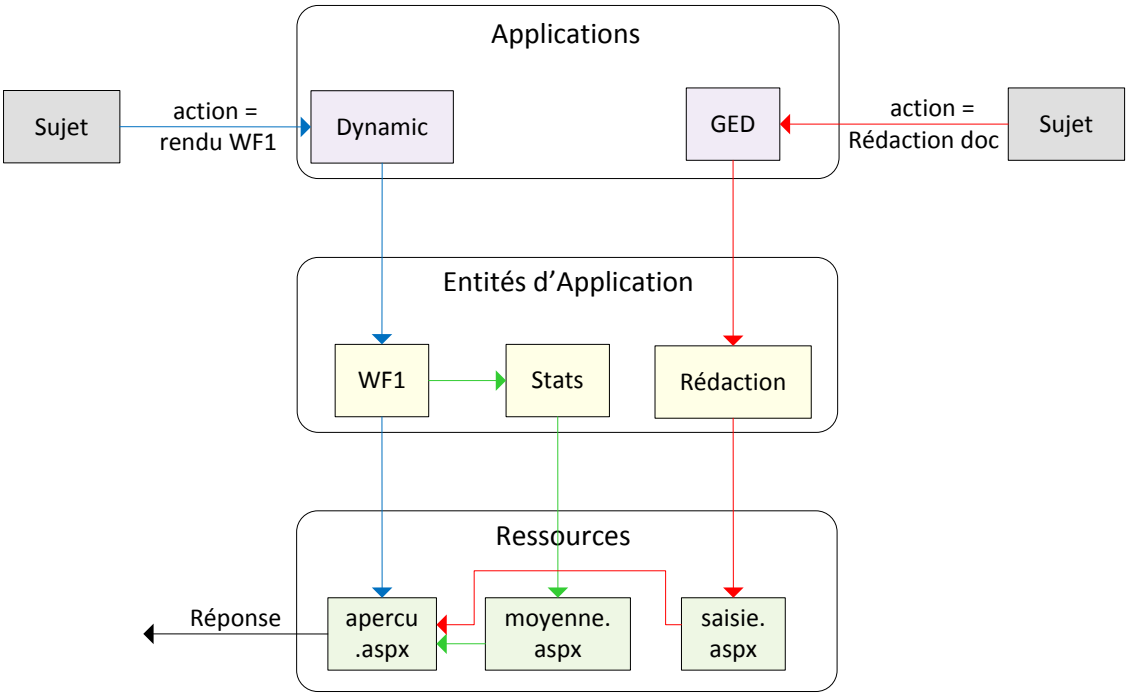

FIGURE 3.3 – Exemple de paramètres et de relations cas d'application.

Relations

Nous proposons une notion de relations qui définit une sémantique pour les différents liens fonctionnels correspondants à un des modèles spécifiques d'application existant sur le système. Une relation porte sur des paramètres c'est-à-dire un ensemble d'applications, d'entités et de ressources. En pratique, une relation peut être vue comme un appel à une fonction ayant une sémantique particulière dans un modèle spécifique.

Définition 8 (Relations)**.**

Soit Paramètres l'ensemble des paramètres du serveur,
ModèlesSpécifique les modèles spécifiques du serveur.
L'ensemble des Relations du serveur est defini tel que :

$$\forall\, rel \in Relations,\ \exists\, param \in Paramètres,\ \exists\, modSpec \in ModèlesSpécifiques\ /\ rel(param) \in modSpec$$

Pour illustrer la notion de relations, reprenons l'exemple de la figure 3.3. Nous pouvons voir sur cet exemple que l'action de rendu du workflow 1 (WF1) fait intervenir une relation *descendanteSimple* définie par le modèle spécifique dont nous donnons ici une explication simplifiée.

$$descendanteSimple(parametres_{descendanteSimple}) \iff descendanteSimple(Dynamic, WF1, apercu.aspx)$$

Cette relation permet l'exécution de la ressource *apercu.aspx* qui va fabriquer la réponse HTTP correspondante au rendu du workflow WF1 et la retourner au client.

Par ailleurs, dans cet exemple existe aussi une relation *descendanteMultiRessources* intervenant lors de l'action de rédaction d'un document :

$$descendanteMultiRessources(GED, Redaction, saisie.aspx, apercu.aspx)$$

Dans ce cas, la requête arrive jusqu'à la ressource *saisie.aspx* qui exécute la page *apercu.aspx*, en charge de fabriquer la réponse HTTP associée à la visualisation du document lors de sa rédaction.

Enfin, sur cet exemple nous pouvons également définir une relation *descendanteMultiEntites* :

$$descendanteMultiEntites(Dynamic, WF1, Stats, moyenne.aspx, apercu.aspx)$$

Cette relation permet l'affichage de la moyenne d'utilisation du workflow WF1. L'entité d'application *Stats* est utilisé par *WF1*. La ressource *moyenne.aspx* est chargée du calcul de la moyenne d'utilisation du workflow, et enfin la ressource *apercu.aspx* retourne le résultat au client en construisant la réponse HTTP correspondante.

Nous reviendrons sur l'aspect modélisation en définissant un modèle spécifique d'application Web qui correspond bien aux organisations classiques des applications Web et qui s'applique bien aux workflows. Cependant, pour l'heure, nous allons proposer une approche globale de protection qui supporte bien notre modèle général d'application Web.

3.2 Approche globale de protection

Dans cette partie, nous proposons une approche globale de protection obligatoire des serveurs d'application Web. Cette approche doit répondre à deux contraintes principales : la solution de protection obligatoire doit être *suffisamment fine* tout en étant *simple à administrer*. Elle fait donc intervenir deux composants principaux, résolvant chacun une de ces deux contraintes. De plus, notre approche globale de protection obligatoire permet de contrôler les actions des sujets sur les objets tels que définies dans notre modèle général de serveur d'application Web, c'est pourquoi celui-ci a été défini en premier.

Avantage de l'approche

Tout d'abord, notre approche de protection obligatoire permet de contrôler précisément les accès des clients aux objets hébergés par un serveur d'applications Web. Pour cela, nous avons proposé dans la partie 3.1 un modèle de représentation abstrait des serveurs d'applications Web s'adaptant au caractère extensible du Web. Notre approche de protection obligatoire utilise ce modèle pour obtenir une représentation fine des sujets et des objets intervenant dans l'ensemble des requêtes HTTP entrantes sur le serveur d'application Web. La protection obligatoire dynamique est, dans notre approche de protection obligatoire, le composant chargé d'appliquer sur le serveur les règles d'autorisations présentes dans la politique de sécurité.

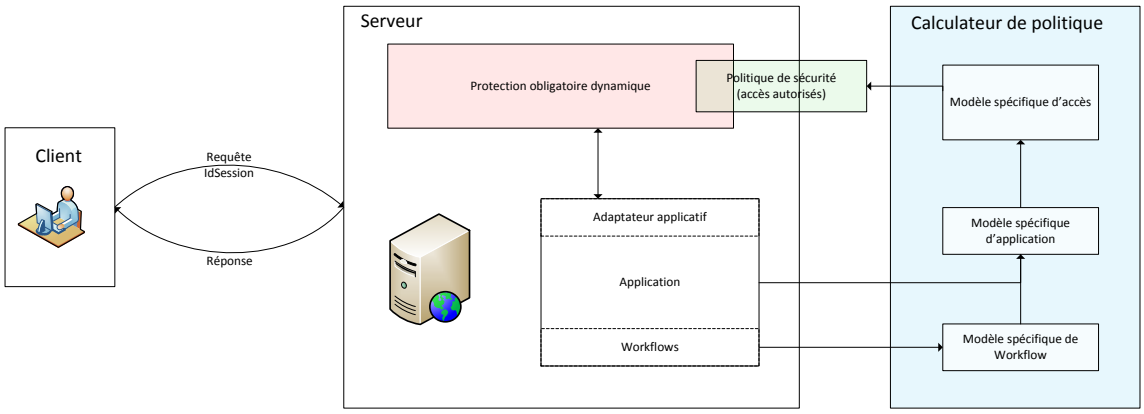

FIGURE 3.4 – Approche globale de protection.

De plus, pour répondre à la contrainte de facilité d'administration de la protection obligatoire, nous proposons une méthode de calcul des politiques de sécurité pour les applications hébergées sur un serveur Web. Cette méthode permet de calculer une politique de sécurité à partir d'un modèle spécifique d'application Web et d'un modèle spécifique de workflows proposés ensuite. Dans notre approche, le calculateur de politique est le composant qui calcule et fournit au mécanisme de protection obligatoire dynamique la politique à appliquer sur le serveur d'applications Web.

La figure 3.4 présente notre modèle global de protection d'un serveur d'applications Web.

3.2.1 Protection obligatoire dynamique

Le rôle principal de notre protection obligatoire est de décider, pour chaque action d'un sujet sur un objet du serveur, si celle-ci est légitime vis-à-vis de la politique de sécurité à

appliquer.

La politique de sécurité appliquée sur un serveur Web représente l'ensemble des accès, d'un sujet sur un objet du serveur, autorisés sur ce serveur. Tout accès non exprimé dans la politique de sécurité est automatiquement rejeté. Les règles d'accès sont exprimées au moyen d'un langage de protection dédié et de bas niveau qui permet de représenter différents types d'accès tout en étant extensible pour s'adapter à des modèles spécifiques d'application Web.

Notre protection obligatoire doit donc fournir un mécanisme permettant d'empêcher qu'une action non autorisée par la politique de sécurité ne soit effectuée. Pour chaque requête HTTP entrante sur le serveur Web, le mécanisme de protection doit déterminer le type d'action requise et doit également définir les contextes de sécurité sujets et objets relatifs à la requête entrante. Ainsi, un ensemble de règles potentielles d'accès est calculé dynamiquement à chaque requête, et celles-ci sont comparées à la politique de sécurité pour décider de l'autorisation ou de l'interdiction de l'accès demandé.

Cependant, notre protection obligatoire est indépendante des applications qu'elle contrôle sur le serveur. Elle laisse libre l'application protégée pour la façon de définir les sujets. Par exemple, ceux-ci peuvent être stockés par les applications dans les sessions utilisateurs, dans le cache applicatif ou encore dans une base de données. Ainsi, notre modèle introduit pour chaque application du serveur un *adaptateur applicatif* qui assurera la liaison entre la protection obligatoire et les applications du serveur.

Adaptateur applicatif

L'adaptateur applicatif est chargé de calculer, pour une requête entrante sur le serveur d'application, les valeurs des paramètres permettant de représenter le contexte de sécurité sujet relatif à cette requête. Celui-ci est fourni par chaque application protégée, qui définit librement sa propre notion de sujet. Il est donc capable de calculer les informations nécessaires au calcul du contexte de sécurité sujet. L'adaptateur applicatif doit fournir des méthodes pour permettre au moniteur de référence d'obtenir les informations lui permettant de calculer le contexte sujet associé à la requête.

Lors d'une requête entrante, le moniteur de référence interroge l'adaptateur applicatif en lui fournissant l'état de la requête entrante. L'adaptateur lui retourne la liste des paramètres identifiant le sujet associé à cette requête. Le moniteur de référence peut calculer les contextes de sécurités sujets associés à cette requête et prendre la décision d'autorisation de l'action requise.

3.2.2 Calculateur de politique

Nous proposons un calculateur de politique de protection obligatoire qui fonctionne en deux étapes.

Tout d'abord, nous définissons un modèle spécifique d'application web qui permet de définir un modèle d'accès spécifique proche du langage d'expression des politiques mais qui impose quatre niveaux de structuration (applications, services, fonctionnalités, ressources) pour l'application contrairement au modèle général qui n'en présente que trois. L'avantage conceptuel de ce modèle d'application spécifique est de pouvoir être projeté sur notre modèle général d'application tout en respectant l'approche proposée par SOA. En ce sens, notre modèle spécifique d'application peut être vu comme une simplification du modèle SOA.

L'intérêt est que, le modèle d'accès spécifique étant proche de notre langage de protection, ce modèle d'accès spécifique permet de calculer automatiquement les règles d'accès tout en restant de haut niveau et donc plus facilement analysable. Ce modèle d'accès spécifique peut être comparé à un langage d'assemblage vis-à-vis d'un langage binaire exécutable qui lui est comparable à notre langage de règles. Il est clairement plus simple d'analyser et d'utiliser un langage d'assemblage que d'analyser et de comprendre un binaire. De même dans notre cas, il est plus simple d'analyser et de comprendre notre modèle d'accès spécifique que les règles d'accès de bas niveau exprimées dans notre langage (proche d'un jeu d'instruction d'un processeur de contrôle d'accès).

Ensuite, nous proposons un modèle d'accès spécifique de workflow qui peut être projeté sur notre modèle d'accès spécifique. L'avantage conceptuel de ce modèle d'accès spécifique de workflow est de respecter les principes rencontrés dans de nombreux logiciels de workflows tout en étant proche de BPEL/XPDL. Ainsi, ce modèle d'accès spécifique de workflow peut être vu comme une simplification de BPEL/XPDL. L'intérêt pratique est de permettre d'automatiser complètement le calcul de la politique pour les systèmes de workflows. Ainsi, nous obtenons à moindre effort une politique de contrôle d'accès obligatoire qui respecte parfaitement le workflow défini. Nous évitons ainsi de nombreuses attaques basées sur le manque de contrôle d'accès et leurs fragilités.

Le calcul des politiques est ainsi automatisé en deux temps. Dans un premier temps, le modèle d'accès spécifique de workflow issu des environnements de workflows est projeté automatiquement sur notre modèle spécifique d'accès. Dans un second temps, notre modèle spécifique d'accès d'application ainsi calculé est projeté automatiquement sur une politique au moyen de notre langage de règles.

3.3 Protection obligatoire dynamique

Nous avons vu dans la partie 3.2 que notre approche de protection globale de serveurs d'application Web doit proposer un mécanisme de contrôle *suffisamment fin*. En premier lieu, il faut donc un langage permettant d'exprimer les règles de sécurité à appliquer sur le serveur qui soit suffisamment *extensible* pour s'adapter à tout type d'application Web. Notre langage est basé sur notre modèle d'applications Web et intègre notamment les notions d'*applications*, d'*entités* et de *ressources* de ce modèle.

Ensuite, notre modèle de protection doit permettre de définir dynamiquement à la réception d'une requête Web les sujets, actions et objets requis. Pour cela, les contextes sujets, les actions et les contextes objets sont calculés dynamiquement à chaque requête, et permettent d'en déduire un ensemble de *règles potentielles d'accès*. Ces *règles potentielles d'accès* sont ensuite comparées à la politique de sécurité par le moniteur de référence pour prendre la décision d'autorisation ou de refus de la requête entrante.

La figure 3.5 présente notre modèle de protection dynamique. Nous retrouvons sur celle-ci le moniteur de référence, qui utilise pour chaque requête entrante sur le serveur la politique de sécurité et l'ensemble des règles potentielles d'accès associées à la requête entrante pour prendre la décision d'autorisation de la requête.

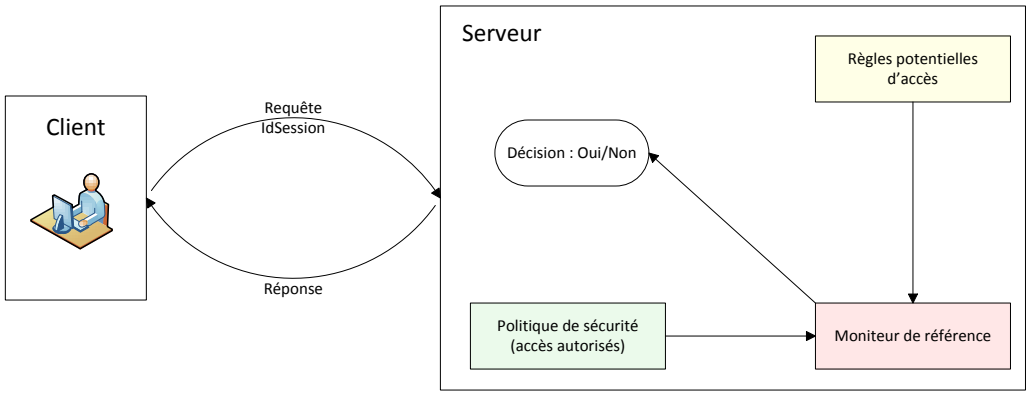

FIGURE 3.5 – Modèle de protection obligatoire dynamique.

3.3.1 Langage de définition de la politique de sécurité

Nous proposons ici un langage permettant de représenter les autorisations que le moniteur de référence devra appliquer au sein du serveur Web. L'ensemble des règles d'autorisations appliquées par le moniteur de référence représente la politique de sécurité de l'application Web considérée. Le listing 3.1 présente la grammaire de notre langage sous la notation EBNF (Extended Backus-Naur Form). Nous détaillerons dans des sous-parties distinctes les notions importantes exprimées dans notre langage.

```
1  politique : (regle'\r\n')*;
2  regle : typeRegle '(' sujet ',' action ',' objet ')' (' : ' condition)*;
3  sujet : userId ':' roleId (':' Chaine)*;
4  action : typeAction;
5  objet :  listeParametre;
6  condition : conditionObjet '("' Chaine '")' Operateur ' ' valeur;
7
8  listeParametre : (parametre ';')*;
9  parametre : (application|entite|ressource);
10 application : typeApplication ':' nomApplication (':' Chaine)*;
11 entite : typeEntite ':' nomEntite (':' Chaine)*;
12 ressource : typeRessource ':' nomRessource (':' Chaine)*;
13
14 userId : Chaine;
15 roleId : Chaine;
16 nomApplication  : Chaine;
17 nomEntite : Chaine;
18 nomRessource : Chaine;
19 valeur : Chaine;
20 typeRegle : (TypeRegle|Chaine);
21 typeAction : (TypeAcces|Chaine);
22 typeApplication : 'A_' Chaine;
23 typeEntite      : 'E_' Chaine;
24 typeRessource   : 'R_' (ClasseRessource|Chaine);
25 conditionObjet  : (ConditionObjet|Chaine);
26
27 TypeRegle : 'allow';
28 TypeAcces :('read'|'write'|'create'|'delete'|'execute');
29 ClasseRessource : ('page'|'file'|'control'|'cookies'|'session'|'cache');
30 ConditionObjet : ('Session'|'Request'|'Cache');
31 Operateur : ('=='|'!='|'>'|'<'|'>='|'<=');
32 Chaine  : ('a'..'z'|'A'..'Z'|'0'..'9'|'*'|'.'|'/'|'?'|'-'|'_'|' '|'$')+ ;
```

Listing 3.1 – Grammaire du langage.

Politique

Dans un système de contrôle d'accès obligatoire, une politique de sécurité représente un ensemble de règles d'autorisations qui vont être appliquées sur un système. Une politique est donc composée d'un ensemble de *règles* séparées par un saut de ligne (Ligne 1).

Règles

La syntaxe d'une règle d'autorisation est la suivante :

allow(sujet,action,objet) : conditions

Une règle de ce type représente l'*autorisation* pour un contexte de sécurité *sujet* d'effectuer une *action* sur un contexte de sécurité *objet*, si les *conditions* exprimées sont vérifiées. Nous considérons un sujet comme une entité active sur le serveur (un utilisateur) tandis qu'un objet sera une entité passive (une ressource). Un contexte de sécurité est un identifiant qui permet de désigner de manière sûre et unique un sujet et/ou un objet sur le serveur. Nous détaillons ci-après ces deux notions de contextes de sécurité pour les sujets et les objets.

Sujet

Un contexte sujet est représenté par une chaîne de caractères contenant un ensemble d'éléments permettant de représenter de manière certaine l'entité considérée. Pour représenter de manière unique et certaine un contexte sujet, plusieurs informations sont nécessaires. Dans notre modèle (listing 3.1 , ligne 3), le contexte de sécurité du sujet est une chaîne de caractères qui doit à minima contenir l'identifiant de l'utilisateur et l'identifiant de rôle applicatif. Ainsi, le contexte de sécurité sujet C_s sera représenté à minima par la chaîne de caractères suivante :

$$C_s = utilisateur : rôle$$

De plus, notre modèle est extensible, et le contexte de sécurité sujet pourra contenir d'autres informations dans le cas où les identifiants d'utilisateur et de rôle ne sont pas suffisants pour représenter de manière sûre et unique un sujet. Par exemple, il est possible d'ajouter un identifiant de localisation, dans le but de restreindre l'accès à certaines ressources selon le pays où l'utilisateur se trouve.

$$C_s = utilisateur : rôle : France$$

Cependant, les identifiants d'utilisateurs et de rôles ne sont pas toujours nécessaires pour représenter un contexte sujet. Cela correspond typiquement à un accès anonyme d'un utilisateur non authentifié. Dans ce cas, un point d'interrogation représente un utilisateur ou un rôle inconnu sur le serveur Web. A contrario, un astérisque désigne n'importe quel utilisateur ou rôle connu sur le serveur Web. Ces notations dédiées permettent de factoriser des règles d'autorisations et ainsi réduire le nombre de règles nécessaires à la définition de la politique de sécurité d'un serveur d'applications Web.

Action

L'action requise par un sujet sur un objet serveur dépend de la requête HTTP entrante mais également du type d'objet accédé.

Tout d'abord, il existe au niveau HTTP plusieurs types de *commandes* auxquelles nous pouvons associer des actions différentes sur le serveur. Les commandes les plus utilisées sont les commandes *GET* et *POST* qui permettent respectivement de demander l'accès à un objet et de transmettre des données qui seront traitées par un objet. Cependant, d'un point de vue HTTP, la commande GET permet d'accéder à un objet, qu'il soit passif (une image par exemple) ou actif (une page dynamique). Mais l'action effectuée par le sujet pour accéder à de tels objets peut être différente.

Pour une ressource passive, c'est-à-dire ne nécessitant pas de traitement par le serveur, il s'agit d'une simple lecture, notée *read*. Une page dynamique implique une *exécution* de code au niveau du serveur. Nous parlerons dans ce cas d'action d'*exécution de code*, notée *execute*. Les données envoyées via une commande *POST* sont traitées au niveau du serveur par un objet actif. Nous pouvons donc associer cette commande à une *exécution de code*, donc une action *execute*.

Il existe également d'autres commandes HTTP telles que *PUT* et *DELETE*. La commande *PUT* permet de remplacer ou d'ajouter une ressource sur le serveur. Nous parlerons dans ces cas là d'actions d'écriture (*write*) ou de création (*create*). La commande *DELETE*, permettant de supprimer un objet sur le serveur, est associée à une action de suppression que nous noterons *delete*.

Objet

Le contexte de sécurité objet, c'est-à-dire une ressource du serveur Web, est représenté par une chaîne de caractères contenant un ensemble d'éléments permettant de représenter de manière unique et sûre l'entité considérée. Dans notre modèle (cf. listing 3.1, ligne 5), le contexte de sécurité de l'objet est une liste de paramètres extensible. Ces paramètres représentent soit une *application*, une *entité d'application* ou une *ressource*. Ainsi, la représentation d'un objet dans notre grammaire reprend bien la notion de relation proposée dans le modèle d'application Web (cf. partie 3.1)

Ressources

Nous avons vu dans la partie 3.1 qu'il existe plusieurs types de ressources sur un serveur Web. Nous proposons (cf. listing 3.1, ligne 19) six types de ressources prédéfinis que nous allons détailler. Néanmoins, l'extensibilité de notre modèle permet d'ajouter d'autres types de ressources (voir listing 3.1, ligne 16) sous la forme de chaînes de caractères. Les types de ressources que nous définissons dans notre modèle sont présentés dans les paragraphes suivants. Pour chaque type de ressource présenté, nous définissons quelles informations nous permettent d'identifier de manière sûre et unique une ressource de ce type.

Page Le type *Page* représente les pages Web dynamiques présentes sur un serveur Web, c'est-à-dire les ressources dont le traitement par le serveur nécessite l'exécution de code. Nous pouvons par exemple citer les fichiers de type ASPX (Serveur IIS Microsoft) ou PHP. Ce type d'objet représente une ressource physique du serveur Web, qui peut donc être identifié par son chemin d'accès physique.

$$nomRessource = Identifiant(Ressource) = Chemin(Page)$$

File A l'inverse du type *Page*, le type *File* représente les pages et les fichiers statiques du serveur Web, qui ne nécessitent aucun traitement particulier par le serveur (une image au format Jpg par exemple). Comme pour les objets de type *Page*, il s'agit de ressources physiques, identifiées par leur chemin d'accès physique sur le serveur.

$$nomRessource = Identifiant(Ressource) = Chemin(Fichier)$$

Control Une page Web, qu'elle soit dynamique ou non, est constituée d'un ensemble de ressources *internes*, communément appelées *composants logiciels*. Dans notre modèle, le type *Control* représente ce type d'objets. Chacun des composants logiciels d'une page Web est identifié de manière stricte grâce à un identifiant unique.

Ainsi, nous pouvons construire son identifiant de ressource par la concaténation de l'identifiant de la page Id_Page auquel le composant logiciel appartient et de son identifiant unique Id_Cmp au sein de cette page.

$$nomRessource = Identifiant(Ressource) = Id\_Page : Id\_Cmp$$

Cookies La RFC 6265 ajoute la notion de *cookies* au protocole HTTP et définit un cookie comme une suite d'informations envoyées par un serveur Web à un client. Lors d'une requête à un serveur Web, le client renvoi le/les cookies précédemment reçus sans les modifier. Les cookies permettent donc à un serveur HTTP de stocker des données, dont

l'accès peut être limité uniquement à certaines type d'utilisateurs, sur le poste client. Les cookies sont représentés sous la forme d'un quintuplet :

$$Cookie = (identifiant, valeur, date\ d'expiration, chemin, domaine)$$

Le navigateur client ne retourne au serveur que les cookies non expirés et dont le chemin et le domaine correspondent à ceux du serveur Web considéré.

Nous pouvons utiliser directement l'identifiant du cookie comme identifiant de ressource dans notre modèle :

$$nomRessource = Identifiant(Ressource) = Identifiant(Cookie)$$

Session Le protocole HTTP étant un protocole non connecté et sans état, les serveurs Web doivent implémenter leur propre gestion d'état. Cette gestion d'état peut être effectuée directement par le serveur. Elle peut également être externalisée, dans un service de gestion d'état dédié ou une base de donnée.

Lors de la première connexion d'un client au serveur Web, le serveur *ouvre* une session via le gestionnaire d'état serveur et retourne un identifiant unique de session au client. L'identifiant de session est stocké par le client, et est renvoyé au serveur à chaque requête suivante, généralement sous forme de paramètre de requête ou de cookie. Avant le traitement effectif d'une page Web ou l'accès à une ressource statique, le serveur Web récupère auprès du gestionnaire d'état, grâce à l'identifiant de session fourni par le client, une copie de l'ensemble des valeurs de sessions associées à l'utilisateur.

Nous pouvons modéliser une session comme un ensemble de couples (identifiant,valeur) stockés par un gestionnaire de session serveur. Cet ensemble s'associe à l'identifiant de session permettant d'accéder à ces données.

$$Session = Id\_Session : \{(id\_donnnée, valeur)\}$$

Ainsi, la concaténation de l'identifiant global de session et de l'identifiant de valeur permet d'obtenir un identifiant de ressource représentant de manière unique chacune des données de la session.

$$nomRessource = Identifiant(Ressource) = Id\_Session : Id\_donnée$$

Cache Les systèmes de *cache* sont utilisés par les serveurs d'applications Web pour stocker des données en mémoire dans l'optique d'y accéder rapidement et améliorer les performances. Les données sont stockées dans le cache du serveur lors de leur première utilisation. Lors des futurs accès, ces données sont extraites directement du cache plutôt que de la source d'origine. Des données sensibles peuvent donc être stockées dans le cache du serveur Web. A la différence des données de sessions qui sont associées à un client unique, les données en cache sont partagées par l'ensemble des utilisateurs du serveur.

Nous pouvons modéliser simplement le cache d'un serveur Web comme un ensemble de couples (identifiant,valeur). L'identifiant de la donnée stockée dans le cache pourra donc directement représenter l'identifiant de ressource de notre modèle.

$$nomRessource = Identifiant(Ressource) = Id\_donnée(Cache)$$

Conditions

Une autorisation d'accès à une ressource du serveur peut être conditionnelle, c'est-à-dire qu'elle ne dépendra pas uniquement de l'objet accédé et du sujet, mais également du contexte d'exécution de la demande. Les conditions permettent donc de représenter des contraintes. Par exemple, la lecture de la valeur d'un compte bancaire client peut être autorisée uniquement si celui-ci est positif, en utilisant une condition sur la valeur du compte bancaire. Nous proposons (cf. listing 3.1, ligne 20) trois types de données prédéfinis à partir desquels il est possible de créer des conditions. Cependant, l'extensibilité de notre modèle permet d'ajouter d'autres types de données (voir listing 3.1, ligne 7) sous la forme de chaînes de caractères.

Nous définissons trois types de données par défaut qu'il est possible d'utiliser pour écrire des conditions : les objets *Request*, *Session* et *Cache*. Les objets *Session* et *Cache* correspondent à ceux présentés dans 3.3.1. Quant aux objets *Request*, il s'agit des données envoyées par le client au serveur lors d'une requête HTTP.

La RFC 2616 définit le protocole HTTP (pour Hypertext Transfer Protocol) et décrit notamment la structure des messages circulant entre un client et un serveur. Deux méthodes d'envoi de données d'un client à un serveur existent : Les paramètres d'URL (ou paramètres GET) permettent à un client d'envoyer à un serveur Web des données en concaténant à l'URL d'une requête HTTP un certain nombre de couples $(identifiant, valeur)$. Les paramètres de formulaires (ou paramètres POST) permettent également à un client d'envoyer des données à un serveur Web. Les données sont dans ce cas envoyées à l'intérieur du corps du message de la requête HTTP. Comme pour des paramètres d'URL, un certain nombre de couples $(identifiant, valeur)$ sont envoyés par le client.

Ainsi, nous pouvons utiliser ces objets pour ajouter des conditions à vérifier lors de l'autorisation d'un accès. Par exemple, les règles d'autorisations suivantes autorisent l'accès à la validation des achats pour le rôle *responsable*, si le montant de l'achat (ici stocké en session) est inférieur à 1000, et au rôle *direction* quelque soit le montant.

$$allow(* : responsable, execute, /Achats/Validation.aspx) : Session(\text{"}montant\text{"}) \leq 1000$$
$$allow(* : direction, execute, /Achats/Validation.aspx)$$

3.3.2 Calcul des règles potentielles d'accès

Dans un mécanisme de contrôle d'accès obligatoire appliqué au niveau système (SE-Linux par exemple), il existe à un instant t un contexte de sécurité unique et connu permettant de représenter un sujet. En effet, il s'agit d'un mécanisme intrusif dans lequel l'architecture du système qu'il contrôle est modifiée. Par exemple, le système de fichiers utilisé contient pour chaque ressource une donnée supplémentaire représentant le contexte de sécurité objet de la ressource. A contrario, le mécanisme de contrôle d'accès obligatoire pour un serveur Web que nous proposons n'est pas intrusif, et indépendant du type de serveur qu'il protège. Nous ne modifions pas l'architecture du système que nous allons contrôler et cela impose certaines contraintes supplémentaires. Tout d'abord, de part la conception des applications Web, une entité active d'une application Web peut être représentée à un instant t par plusieurs contextes de sécurité.

Ainsi, le moniteur de référence doit être capable de calculer dynamiquement l'ensemble des contextes de sécurité sujet possible associés à une demande.

De plus, notre mécanisme de contrôle d'accès n'utilise pas de système de labellisation statique des sujets et des objets, mais une labellisation dynamique. Le moniteur de référence doit donc calculer dynamiquement pour chaque requête le contexte de sécurité sujet, le contexte de sécurité objet et le type d'action requise.

Calcul des contextes de sécurité sujets

Nous avons vu dans la partie 3.3.1 que le contexte de sécurité d'un sujet est composé d'un ensemble non borné de paramètres, et qu'il doit a minima contenir un identifiant d'*utilisateur* et de *rôle*. Les valeurs des identifiants d'utilisateurs et de rôles peuvent être variables pour chaque requête et un utilisateur peut posséder plusieurs rôles différents au sein d'une application Web. Par exemple, l'utilisateur *Bob* peut posséder deux rôles (*Administrateur* et *Rédacteur*) au sein du serveur. Ainsi, deux contextes de sécurité sujets pour l'utilisateur *Bob* sont possible :

$$Bob : Administrateur$$
$$Bob : Rédacteur$$

Le moniteur de référence doit donc extraire ces valeurs pour chaque requête entrante sur le serveur. Cependant, même si ces informations sont disponibles dans l'état d'une requête, chaque application Web implémente sa propre méthode de représentation de ces informations. Pour la raison d'indépendance du moniteur de référence vis-à-vis de l'application qu'il contrôle évoquée dans 3.2.1, le moniteur de référence ne peut pas directement implémenter en son sein les méthodes utilisées par l'application afin de calculer les valeurs d'identifiants d'utilisateurs et de rôles pour une requête entrante. A chaque requête, le moniteur de référence interroge l'adaptateur applicatif en lui fournissant l'état de la requête entrante. Grâce à ces paramètres, le moniteur de référence peut obtenir dynamiquement le contexte de sécurité sujet associé à la requête entrante auprès de l'adaptateur applicatif.

Ainsi, nous obtenons une liste de contextes sujets CS_R associés à la requête entrante R. Pour décider de l'autorisation ou du refus de l'accès à l'objet, il est également nécessaire de calculer les contextes de sécurité objets associés à cet objet.

Calcul des contextes de sécurité objets

Dans notre langage, il est possible d'utiliser des expressions régulières pour représenter un ensemble d'objets ou de ressources pour lesquels les droits d'accès sont identiques. Cela permet de factoriser l'écriture des règles dans la politique de sécurité et de réduire la taille de la politique de sécurité. Cela facilite également les tâches d'administration de la politique de sécurité. Cependant, les types d'expressions régulières utilisables sont limités et connus. Le nombre d'expressions régulières pouvant représenter une ressource est donc borné et calculable. Ainsi, pour chaque objet ou ressource d'une application Web nous obtenons un ensemble borné, notée CO_R, des contextes de sécurité représentant cet objet.

Par exemple, pour la page /GED/Stats/moyenne.aspx les identifiants de ressources représentant cette page sont entre autres :

$$page : /GED/Stats/moyenne.aspx$$
$$page : /GED/Stats/*.aspx$$
$$page : /GED/Stats/*$$

La combinaison des ensembles CS_R (les contextes de sécurité sujets) et CO_R (les contextes de sécurité objets) associés a la requête entrante R nous permet de calculer l'ensemble des règles pouvant potentiellement autoriser l'*Action* associée à la requête entrante, noté Rp_R, où l'action est déduite du type de l'objet.

Définition 9 (Règles potentielles d'accès)**.**

$$Rp_R = \{allow(C, Action, O) \ / \ C \in CS_R \ , \ O \in CO_R \ , \ Action = typeAction(O)\}$$

3.3.3 Décision d'autorisation

Étant donnée une politique de sécurité POL et un ensemble Rp_R de règles potentielles d'accès pour une requête entrante R, l'accès à l'objet pour la requête considérée est autorisé si une des règles potentielles d'accès pour la requête entrante est présente dans la politique de sécurité. Il doit donc exister dans la politique au moins une règle autorisant cet accès. Si une telle règle existe, il est également nécessaire que les conditions exprimées pour cette règle soient vérifiées pour l'état de la requête entrante R :

Définition 10 (Décision d'autorisation)**.**

Soit R une requête entrante
Rp_R l'ensemble des règles potentielles d'accès de R
La requête R est autorisé si et seulement si :
$\exists\ A \in Rp_R,\ \exists\ P \in POL\ /\ A = P$ *et* $conditions(P, R) = vrai$

3.4 Calcul des politiques de sécurité

Le calculateur de politique (voir figure 3.4 et partie 3.2.2) permet de résoudre le problème de création d'une politique de sécurité obligatoire satisfaisant le principe de moindre privilège. Ainsi, la politique autorise les utilisateurs à accéder uniquement aux objets qui leur sont nécessaires dans l'accomplissement de leur tâches, et refuse tous les autres accès.

La création d'une politique de sécurité sûre et satisfaisant les objectifs de sécurité est un problème majeur et un frein à la mise en place d'un système de contrôle d'accès obligatoire. En pratique, obtenir et administrer une politique de sécurité obligatoire complète est difficile car il n'existe pas d'outils permettant de calculer complètement ces politiques. Celles-ci doivent donc être créées manuellement par l'administrateur. Cela s'avère très long et décourage les administrateurs à utiliser de tels systèmes de contrôle d'accès.

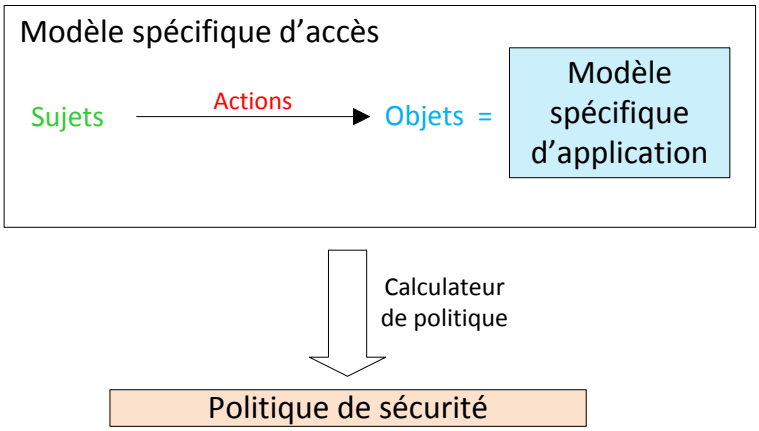

FIGURE 3.6 – Modèle spécifique d'accès aux applications Web.

Par simplification, nous utiliserons le terme **modèle d'application Web** pour évoquer un modèle spécifique d'accès aux applications Web.

Le but du calculateur de politique est donc d'automatiser l'obtention d'une politique de sécurité obligatoire pour un serveur d'applications Web. Celui-ci s'appuie sur un **modèle d'application Web** pour calculer un modèle spécifique d'accès qui représente les accès autorisés au sein d'une application.

À partir de ce modèle spécifique d'accès, nous pouvons calculer une politique de sécurité dans notre langage de protection grâce à un mécanisme de conversion. Nous proposons tout d'abord un modèle spécifique d'application reposant sur quatre niveaux de structuration des objets (Applications, Services, Fonctionnalités et Ressources). Nous en déduisons un modèle spécifique d'accès. De plus, nous proposons un modèle spécifique de workflows associé au modèle spécifique d'accès aux workflows. Nous verrons que ce modèle spécifique d'accès pour les workflows est facilement transposable vers notre modèle spécifique d'accès aux applications Web et permet donc d'en déduire une politique de sécurité pour les systèmes de workflows.

3.4.1 Modèle spécifique d'accès pour les applications Web

Modèle spécifique d'application

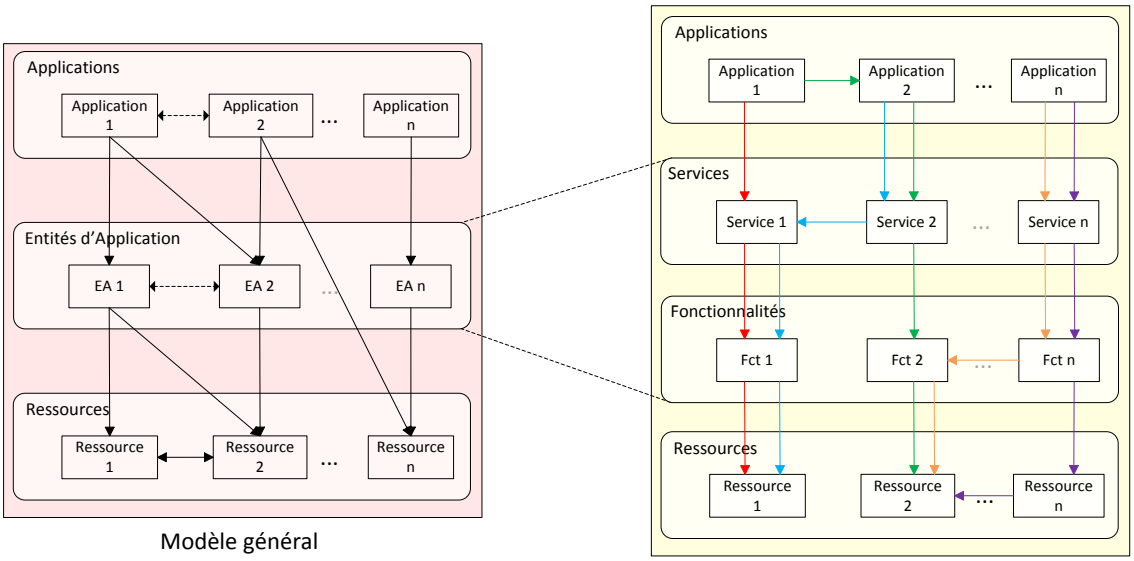

FIGURE 3.7 – Modèle spécifique d'applications Web.

Concernant la modélisation des objets, nous proposons un modèle spécifique d'application structuré en quatre niveaux (figure 3.7). Les niveaux *Applications* et *Ressources* correspondent aux notions d'*Applications* et *Ressources* que nous avions introduits dans notre modélisation des applications Web (cf. partie 3.1). Les *entités d'applications* sont quant à elles séparées en deux niveaux : le niveau *Services* et le niveau *Fonctionnalités*.

Ce modèle spécifique ne fait intervenir que deux types de relations que nous nommerons relations *directes* et *indirectes*.

Les relations *directes* représentent des relations pouvant exister au sein d'une application Web et imposent une hiérarchisation des entités, le niveau le plus haut dans la hiérarchisation étant le niveau *Applications*.

Le second niveau de notre modèle est le niveau *Services*. Les *Fonctionnalités* représentent le troisième niveau de structuration de notre modèle. Enfin, le niveau le plus bas de notre modèle est le niveau *Ressources*.

Cette hiérarchisation entre les quatre niveaux de notre modèle impose qu'une relation directe ne fasse intervenir au maximum qu'un seul paramètre pour chaque niveau de structuration. Cela signifie qu'un paramètre d'un niveau n ne peut intervenir dans une relation uniquement si un paramètre du niveau *n-1* intervient dans la relation. Une relation représente donc l'organisation logicielle de l'application. Cependant, la sémantique reste à la charge des applications et il n'est pas necessaire de la connaître pour calculer les politiques de sécurité. Cela correspond à la structure généralement utilisée au sein des applications Web, notamment dans des modèles tels que SOA. Ainsi, nous obtenons par récurrence pour ce modèle quatre *prototypes* différents pour une relation *directe* :

Définition 11 (Relations directes)**.**

$$directe(A_r)$$
$$directe(A_r, S_r)$$
$$directe(A_r, S_r, F_r)$$
$$directe(A_r, S_r, F_r, R_r)\ /$$
$$A_r \in Applications,\ S_r \in Services,\ F_r \in Fonctionnalités,\ R_r \in Ressources$$

Quant à elles, les relations *indirectes* peuvent faire intervenir jusqu'à deux paramètres pour un des quatre niveaux de structuration. Il existe donc quatre *prototypes* différents pour la relation *indirecte* :

Définition 12 (Relations indirectes)**.**

$$indirecte(\boldsymbol{A_{r1}}, \boldsymbol{A_{r2}}, S_r, Fr, R_r)$$
$$indirecte(A_r, \boldsymbol{S_{r1}}, \boldsymbol{S_{r2}}, F_r, R_r)$$
$$indirecte(A_r, S_r, \boldsymbol{F_{r1}}, \boldsymbol{F_{r2}}, R_r)$$
$$indirecte(A_r, S_r, F_r, \boldsymbol{R_{r1}}, \boldsymbol{R_{r2}})\ /$$
$$A_r \in Applications,\ S_r \in Services,\ F_r \in Fonctionnalités,\ R_r \in Ressources$$

Modèle spécifique d'accès

La figure 3.7 présente notre modèle spécifique d'accès pour les applications Web permettant le calcul d'une politique de sécurité. Ce modèle reprend les notions de sujet, actions, objets et conditions présentées dans notre grammaire (voir 3.3.1). Il permet également de représenter les relations entre les entités d'applications.

Dans notre modèle, un sujet est représenté par un utilisateur et un rôle associé à cet utilisateur. Ces deux attributs correspondent aux attributs minimum requis par notre langage de règles (voir partie 3.3.1). Les informations nécessaires pour la représentation d'un sujet sont donc les identifiants d'utilisateur (la table *Utilisateur*), les rôles applicatifs (la table *Rôles*) et les associations de rôles aux utilisateurs (la table *Laison_Roles_Ut*).

La hiérarchisation entre les applications et les services est représentée par la table *Liaison_Appli_Services*. Les services sont hiérarchiquement liés aux fonctionnalités par la table *Liaison_Service_Fct*. Les fonctionnalités et les ressources sont liées entre elles via la table *Liaison_Fct_Ressources*.

Les actions et les conditions sont regroupées au sein de la table *Actions*. Chaque action possède un attribut de type qui représente les types d'actions (*read, write, execute, etc.*) tels que nous les avons défini dans la partie 3.3.1.

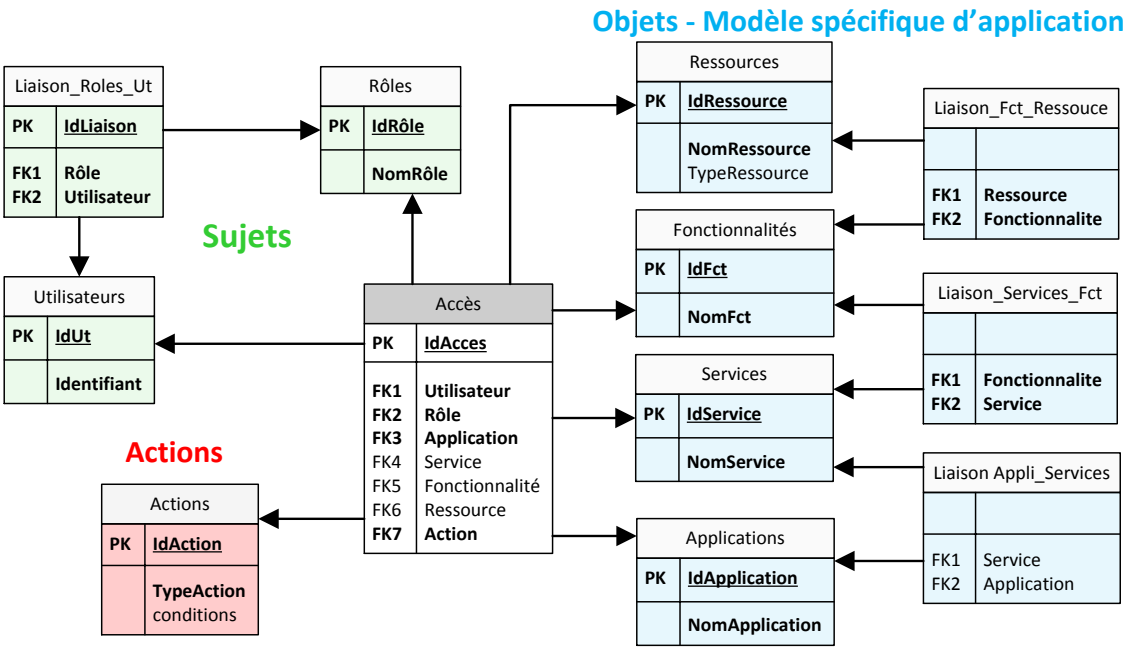

FIGURE 3.8 – Modèle spécifique d'accès pour les applications Web

3.4.2 Modèle spécifique d'accès pour les workflows

Modèle spécifique de workflows

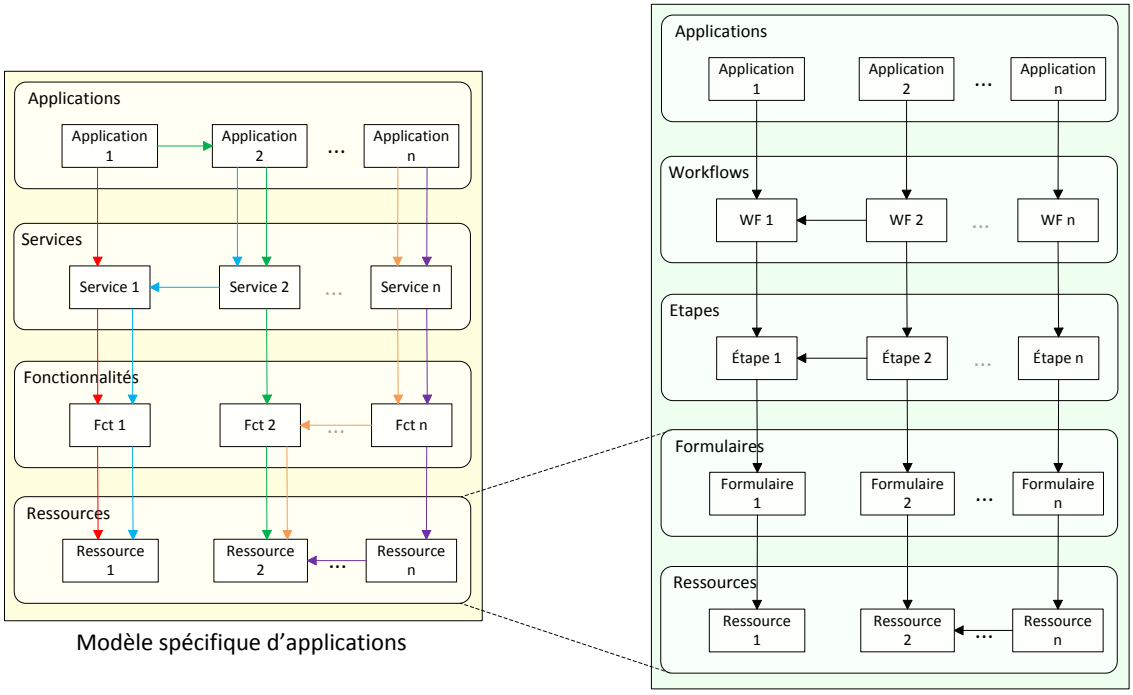

FIGURE 3.9 – Modèle spécifique de workflows.

Pour la modélisation des objets, nous proposons un *modèle spécifique de Workflows* structuré en cinq niveaux (figures 3.9 et 3.10). Ces cinq niveaux (*Applications, Workflows, Étapes, Formulaires et Ressources*) modélisent les notions principales existantes dans tout système de workflows.

Généralement, une application de workflow (représentée dans notre modèle par un identifiant et un nom d'application) est composée d'un ensemble de workflows. Chaque workflow est également représenté par un identifiant et un nom. Un *workflow* est composé d'un ensemble d'*étapes* liées entre elles, qui représentent les différentes opérations à accomplir pour réaliser un processus métier.

Dans notre modèle spécifique d'accès pour les workflows de la figure 3.10, les liaisons entre les étapes (table *Laison_Entre_Etapes*) modélisent l'enchainement des étapes au sein du workflow, c'est-à-dire le flux d'information au sein du workflow. L'ordre d'exécution des étapes au sein d'un workflow est imposé par le modèle de workflow. Ainsi, les liaisons sont ordonnées (attribut *Ordre* de la table *Laison_Entre_Etapes*). Cependant, le flux suivi par des données dans un workflow, et donc le modèle de workflow en lui-même, peut être dépendant des données traitées dans une instance particulière du workflow. Le flux de données peut être conditionné aux valeurs des données traitées (attribut *Conditions* de la table *Laison_Entre_Etapes*). Par exemple, une demande d'achats peut être effectuée automatiquement si son montant est inférieur à 1000€, ou nécessiter une validation hiérarchique si le montant est supérieur à cette valeur.

À chacune des étapes est associée un *Formulaire* (table *Etapes_Formulaire*) qui centralise l'ensemble des *ressources* nécessaires à la réalisation de l'étape du workflow.

modèle spécifique d'accès de workflows

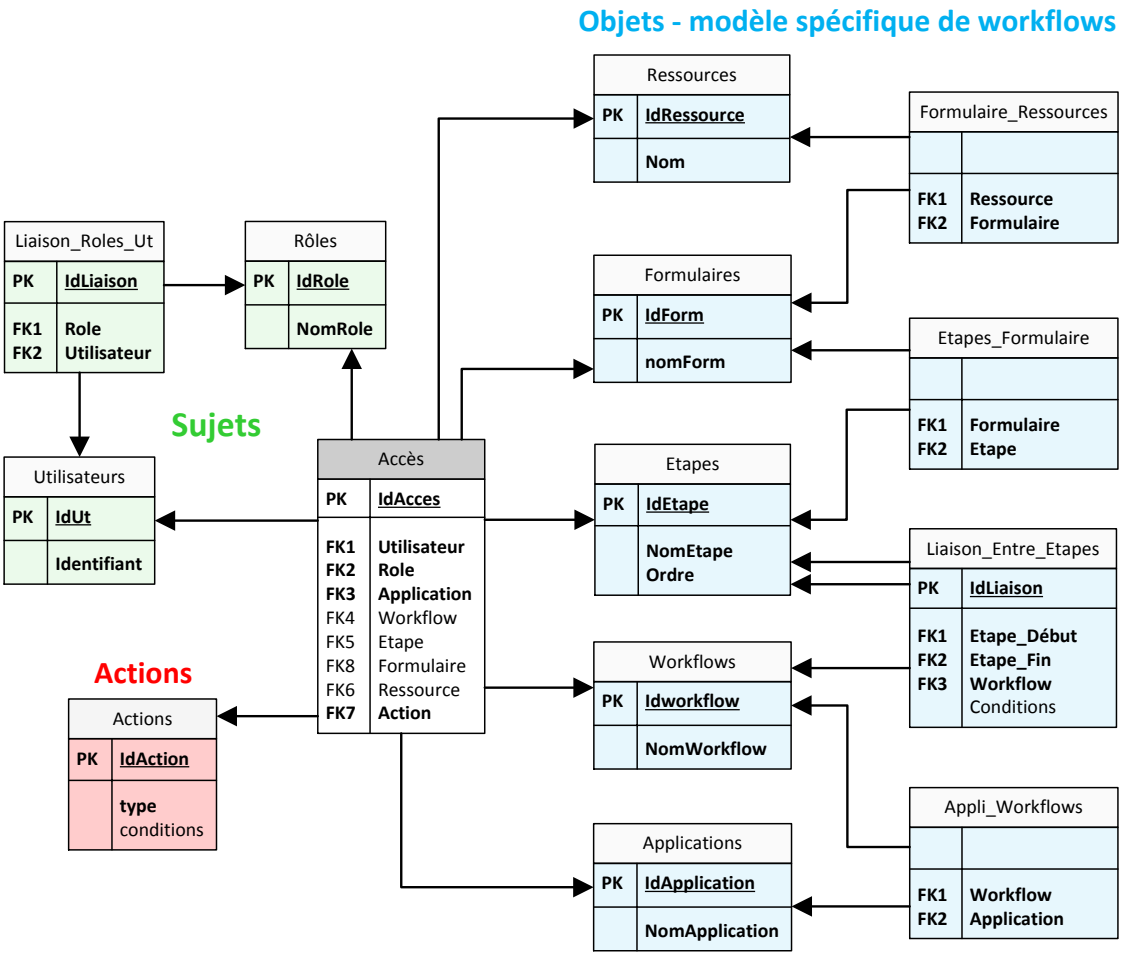

FIGURE 3.10 – Modèle spécifique d'accès pour les workflows

La figure 3.9 présente notre **modèle spécifique d'accès pour les workflows**. La représentation des sujets et des actions est identique à celle proposée dans notre modèle spécifique d'accès pour les applications dans la partie 3.4.1. La transposition des sujets et des actions vers le modèle spécifique d'accès permettant le calcul de la politique de sécurité est donc triviale.

Transposition vers le modèle spécifique d'application

Pour calculer une politique de contrôle d'accès à partir de notre modèle spécifique d'accès pour les workflows, il est nécessaire de transposer celui-ci vers notre modèle spécifique d'accès pour les applications. Les niveaux *Applications* et *Ressources* peuvent être transposés directement d'un modèle à l'autre. Pour les *Workflows*, nous proposons de les transposer en *Services* de notre modèle spécifique d'application, et les *Étapes* correspondent à des *Fonctionnalités* d'un workflow. Enfin, nous proposons de regrouper les *Formulaires* avec les ressources. Ce regroupement impose cependant l'utilisation de relations indirectes faisant intervenir deux paramètres du niveau *Ressources*, c'est-à-dire des relations du type :

$$Laison\_Formulaire\_Ressource = indirecte(A_r, S_r, F_r, \boldsymbol{R_{r1}}, \boldsymbol{R_{r2}})$$
$$R_{r1} \in Formulaires, R_{r2} \in Ressources$$

De plus, un workflow peut faire référence à un autre workflow ou à un autre service de l'application. Ce fonctionnement peut être modélisé par une relation indirecte dans laquelle interviennent deux paramètres du niveau *Services* :

$$Laison\_Workflows\_Services = indirecte(A_r, \boldsymbol{S_{r1}}, \boldsymbol{S_{r2}}, F_r, R_{r1}, R_{r2})$$
$$S_{r1} \in Workflow, S_{r2} \in Workflows \cup Services$$

Exemple de workflow - Demande d'achats

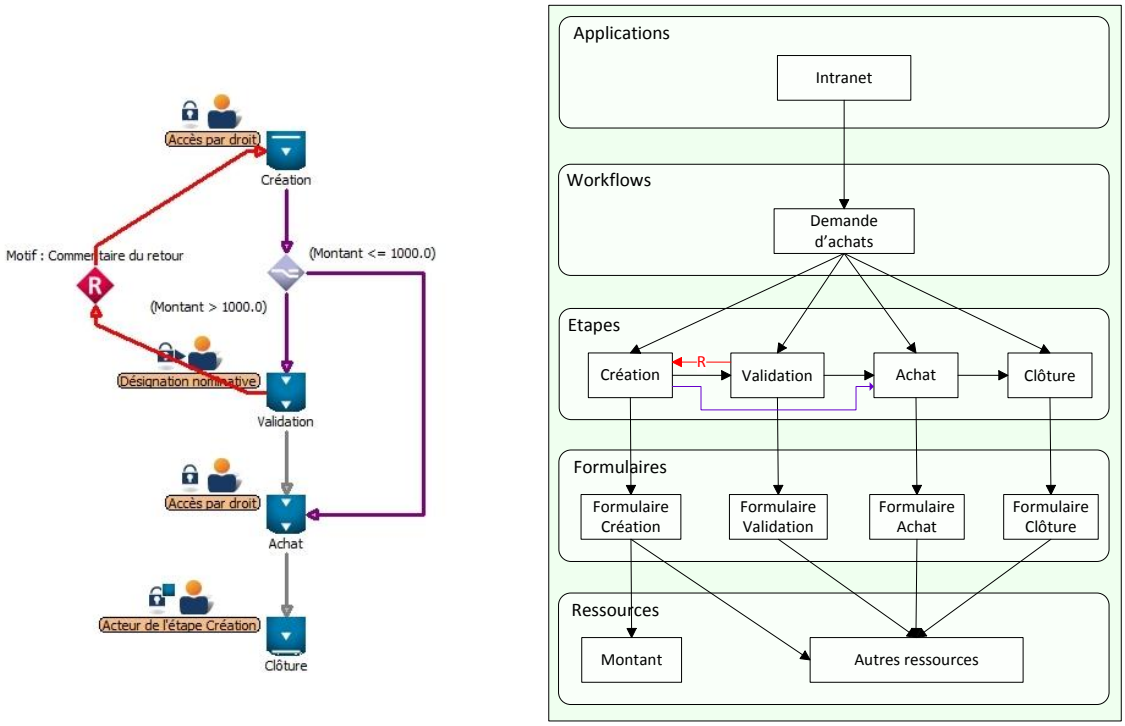

FIGURE 3.11 – Exemple : Workflow de demande d'achats et modèle spécifique correspondant.

La figure 3.11 illustre notre modèle spécifique de workflow sur un workflow de demande d'achats. Ce workflow est constitué de quatre étapes : *Création*, *Validation*, *Achat* et *Clôture*. Le modèle spécifique ne fait apparaître que les entités relatives à ce workflow. Nous y retrouvons les différentes *étapes*, ainsi que les formulaires associés à ces étapes. Ces formulaires utilisent un certain nombre de ressources qui ne sont pas représentées ici. Le modèle spécifique fait également apparaître les relations associées à ce workflow qui permettent notamment de représenter la sémantique d'exécution du workflow.

Par exemple, l'appartenance de la ressource *Montant* au formulaire de création, qui lui même appartient à l'étape de *Création* peut être modélisée par une relation du type :

$$directe(Intranet, Demande\ d'achats, Création, Form\ création, Montant)$$

Le cas particulier du $Refus\_Validation$, qui représente notamment la possibilité lors de la validation de la demande (Si Montant $>$ 1000) de refuser la demande et la renvoyer à l'étape précédente sera modélisé par une relation indirecte :

$$Refus\_Validation = \\ indirecte(Intranet, Demande\ d'achats, \boldsymbol{Validation}, \boldsymbol{Création}, Form\ création, Montant)$$

Pour finir, le court-circuit de l'étape *Validation* (Si Montant $\leq$ 1000) vers l'étape Achat peut également être modélisé par une relation indirecte :

$$Sans\_Validation = \\ indirecte(Intranet, Demanded'achats, \boldsymbol{Création}, \boldsymbol{Achat}, Form\ Achat, *)$$

Ainsi, toute relation présente au niveau du modèle de workflows peut être traduite dans notre modèle spécifique d'applications Web via les deux relations directes et indirectes entre les applications, les services, les fonctionnalités et les ressources. Ces relations représentent l'organisation fonctionnelle de l'application et ne sont pas necéssaires au calcul de la politique de sécurité.

3.5 Méthode de test et de vérification de notre protection obligatoire

Nous proposons ici une méthode permettant de tester et vérifier l'exactitude de notre protection. Grâce au calculateur de politique, nous obtenons une politique de sécurité qui est directement applicable par notre protection obligatoire dynamique sur le serveur d'applications Web. Cependant, il est nécessaire de vérifier que la politique générée correspondent bien au modèle spécifique d'accès à partir duquel elle a été calculée. Cette vérification nous assurera l'exactitude de la protection appliquée. L'approche de vérification permet de valider que la politique calculée et son application sur le serveur d'applications Web par le moniteur de référence correspond bien au modèle spécifique d'application. Pour cela, notre approche permet de vérifier par un jeu de tests et une condition de validation que toutes les autorisations exprimées dans la politique calculée sont bien mises en œuvre. Elle permet également de vérifier qu'aucun accès illégitime vis-à-vis du modèle spécifique d'accès n'est autorisé par le moniteur de référence.

Le principe de notre méthode de vérification est de tester exhaustivement toutes les actions qui peuvent potentiellement être effectuées sur le système par tous les sujets du serveur d'application et pour toutes les relations existantes. Pour chacune de ces actions, nous vérifions si celle-ci est autorisée ou interdite par notre protection obligatoire. En comparant cela avec le modèle spécifique d'accès, nous pouvons confirmer ou infirmer la validité de notre protection relativement à ce modèle.

Expression formelle

Soit N_P le nombre de règles d'autorisations exprimées dans la politique de sécurité P, S le nombre de sujets existants au sein du serveur d'applications et R le nombre de relations du modèle spécifique d'application.

Le nombre total d'actions C réalisables sur le système, c'est-à-dire le nombre de cas à vérifier pour couvrir l'intégralité du serveur d'application est donc le produit du nombre de sujets par le nombre de relations du modèle spécifique d'application.

$$C = S * R$$

Cet ensemble de cas de tests peut être divisé en deux sous-ensembles :
- $Refus$: les accès qui auront été refusés par la méthode de vérification.
- $Autorisations$: les accès qui auront été autorisés.

De plus, l'ensemble $Autorisations$ des accès autorisés peut être divisé en deux sous ensembles :
- A_P : les accès qui auront été légitimement autorisés, c'est-à-dire les accès pour lesquels correspond une règle d'autorisation dans la politique de sécurité P.
- $A_{\overline{P}}$: les accès qui auront été autorisés, mais pour lesquels il n'existe pas de règles d'autorisation dans la politique P.

Soit N le nombre de tests effectués par notre méthode de vérification. Ce nombre de tests est égal à la somme des $Autorisations$ et des $Refus$, c'est-à-dire :

$$N = Autorisations + Refus$$
$$\Longleftrightarrow$$
$$N = A_P + A_{\overline{P}} + Refus$$

Trois conditions permettent de valider l'exactitude de notre protection obligatoire.
- Pour assurer une couverture exhaustive du système, Le nombre N de tests effectués doit être égal à C le nombre total d'actions réalisables sur le système.
- Le nombre A_P d'accès légitimement autorisés doit être égal au nombre de règles d'autorisation N_P de la politique de sécurité.
- Le nombre $A_{\overline{P}}$ d'accès illégitimement autorisés doit être nul.

Ces conditions nous permettent de définir trois taux permettant de mesurer de manière efficace l'exactitude de notre protection obligatoire. Tout d'abord, T_C représente le taux de couverture de la politique de sécurité, c'est-à-dire le rapport du nombre de tests effectués sur le nombre total de tests nécessaires à une couverture complète du système.

$$T_C = \frac{N}{C}$$

Ensuite, T_P constitue le taux d'autorisations légitimes, c'est-à-dire le rapport entre le nombre d'autorisations effectives A_P et le nombre d'autorisations de la politique de sécurité.

$$T_P = \frac{A_P}{N_P}$$

Enfin, le taux d'autorisations illégitimes T_I représente le rapport du nombre d'accès pour lesquels il n'existe pas de règles d'autorisation dans la politique vis-à-vis du nombre de règles d'autorisation de la politique.

$$T_I = \frac{A_{\overline{P}}}{N_P}$$

De manière plus formelle, Les conditions d'exactitude de notre protection peuvent donc se définir ainsi :

Définition 13 (Conditions de sûreté de la protection)**.**

Soit C le nombre total d'actions réalisables sur le système,
N le nombre de tests effectués par la méthode de vérification,
N_P le nombre d'autorisations de la politique,
A_P le nombre d'accès légitimement autorisés,
$A_{\overline{P}}$ le nombre d'accès illégitimement autorisés,

La sûreté de la protection est assurée si et seulement si :

$$N = C; A_P = N_P; A_{\overline{P}} = 0$$
$$\Longleftrightarrow$$
$$T_C = \frac{N}{C} = 1;\ T_P = \frac{A_P}{N_P} = 1;\ T_I = \frac{A_{\overline{P}}}{N_P} = 0$$

Algorithme de test et de vérification

L'algorithme 1 présente notre méthode de test d'exactitude de notre protection.

À partir d'un ensemble de contextes sujets et de l'ensemble des relations du modèle spécifique d'application, notre méthode effectue une tentative d'accès pour chaque relation et pour chaque contexte sujet. Si une tentative d'accès est autorisée, notre méthode vérifie

si celle-ci fait partie ou non de la politique de sécurité. Si celle-ci est autorisée, le compteur A_P est incrémenté. S'il s'agit d'un accès illégitime, la valeur $A_{\overline{P}}$ est incrémentée. Le nombre de cas testés N est également mesuré.

À là fin du processus de test, nous récupérons donc les valeurs N du nombre de cas testés, A_P du nombre d'autorisations légitimes et $A_{\overline{P}}$ d'autorisations illégitimes. Grâce à ces valeurs nous pouvons appliquer la définition précédente et vérifier l'exactitude de notre protection.

Algorithme 1 : Méthode de test et de vérification de la protection.

```
Entrées : Contextes Sujets, Relations, Politique
Résultat : A_P, A_P̄, N
pour chaque Sujet dans Contextes Sujets faire
    pour chaque Relation dans Relations faire
        accès = tentative_accès(Sujet, Relation)
        N += 1
        si ( accès est autorisé) alors
            si ( (Sujet,Relation) est dans la Politique) alors
                A_P += 1
            sinon
                A_P̄ += 1
            fin
        fin
    fin
fin
```

Exemple simple

Pour illustrer cette méthode, nous allons l'appliquer sur un exemple simple. Prenons le cas d'une application composée de trois relations notées $Rel1$, $Rel2$ et $Rel3$ dans laquelle interviennent deux contextes de sécurité sujet $S1$ et $S2$.

La politique de sécurité en place autorise trois accès :

$$\begin{aligned} S1 &\rightarrow Rel1\ (A1, S1, F1, R1) \\ S2 &\rightarrow Rel2\ (A2, S2, F2, R2) \\ S2 &\rightarrow Rel3\ (A3, S3, F3, R3) \end{aligned}$$

Dans cet exemple, le nombre de règles d'autorisation est donc de trois, et le nombre total d'actions réalisables C est de six.

$$N_P = 3,\ C = 6$$

Imaginons que le déroulement de notre méthode de vérification donne les résultats suivant :

$$\begin{aligned} S1 &\rightarrow R1 : Autorisé \\ S1 &\rightarrow R2 : Refusé \\ S1 &\rightarrow R3 : Autorisé \\ S2 &\rightarrow R1 : Refusé \\ S2 &\rightarrow R2 : Autorisé \\ S2 &\rightarrow R3 : Autorisé \end{aligned}$$

Dans ce cas, le nombre de tests effectué est de six :

$$N = 6$$

Trois accès ont été légitimement autorisés : $S1 \rightarrow R1$, $S2 \rightarrow R2$, $S2 \rightarrow R3$:

$$A_P = 3$$

Il existe cependant un accès illégitimement autorisé vis-à-vis de la politique de sécurité : $S1 \rightarrow R3, donc$:

$$A_{\overline{P}} = 1$$

Nous pouvons donc vérifier l'exactitude de la protection en appliquant la définition 13 :

$$T_C = \frac{N}{C} = \frac{6}{6} = 1$$

$$T_P = \frac{A_P}{N_P} = \frac{3}{3} = 1$$

$$T_I = \frac{A_{\overline{P}}}{N_P} = \frac{1}{3} = 0.33$$

Dans ce cas, les conditions d'exactitude ne sont pas vérifiées car le taux d'autorisations illégitimes n'est pas nul. En pratique, nous verrons à travers les expérimentations que la politique de sécurité est bien sûre lorsqu'elle est calculée automatiquement.

3.6 Conclusion

Dans ce chapitre, nous avons proposé un modèle d'application Web supportant différents types d'applications. Ce modèle permet de prendre en compte la structuration des applications et de décrire grâce à la notion de relations les liens entre les entités et les ressources composants les applications. Bien que nous ayons défini un modèle d'application particulier reposant sur quatre niveaux de structuration, ce modèle abstrait permet de représenter tous types d'applications Web comme par exemple le modèle SOA.

Nous avons également défini un modèle de protection obligatoire dynamique s'appuyant sur un langage de protection permettant d'exprimer les règles de sécurité à appliquer sur le serveur d'applications Web. Ce langage est suffisamment fin et extensible pour décrire de manière précise les sujets, les actions et les objets. Il permet de définir une politique de sécurité à appliquer sur le serveur et reprend l'ensemble des notions de notre modèle d'application Web, notamment les notions d'applications, de services, de fonctionnalités et de ressources. Le moniteur de référence, composant principal de notre protection obligatoire, peut calculer dynamiquement à chaque requête HTTP entrante, les contextes de sécurité sujets et objets associés à cette requête. Il en déduit des règles potentielles d'accès qui vont être comparées à la politique de sécurité pour décider d'autoriser ou refuser la demande en cours. Ce modèle de protection obligatoire est totalement indépendant des applications protégées en reposant notamment sur un adaptateur applicatif pour assurer cette indépendance.

Nous avons présenté une méthode de calcul automatisé des politiques de sécurité. Cette méthode permet de simplifier l'administration de la protection obligatoire en permettant d'obtenir de manière automatique les politiques de sécurité à appliquer. Cette méthode

s'appuie sur un modèle spécifique d'application et permet de déduire les politiques de sécurité à appliquer pour protéger efficacement un serveur d'applications web. De plus, un modèle de workflows et d'accès pour les workflows permet de déduire les modèles spécifiques d'applications correspondants, à partir desquels nous pouvons en déduire les politiques de sécurité requises. Un modèle spécifique d'accès représente les accès autorisés au sein d'une application. Nous avons proposé un modèle spécifique d'accès pour les applications et pour les workflows, mais notre méthode peut s'appliquer à d'autres modèles.

Enfin, nous avons défini une méthode permettant de tester et de vérifier la sûreté de la protection appliquée et des politiques de sécurité calculées automatiquement. Cette méthode consiste en un test exhaustif de tous les accès pouvant potentiellement être effectués sur le serveur d'application. Le calcul d'une condition de sûreté permet alors d'évaluer le taux de couverture de la protection.

Chapitre 4

Implémentations

Pour cette thèse, nous avons collaboré avec la société Qual'Net qui édite notamment des logiciels de workflows basés sur les environnements Web de Microsoft. Ces environnements s'appuient sur le serveur Web IIS (Internet Information Services) de Microsoft qui supporte nativement le Framework .Net.

Un des objectifs principaux de cette collaboration était d'obtenir une implémentation fonctionnelle de notre modèle de protection obligatoire dynamique qui soit utilisable par Qual'Net pour améliorer la sécurité des applications qu'elle édite. Nous avons donc fait le choix d'implémenter notre protection obligatoire sur les environnements Web utilisés par Qual'Net. Le but était d'intégrer notre protection obligatoire dans les nouvelles versions des logiciels de Qual'Net. Cependant, une implémentation sur d'autres environnements (tels que Apache/PHP) est envisageable mais n'a pas été étudiée.

D'un point de vue technique, la collaboration avec QualNet nous a imposé certaines contraintes sur les technologies utilisées : le serveur Web IIS devait être au minimum en version 7.0 et le Framework .Net en version 4.0.

Dans ce chapitre, nous présenterons donc une implémentation de notre protection obligatoire pour un environnement d'application Web Microsoft. Nous verrons que cette implémentation reprend la majorité des notions exprimées dans le chapitre 3, tout en s'adaptant aux contraintes techniques intrinsèques aux environnements utilisés.

De plus, notre implémentation reste totalement indépendante des applications protégées et donc de l'architecture logicielle des applications Qual'Net et peut donc s'appliquer à n'importe quelle application utilisant une architecture IIS/.Net dans les versions que nous supportons.

Nous détaillerons également l'implémentation de notre système de calcul des politiques obligatoires. Ce système de calcul s'appuie sur un modèle spécifique de workflow. Nous présenterons tout d'abord le modèle de workflows de QualNet et montrerons que celui-ci se transpose bien dans le modèle spécifique de workflows que nous proposons.

Enfin, nous appliquerons notre méthode de vérification de la protection dynamique obligatoire sur un environnement de workflows QualNet pour vérifier l'exactitude de notre protection et des politiques de sécurité calculées à partir du modèle de workflows de QualNet.

4.1 Rappels

Nous avons proposé dans le chapitre 3 une approche globale de protection des serveurs d'applications Web (figure 4.1). Notre approche obligatoire dynamique permet de contrôler précisément les accès des *sujets* sur les *objets* des serveurs d'applications tout en s'adaptant au caractère extensible du Web.

Elle utilise notamment un *adaptateur applicatif* pour calculer dynamiquement, à chaque requête HTTP, le contexte de sécurité du sujet. L'adaptateur applicatif permet entre autre de garantir l'interopérabilité de notre protection obligatoire avec n'importe quelle application, en assurant également l'indépendance de la protection vis-à-vis des modèles d'applications.

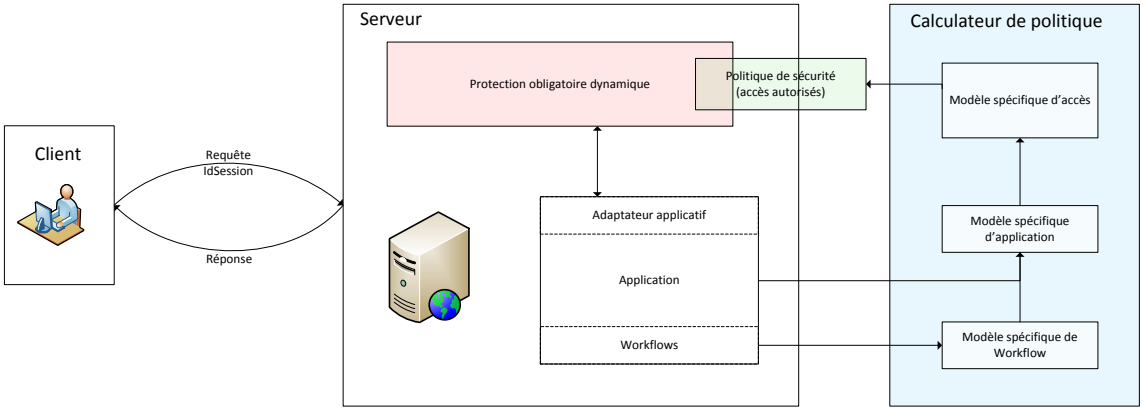

FIGURE 4.1 – Approche globale de protection.

Nous avons défini dans la partie 3.3.1 un langage de protection dédié qui permet de décrire les autorisations d'accès, c'est-à-dire la politique de sécurité à appliquer sur le serveur d'applications Web.

Nous avons également proposé une méthode de calcul automatisé des politiques de sécurité. L'approche repose sur un modèle spécifique de workflows qui se projette sur un modèle d'applications Web. Ce modèle spécifique d'applications (ou de workflows) permet de représenter les relations entre les différentes entités d'une application. Grâce à ce modèle spécifique d'application, nous pouvons en déduire un modèle spécifique d'accès. Un modèle spécifique d'accès permet de représenter les accès nécessaires à une application (ou à des workflows) et le calcul des politiques de sécurité correspondantes.

Enfin, nous avons présenté une méthode de test et de vérification permettant de valider la sûreté de notre protection et des politiques de sécurité calculées. Cette méthode s'appuie sur un test exhaustif de tous les accès possibles au sein d'une application, afin de vérifier les politiques de sécurité appliquées.

4.2 Implémentation de la protection obligatoire dynamique

4.2.1 Intégration au serveur Web IIS

Notre protection obligatoire est implémentée au sein des serveurs Web IIS sous la forme d'un module HTTP (ou HTTPModule). Un module HTTP est un assemblage .Net qui implémente l'interface *IHttpModule* et gère des événements ASP.NET. Par exemple, le module SessionStateModule est fourni par ASP.NET pour doter une application de services d'état de session.

L'architecture que nous avons proposée dans le chapitre 3 n'impose pas que la protection obligatoire soit globale à tout le serveur. Nous proposons ici une protection obligatoire qui s'applique au niveau de chaque application hébergée sur le serveur.

Les modules HTTP sont exécutés à chaque demande entrante sur le serveur Web en réponse notamment aux événements BeginRequest et EndRequest. Les modules HTTP s'exécutent donc avant et après le traitement d'une demande. Une application peut utiliser plusieurs modules HTTP. Les modules HTTP sont exécutés suivant l'ordre dans lequel ils sont inscrits dans le fichier de configuration de l'application (cf. figure 4.2).

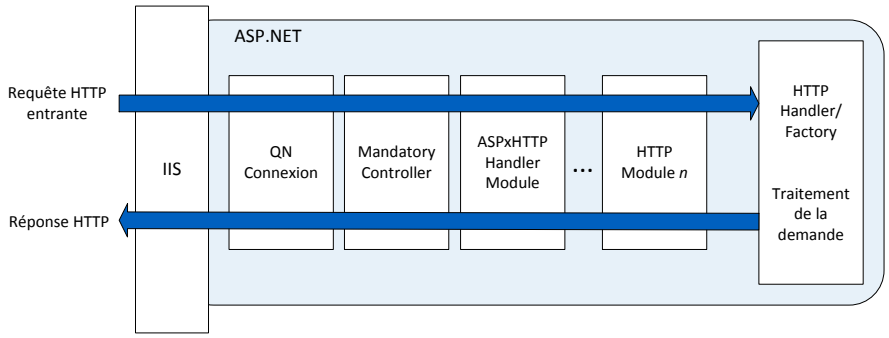

FIGURE 4.2 – HTTPModules.

Pour une application donnée, les modules HTTP sont inscrits au sein du fichier de configuration *Web.config* de cette application. Le *Web.config* est un fichier au format XML contenant toute la configuration de l'application (cf. listing 4.1).

```
1  <?xml version="1.0"?>
2  <configuration>
3    ...
4   <system.webServer>
5    <modules>
6      <add name="QN.HttpModuleConnexion"
7        type="Qualnet.Connection.HttpModuleConnexion" />
8
9      <add name="MandatoryController"
10       type="MandatoryController.MandatoryController" />
11
12     <add name="ASPxHttpHandlerModule"
13       type="DevExpress.Web.ASPxClasses.ASPxHttpHandlerModule,
14       DevExpress.Web.v12.2, Version=12.2.7.0, Culture=neutral,
15       PublicKeyToken=b88d1754d700e49a" />
16   </modules>
17  </system.webServer>
18 </configuration>
```

Listing 4.1 – Exemple d'inscription de modules HTTP dans un Web.config.

D'un point de vue .Net, un module HTTP est donc une classe qui implémente une interface particulière (*IHttpModule*). Cette classe contient un certain nombre de fonctions, appelées gestionnaires, qui sont associées à certains événements fournis par le framework.

Le premier événement déclenché par un module HTTP est l'événement *Init*. Celui-ci n'est déclenché qu'une seule fois, lors du chargement en mémoire du module par le serveur IIS, contrairement aux autres événements (les méthodes *Appli_BeginRequest* et *Appli_PostAcquireRequestState* dans l'exemple du listing 4.2). Lors de cet événement, il est nécessaire d'inscrire au sein du gestionnaire global d'événements de l'application nos gestionnaires d'événements dédiés, présents dans le module HTTP.

Dans notre exemple, nous retrouvons deux gestionnaires. Ceux-ci sont associés aux événements *BeginRequest* et *PostAcquireRequestState* de l'*application* dans la fonction *Init* qui est appelée par le serveur IIS lors du chargement du module (lignes 5 à 11).

```
1  namespace MandatoryController
2  {
3   public class MandatoryController : IHttpModule
4   {
5    public void Init(HttpApplication application)
6    {
7     /* Inscription des gestionnaires d'evenements */
8     application.BeginRequest += new EventHandler(Appli_BeginRequest);
9     application.PostAcquireRequestState +=
10         new EventHandler(Appli_PostAcquireRequestState);
11   }
12
13   private void Appli_BeginRequest(object source, EventArgs e)
14   {
15    //Traitement au debut de la requete.
16   }
17
18   private void Appli_PostAcquireRequestState(object source, EventArgs e)
19   {
20   //Traitement apres chargement de la session.
21   }
22  }
23 }
```

Listing 4.2 – Exemple de module HTTP en C#.

4.2.2 Choix d'implémentation

Pour simplifier l'écriture des politiques de sécurité, nous utilisons les deux notations permises par notre langage de protection et permettant de factoriser un ensemble de ressources dans un même contexte objet. Il s'agit en quelque sorte d'expressions régulières simplifiées.

La première notation * nous permet de représenter un ensemble de ressources relatives à une ressource parent. L'objet suivant représente par exemple l'ensemble des fichiers contenus dans le répertoire */dynamic*.

$$file : \textbf{/dynamic/*}$$

Par exemple, la règle suivante autorise à tous les utilisateurs possédant le rôle *inscrit* l'accès à l'ensemble des fichiers du répertoire */dynamic*.

$$allow(* : inscrit, read, file : \textbf{/dynamic/*})$$

La seconde notation est dédiée aux objets de type *file* et *page* et permet de regrouper dans un même contexte de sécurité l'ensemble des objets possédant la même extension. L'objet suivant regroupe par exemple toutes les pages *aspx* de l'application présentes dans le répertoire */Dynamic/admin*.

$$page : /dynamic/admin/\textbf{*.aspx}$$

La règle suivante autorise tous les utilisateurs possédant le rôle *admin* à exécuter toutes les pages *aspx* de l'application se trouvant dans le répertoire */Dynamic/admin*.

$$allow(* : admin, execute, page : /dynamic/admin/\textbf{*.aspx})$$

L'utilisation de ce type de notations pour factoriser l'écriture des règles d'autorisation met en évidence deux façons différentes de gérer ces règles factorisées :

Développement lors de la compilation.

Une première façon de gérer les notations factorisées serait de développer lors de la compilation de la politique chaque règle contenant une factorisation des objets pour la remplacer par l'ensemble des règles d'autorisation correspondantes. Une règle factorisée serait donc remplacée par un ensemble de règles énumérant chaque ressource correspondant à la factorisation. Par exemple, pour un répertoire */dynamic/admin/* contenant trois fichiers aspx *menu.aspx*, *gestion_serveur.aspx* et *config_workflows.aspx*, la règle d'autorisation précédente serait remplacée par les trois règles suivantes :

$$allow(* : admin, execute, page : /dynamic/admin/\textbf{menu.aspx})$$
$$allow(* : admin, execute, page : /dynamic/admin/\textbf{gestion\_serveur.aspx})$$
$$allow(* : admin, execute, page : /dynamic/admin/\textbf{config\_workflows.aspx})$$

Calcul dynamique des expressions factorisées.

La seconde façon de gérer les contextes de sécurité factorisés est de calculer à chaque requête l'ensemble des expressions de factorisations dans lesquelles la ressource concernée peut intervenir. En effet, plusieurs factorisations peuvent faire intervenir une ressource donnée. Par exemple, pour le fichier */dynamic/admin/menu.aspx*, il existe quatre expressions factorisées dans lesquelles cette ressource peut intervenir :

$$/dynamic/admin/\textbf{*.aspx}$$
$$/dynamic/admin/\textbf{*}$$
$$/dynamic/\textbf{*}$$
$$/\textbf{*}$$

Le fait de n'utiliser que deux types de factorisations des règles d'autorisations permet de borner, et donc de rendre calculable, l'ensemble des expressions de factorisations faisant intervenir une ressource donnée. En effet, si nous avions utilisé un système d'expressions régulières complet, il aurait été difficile de calculer, pour une ressource donnée et à chaque requête entrante sur le serveur, l'ensemble des factorisations dans lesquelles la ressource peut intervenir.

La première méthode présentée est plus contraignante que la seconde. En effet, le développement des règles factorisées ne serait effectué qu'une seule fois, lors de la compilation

de la politique. Tout ajout ou suppression d'une ressource après la compilation de la politique ne serait pas correctement traité. Il serait donc nécessaire de recompiler la politique de sécurité à chaque ajout ou suppression d'une ressource. Nous avons donc fait le choix d'implémenter la seconde méthode qui facilite l'administration des politiques de sécurité.

4.2.3 Architecture détaillée

La figure 4.3 présente une vue précise de l'architecture de notre protection obligatoire dynamique.

À l'arrivée d'une requête HTTP sur le serveur, le moniteur de référence doit tout d'abord calculer les contextes de sécurité sujets et objets associés à la requête. Pour cela, un gestionnaire d'événement est utilisé pour calculer les contextes de sécurité associés à l'état de la requête, et en déduire la liste des *règles potentielles d'accès* pour la requête considérée.

Ensuite, le gestionnaire d'événements exécute le moteur de décision, qui va comparer la politique de sécurité à la liste des règles potentielles d'accès. Ainsi, le moteur de décision pourra décider si l'accès est autorisé ou non. Enfin, le moniteur de référence bloque la requête entrante si l'accès est interdit, ou laisse le serveur traiter la requête si celle-ci est légitime vis-à-vis de la politique de sécurité.

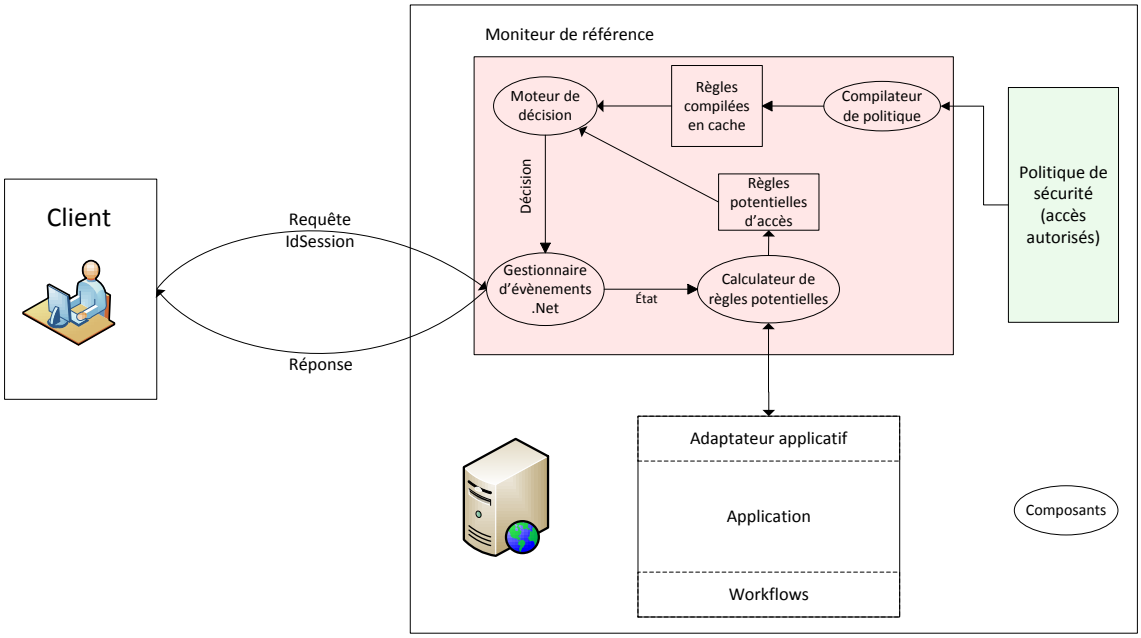

FIGURE 4.3 – Architecture détaillée du système de contrôle d'accès.

Nous allons maintenant expliciter dans des parties distinctes le fonctionnement de chacun des composants de notre architecture détaillée.

4.2.4 Gestionnaire d'événements

Au fil du traitement d'une requête HTTP, le framework .Net fourni un ensemble d'événements tels que *BeginRequest, AcquireRequestState,etc.* associés à différents états de la requête. Un gestionnaire dédié peut être associé à chacun de ces événements. Le tableau 4.1 présente la liste des différents événements déclenchés chronologiquement au cours du traitement d'une requête HTTP.

Notre moniteur de référence est implémenté comme un ensemble de gestionnaires associés aux événements fournis par le framework .Net. Ces gestionnaires d'événements sont capables d'extraire des informations sur la requête en cours de traitement (l'état associé à la requête), mais permet également de modifier l'état de la requête et le résultat du traitement de la requête, retourné au client. Il est donc non seulement possible de bloquer une requête entrante sur le serveur sur un de ces évènements, mais il est également possible de modifier l'état de la requête entrante.

Le gestionnaire associé à l'événement *PostAcquireRequestState* exécute tous les composants de notre moniteur de référence. En effet, *PostAcquireRequestState* est le premier événement pour lequel il est possible d'accéder à l'ensemble de l'état de la requête entrante. Dans les bonnes pratiques, l'événement *AuthorizeRequest* est l'événement dans lequel il faudrait placer les mécanismes d'autorisations.

Cependant, la session utilisateur étant chargée uniquement lors de l'événement *AcquireRequestState*, nous sommes contraints d'utiliser l'événement *PostAcquireRequestState* pour réaliser l'autorisation d'une requête. Ainsi, notre moniteur de référence couvre tous les cas possibles lors de l'événement *PostAcquireRequestState*.

Nom de l'évènement	**Description**
BeginRequest	Signale une nouvelle demande.
AuthenticateRequest	Signale que la demande est prête à être authentifiée.
AuthorizeRequest	Signale que la demande est prête à être autorisée.
ResolveRequestCache	Utilisé par le module de cache de sortie pour court-circuiter le traitement des demandes qui ont été mises en cache.
AcquireRequestState	Récupération de l'état de session associé à la demande.
PreRequestHandlerExecute	Signale que le gestionnaire de demande est sur le point de s'exécuter.
Traitement effectif de la requête	
PostRequestHandlerExecute	Signale que le gestionnaire de demande a fini de traiter la demande.
ReleaseRequestState	Signale que l'état de session doit être stocké parce que l'application en a terminé avec la demande.
UpdateRequestCache	Signale que le traitement du code est achevé et que le fichier est prêt à être ajouté au cache ASP.NET.
EndRequest	Signale la fin de tous les traitements pour la demande. Il s'agit du dernier événement appelé.

TABLE 4.1 – Événements .Net lors du traitement d'une requête HTTP.

À cet instant, le moniteur de référence dispose donc de toutes les informations nécessaires au calcul des règles potentielles d'accès relatives à la demande en cours (cf. 4.2.6) et à la prise de décision d'autorisation de la demande (voir 4.2.7). Le moteur de décision peut donc vérifier que l'accès demandé est autorisé par la politique. Si le moteur de décision refuse une requête, le moniteur de référence peut retourner un message d'erreur (par exemple *Vous n'avez pas les droits suffisants pour accéder à cette ressource*) ou un code d'erreur HTTP *403 - accès refusé* et la requête n'est pas traitée par le serveur.

4.2.5 Compilateur de politiques

Les politiques de sécurité sont exprimées au format texte par le biais de notre langage dédié. Elles sont donc stockées dans des fichiers texte sur le serveur.

Le but du compilateur de politique est de transformer un fichier de politique de sécurité en une version utilisable par le moniteur de référence.

Grâce à notre grammaire définie dans 3.3.1, l'outil ANTLR (pour ANother Tool for Language Recognition) nous permet de générer un analyseur syntaxique (parser) et un analyseur lexical (lexer) en langage C# permettant d'analyser une politique de sécurité exprimée dans cette grammaire.

La compilation est effectuée par un gestionnaire d'évènement associé à l'évènement *Init*. Cet évènement apparait lors du chargement initial du module de contrôle d'accès par le serveur Web. Les règles exprimées dans un fichier de politique de sécurité sont stockées dans des objets de type *Rule* dont une partie de la structure est exposée dans le listing 4.3.

```
1  class Rule
2  {
3    private byte[] P_HashedRule;
4    private String P_TextRule;
5    private List<String> P_Conditions;
6
7    public byte[] HashedRule
8    { get { return P_HashedRule; } }
9
10   public String TextRule
11   { get { return P_TextRule; } }
12
13   public List<String> Conditions
14   { get { return P_Conditions; } }
15
16   /* Constructeur */
17   public Rule(byte[] hashedRegle, String textRegle, String[] conditions)
18 }
```

Listing 4.3 – Structure des règles en cache.

Le compilateur effectue un hachage via l'algorithme SHA-512 des règles présentes dans la politique (voir lignes 2 et 6 du listing 4.4) . Ces règles hachées sont stockées en mémoire dans le cache du serveur Web, ce qui permet d'accéder directement en mémoire à la politique de sécurité lors de la prise de décision d'autorisation. L'utilisation de règles hachées stockées dans le cache du serveur Web permet de réduire le temps de recherche d'un accès dans la politique vis-à-vis d'une recherche directe dans le fichier textuel de politique. Les objets *Rule* sont stockés dans le cache du serveur dans un objet *Dictionary* (voir ligne 11 du listing 4.4).

Les objets *Dictionary* (ou dictionnaires) représentent une collection de couples (clé, valeur). Dans notre cas, la clé du dictionnaire est la version hachée de la règle d'autorisation (de type *byte[]*) et la valeur stockée est un objet de type *Rule*.

L'intérêt d'utiliser un objet de type dictionnaire est le fait que ces objets fournissent des méthodes de recherche de valeurs (notamment la méthode *TryGetValue(clé)*) en temps constant, ce qui permet une prise de décision efficace et indépendante de la taille de la politique de sécurité (voir 4.2.7). Nous reviendrons sur ces aspects de performance dans le chapitre 5.

```
1  UTF8Encoding encoder = new UTF8Encoding();
2  SHA512Managed hasher = new SHA512Managed();
3  String strRegle = rule.text.ToLower();
4
5  /* calcul du Hash */
6  byte[] regleHash = hasher.ComputeHash(encoder.GetBytes(strRegle));
7  /* Creation d'une nouvelle regle */
8  Rule regle = new Rule(regleHash,strRegle, conditions);
9
10 /* Recuperation des regles en cache et ajout de la nouvelle regle */
11 Dictionary<byte[], Rule> politique = HttpContext.Current.Cache["allow"];
12 politique.Add(regleHash,regle);
```

Listing 4.4 – Méthode de compilation des règles d'autorisation.

La compilation des fichiers de politique est effectuée lors du chargement du module de protection obligatoire (cf. listing 4.2 fonction *Init*). Cependant, il est aussi possible de forcer la compilation des fichiers de politique et le rechargement des règles en cache en utilisant un paramètre d'URL particulier et configurable, le paramètre par défaut étant *reload_policy=1*. L'ajout de ce paramètre à n'importe quelle requête HTTP entrante est automatiquement détecté par le moniteur de référence et déclenche la compilation et le rechargement en cache de la politique de sécurité.

4.2.6 Calcul des règles potentielles d'accès

Pour décider d'autoriser ou non une requête entrante, il est nécessaire de calculer l'ensemble des règles potentielles associées à cette requête, qui seront ensuite comparées à la politique de sécurité compilée et disponible dans le cache du serveur d'applications. Pour effectuer ce calcul des règles potentielles d'accès relatives à une requête HTTP entrante sur le serveur, il convient tout d'abord de calculer les contextes de sécurité objets et sujets associés à cette requête.

Calcul des contextes de sécurité objets

Le but ici est de calculer l'ensemble des contextes de sécurité objets pouvant représenter l'objet associé à la requête HTTP entrante.

Pour chaque requête, il est donc nécessaire de calculer l'ensemble des contextes factorisés auxquels peut être associée la ressource requise. Pour l'implémentation de cette méthode de calcul, nous introduisons la notion de *profondeur de calcul*.

La profondeur de calcul représente le niveau le plus haut auquel le calculateur de contextes objets va remonter, c'est-à-dire le nombre d'itérations de calcul des contextes factorisés. Une profondeur de 1 représente donc uniquement la ressource. Une profondeur de 2 représente le niveau immédiatement supérieur, et ainsi de suite. La table 4.2 illustre par un exemple cette notion de profondeur de calcul pour la ressource de type *file* et d'identifiant */Dynamic/Modeliseur/images/workflow.png*.

Profondeur	Contextes de sécurité objets
1	file :/Dynamic/Modeliseur/images/workflow.png
2	file :/Dynamic/Modeliseur/images/* file :/Dynamic/Modeliseur/images/*.png
3	file :/Dynamic/Modeliseur/*
4	file :/Dynamic/*
5	file :/*

TABLE 4.2 – Exemple illustrant la notion de profondeur.

Une profondeur de calcul trop importante reflète généralement d'une mauvaise hiérarchisation des ressources et peut augmenter le temps de prise de décision de l'autorisation d'accès à une ressource. Par exemple, sur les systèmes de contrôle d'accès obligatoire type SELinux, l'autorisation d'accès à un fichier nécessite de vérifier les autorisations pour tous les répertoires parents de ce fichier. Une hiérarchisation importante augmente donc le nombre d'autorisations à vérifier et donc le temps d'autorisation. D'un point de vue système, une bonne pratique est donc de limiter le nombre de sous-répertoires dans l'arborescence de fichier.

L'algorithme 2 expose la méthode de calcul de l'ensemble des contextes de sécurité objets associés à une requête HTTP entrante. Nous y retrouvons le calcul des contextes factorisés relatifs à la ressource requise, en prenant en compte la profondeur de calcul. Pour le calcul des contextes factorisés, il est tout d'abord nécessaire d'extraire de la requête l'identifiant de l'objet accédé. Il est également nécessaire de récupérer l'extension de l'objet dans le cas d'un objet de type *page* ou *file*. Ensuite, l'algorithme calcule les contextes objets associés en remontant au niveau supérieur de manière itérative, et ce jusqu'à atteindre la profondeur de calcul ou la racine de l'objet.

Complexité Le nombre maximal de contextes de sécurité objets retournés par cette algorithme est linéairement dépendant de la profondeur de calcul. Chaque niveau nous retourne un contexte de sécurité objet, exception faite du niveau 2 qui en retourne deux. Si la profondeur de calcul p est supérieure ou égale à 2, le nombre maximal de contextes de sécurité objets est donc de $c = p + 1$. Pour une profondeur de 1, cet algorithme ne retourne qu'un seul contexte de sécurité objet.

Les calculs de l'identifiant de la ressource, de l'extension, de la racine et du parent d'un objet à partir de son identifiant ne dépendent pas du nombre de ressources, du nombre d'utilisateur ou du nombre de rôles et sont donc de complexité $O(1)$. Pour une application donnée la profondeur de calcul p est une constante. Cet algorithme possède donc une complexité indépendante du nombre de ressources, d'utilisateurs et de rôles.

Calcul des contextes de sécurité sujets - Adaptateur applicatif

Comme nous l'avons présenté dans la partie 3.2, l'adaptateur applicatif permet d'assurer l'indépendance du moniteur de référence vis-à-vis de l'application contrôlée. Il permet au moniteur de référence de retrouver les identifiants de l'utilisateur et ses rôles, c'est-à-dire les informations minimales nécessaires au calcul des contextes sujets associés à la requête en cours d'autorisation.

L'adaptateur applicatif doit donc fournir deux méthodes : la première méthode, *getUser()*, permet de retrouver l'identifiant de l'utilisateur à partir de l'état de la requête (c'est-à-dire notamment l'url de la requête, les données de formulaires, la session utilisa-

Algorithme 2 : Calcul des contextes de sécurité objets.

```
Données : Profondeur
Entrées : Requête
Résultat : Liste des contextes objets

identifiant = ExtraireIdentifiantRessource(Requête)
extension = ExtraireExtension(identifiant)
racine = ExtraireRacine(identifiant)

si Profondeur >= 1 alors
  | ajouter identifiant dans contextes
fin

si Profondeur >= 2 alors
  | chemin = Parent(identifiant)
  | ajouter chemin + "*" dans contextes
  | ajouter chemin + "*" + extension dans contextes
fin

si Profondeur > 2 alors
  | boucle = 1
  | tant que (boucle + 2) < Profondeur et identifiant <> racine faire
  |   | identifiant = Parent(identifiant)
  |   | ajouter identifiant + "*" dans contextes
  |   | boucle += 1
  | fin
fin
retourner contextes
```

teur, et le cache applicatif). La méthode *getRoles()* renvoie quant à elle la liste des rôles applicatif associés à l'utilisateur.

Le listing 4.5 présente l'implémentation de la fonction *GetUser* pour l'application QualNet. Au niveau ASP.Net, l'adaptateur applicatif s'exécute au sein de l'application et peut donc accéder de manière directe à l'état de la requête. C'est pourquoi dans notre implémentation l'état de la requête n'est pas passé en paramètre des fonctions *getUser* et *getRoles*.

```
1  using Qualnet.BL;
2
3  public class AdaptateurApplicatif
4
5    public String getUser()
6    {
7      String l_login = UtilisateurConnecte.GetLogin();
8      if (l_login.Equals("No Login") || String.IsNullOrEmpty(l_login))
9        return "?";
10
11     return l_login
12   }
```

Listing 4.5 – Adaptateur Applicatif pour l'application QualNet - Fonction GetUser.

L'adaptateur applicatif peut donc notamment accéder au cache applicatif, dans lequel sont stockés les objets *UtilisateurConnecte* qui, dans les applications QualNet, représentent chaque utilisateur connecté et contient entre autre son identifiant d'utilisateur (*Login*).

Pour autoriser un accès à une ressource, le moteur de décision effectue un nombre de comparaisons, entre les règles potentielles d'accès et la politique de sécurité, qui est linéairement dépendant du nombre de rôles retournés par l'applicateur adaptatif (cf. Algorithme 4). Nous avons donc envisagé deux versions d'adaptateur applicatif.

La première solution pour l'adaptateur applicatif consiste à retourner tous les rôles applicatifs de l'utilisateur associé à la requête. Cette liste de rôles est donc indépendante de la ressource à laquelle l'utilisateur tente d'accéder. Cependant, pour une ressource donnée certains de ces rôles ne permettront jamais d'accéder à cette ressource. Il s'agit donc de réduire la liste des rôles retournée par l'adaptateur applicatif aux rôles donnant effectivement accès à la ressource considérée.

Nous proposons donc une seconde version plus intelligente de l'adaptateur applicatif, qui consiste à retourner uniquement les rôles permettant l'accès à la ressource considérée. Ainsi, le nombre de rôles retournés par l'adaptateur applicatif intelligent est moindre, ce qui limite le nombre de comparaisons et accélère donc la prise de décision d'autorisation.

Le listing 4.6 révèle un extrait de la fonction *getRoles* pour l'application QualNet. Nous retrouvons dans cet extrait de code la classe *UtilisateurConnecte* qui nous permet de retrouver les rôles de l'utilisateur, et qui retourne pour chaque requête l'ensemble des rôles de l'utilisateur sous forme d'une liste de chaînes de caractères. Une première optimisation pourrait être de ne renvoyer que les rôles associés à la partie *Dynamic* de l'application si la ressource accédée intervient uniquement dans cette partie de l'application, et de même pour la partie *Doc*. Un rôle associé à *Dynamic* ne pourra jamais accéder à une ressource de la partie *Doc*, ce qui est une des contraintes appliquées par cet adaptateur intelligent.

```
1  public String getRoles()
2  {
3    List<String> roles = new List<String>();
4
5    //Recuperation des roles Dynamic
6    if (UtilisateurConnecte.IsAdmin)
7      roles.Add("admin dyn");
8    else if (UtilisateurConnecte.IsConcepteur)
9      roles.Add("concepteur");
10   if (UtilisateurConnecte.IsInscrit)
11     roles.Add("inscrit");
12
13   //Recuperation des roles Doc
14   foreach (String l_Role in UtilisateurConnecte.GetListeRolesDoc())
15     roles.Add(l_Role.ToLower());
16
17  /* ..... */
18   }
```

Listing 4.6 – Extrait de l'adaptateur Applicatif QualNet - Fonction GetRoles simple.

Nous montrerons dans les expérimentations qu'il est possible de développer un adaptateur applicatif intelligent de ce genre pour un environnement de workflows, et nous verrons que le gain sur le temps de prise de décision d'autorisation avec un tel adaptateur vis-à-vis d'un adaptateur simple est substantiel.

Règles potentielles d'accès

L'algorithme 3 décrit le mécanisme de calcul dynamique des règles potentielles correspondant à une requête entrante sur le serveur Web.

Tout d'abord, le type d'action associé à la requête doit être défini. Si la requête entrante concerne une page web physique du serveur, il s'agit d'une action d'exécution, notée *execute*. Pour les autres ressources physiques n'ayant pas d'impact direct sur le système tels que les images ou les fichiers CSS, il s'agira d'une action de lecture notée *read*.

Ensuite, le calculateur de règles potentielles d'accès récupère grâce à l'adaptateur applicatif les identifiants d'utilisateur et les rôles correspondants à la requête en cours d'autorisation. Il récupère également l'ensemble des contextes de sécurité objets correspondant à la requête, calculés grâce à l'algorithme 2. Enfin, pour chaque contexte de sécurité objet, l'algorithme calcule l'ensemble des règles potentielles correspondantes.

A cet instant, il convient de différencier deux cas :

– l'utilisateur est connu et authentifié dans l'application.

– l'utilisateur est inconnu dans l'application.

Si l'utilisateur est inconnu, et par conséquent ne possède aucun rôle applicatif, il n'existe qu'une seule règle d'accès possible : un utilisateur *inconnu* avec le rôle *inconnu* tente d'effectuer l'action considérée sur la ressource demandée :

$$allow(? :?, Action, Ressource)$$

Au contraire, pour un utilisateur connu et authentifié, plusieurs règles potentielles différentes sont possibles, car l'utilisateur peut posséder plusieurs rôles au sein de l'application.

Pour chaque rôle, deux règles potentielles peuvent autoriser l'accès à la ressource :

– n'importe quel utilisateur *connu* avec un *rôle* applicatif particulier pourrait effectuer une *action* sur la *ressource*.

$$allow(* : rôle, Action, Ressource)$$

– uniquement *l'utilisateur* authentifié avec un *rôle* applicatif particulier pourrait effectuer une *action* sur la *ressource*.

$$allow(utilisateur : rôle, Action, Ressource)$$

De plus, deux autres règles indépendantes des rôles applicatifs sont possibles :

– uniquement *l'utilisateur* authentifié avec n'importe quel *rôle* applicatif serait autorisé.

$$allow(utilisateur : *, Action, Ressource)$$

– n'importe quel utilisateur *authentifié*, avec n'importe quel *rôle* applicatif serait autorisé.

$$allow(* : *, Action, Ressource)$$

Complexité Soit r le nombre de rôles retournés par l'adaptateur applicatif, et c le nombre de contextes de sécurité objets relatifs à la ressource accédée. Nous avons vu que le nombre de contextes objets retournés par l'algorithme 2 est $c = p + 1$. La profondeur de calcul est une constante p fixée pour chaque application. Le nombre maximal de contextes de sécurité objets est donc constant pour une application donnée.

Le nombre maximal de règles potentielles calculées par cet algorithme dépend linéairement du nombre de contextes de sécurité objets, mais également du nombre de rôles applicatifs retournés par l'adaptateur. Pour chaque rôle applicatif, deux règles potentielles sont créées. En plus des règles potentielles dépendantes des rôles, deux règles potentielles supplémentaires sont calculées. Pour chaque contexte de sécurité objets, le nombre maximal de règles potentielles calculées est donc de $2r + 2$.

Le nombre maximal de règles potentielles calculées par cet algorithme est donc de $c(2r + 2)$. La complexité de cet algorithme est donc :

$$O(c(2r + 2)).$$

Algorithme 3 : Calcul des règles potentielles d'accès.

Entrées : Requête, État, Ressource
Résultat : Liste des règles potentielles d'accès

si *Ressource* ***est*** *une page Web* **alors**
  Action = "execute"
sinon
  Action = "read"
fin

Utilisateur = appAdaptor.getUser(État)
Rôles = appAdaptor.getRôles(État)
Objets = getContextesObjets(Requête)

pour chaque *Objet* ***dans*** *Objets* **faire**
  si *Utilisateur* ***est*** *inconnu* **alors**
    règle = genererRègles(?, ?,*Action*,*Objet*)
    ajouter règle **dans** *ReglesPotentielles*
  sinon
    pour chaque *rôle* ***dans*** *Rôles* **faire**
      règle = genererRègles(*,*rôle*,*Action*,*Objet*)
      ajouter règle **dans** *ReglesPotentielles*
      règle = genererRègles(*Utilisateur*,*rôle*,*Action*,*Objet*)
      ajouter règle **dans** *ReglesPotentielles*
    fin

    règle = genererRègles(*Utilisateur*,*,*Action*,*Objet*)
    ajouter règle **dans** *ReglesPotentielles*
    règle = genererRègles(*,*,*Action*,*Objet*)
    ajouter règle **dans** *ReglesPotentielles*

  fin
fin
retourner *ReglesPotentielles*

Fonction genererRègles(*Utilisateur*,*Rôle*,*Action*,*Ressource*)
  sujet = "*Utilisateur* : *Rôle*"
  accès = allow(*Sujet*,*Action*,*Ressource*)
  retourner accès
Fin Fonction

4.2.7 Moteur de décision

L'algorithme 4 présente le mécanisme de décision d'autorisation d'accès à une ressource. Il est en charge de décider si l'accès demandé par un sujet sur un objet, en accord avec l'état de la requête traitée, est autorisé par la politique de sécurité.

Autorisations

Le moteur de décision utilise les règles potentielles d'accès calculées par l'algorithme 3 et les compare aux règles d'autorisations définies dans la politique de sécurité compilée. Comme nous avons pu énumérer exhaustivement toutes les règles pouvant potentiellement autoriser l'accès à la ressource considérée, il suffit qu'une de ces règles potentielles soit présente dans la politique de sécurité pour que l'accès soit autorisé. Ainsi, à la première correspondance, le moteur de décision autorise le traitement de la requête, qui continue de manière transparente pour l'utilisateur. Dans le cas d'une action conditionnelle, le moniteur de référence vérifie également les conditions associées à l'action requise. Si ces conditions sont satisfaites, la requête est autorisée.

Sinon, le moteur de décision recherche la règle potentielle suivante dans la politique de sécurité jusqu'à trouver une règle potentielle concordante avec la politique de sécurité.

Refus

Si aucune des règles potentielles d'accès n'est trouvée dans les règles d'autorisations de la politique de sécurité, le moteur de décision refuse l'accès à la ressource. Le moniteur de référence arrête le traitement de la requête et retourne au client un message d'erreur générique et configurable, du type *Vous n'avez pas les droits nécessaires pour accéder à cette ressource.*

Algorithme 4 : Prise de décision.

```
Entrées : ToutLesAccès, Politique
Résultat : Booléen : accès autorisé ?
pour chaque accès dans ToutLesAccès faire
    si accès est dans Politique alors
        si accès a des conditions alors
            si les conditions sont vérifiées alors
                retourner Vrai
            sinon
                continuer
            fin
        sinon
            retourner Vrai
        fin
    fin
fin
retourner Faux
```

Complexité

Chaque règle potentielle calculée est recherchée dans la politique de sécurité compilée en cache. Dans le pire cas, c'est-à-dire un refus, toutes les règles potentielles calculées vont être recherchées dans la politique de sécurité compilée et aucune correspondance ne sera trouvée. Le nombre maximal de recherches effectuées par l'algorithme de décision est égal au nombre de règles potentielles générées par l'algorithme 3. Le nombre maximal de règles potentielles calculées est de $c(2r+2)$, avec r le nombre de rôles retournées par l'adaptateur applicatif et c le nombre de contextes de sécurité objets associés à l'objet accédé (qui est constant pour une application donnée).

De plus, l'utilisation d'objets de type *Dictionnary* permet la recherche d'une règle potentielle d'accès dans la politique de sécurité en temps constant. La complexité de l'algorithme de décision est donc uniquement dépendante du nombre de règles potentielles d'accès, celle-ci est donc :

$$O(c(2r+2))$$

Il est important de noter que la complexité de l'algorithme de la prise de décision n'est pas dépendante de la taille de la politique de sécurité, ce qui permet l'utilisation de politiques de sécurité de tailles importantes sans impact sur les performances. Nous vérifierons ce point dans le chapitre 5.

Cette complexité algorithmique fait donc apparaître deux façons de réduire le temps de prise de décision d'autorisation ou de refus de la requête entrante :

- Diminuer la profondeur de calcul ($c = Profondeur + 1$).
- Réduire le nombre de rôles retournés par l'adaptateur applicatif.

Réduire la profondeur de calcul impose de limiter l'utilisation de règles factorisées, ce qui par conséquent augmente le nombre de règles d'autorisations dans la politique de sécurité. Cela peut rendre la politique de sécurité plus lourde et donc plus difficile à maintenir.

Pour réduire le nombre de rôles retournés par l'adaptateur applicatif, nous avons vu dans la partie 4.2.6 qu'il est possible d'implémenter un adaptateur applicatif intelligent qui, pour un objet donné ne retourne que les rôles pouvant potentiellement autoriser l'accès à cet objet. Nous verrons dans le chapitre 5 le gain sur les performances de l'utilisation d'un adaptateur applicatif intelligent vis-à-vis d'un adaptateur applicatif simple.

4.3 Calculateur de politiques

Dans cette partie, nous présentons le modèle de workflows utilisé dans les applications QualNet. Nous détaillerons ce modèle ainsi que la méthodologie permettant de transposer ce modèle vers notre modèle spécifique de workflows. Nous développerons également la méthode de vérification de notre protection obligatoire proposée dans la partie 3.5 pour valider la sûreté de notre protection.

4.3.1 Modèle de workflow QualNet

La figure 4.4 présente une version épurée du modèle de données des workflows utilisés par les applications QualNet. Nous y retrouvons toutes les notions usuellement utilisées dans les workflows (notamment les notions d'étapes, de liaisons et de ressources). Nous allons détailler chaque table dans le but de présenter les notions de workflows qu'elles représentent.

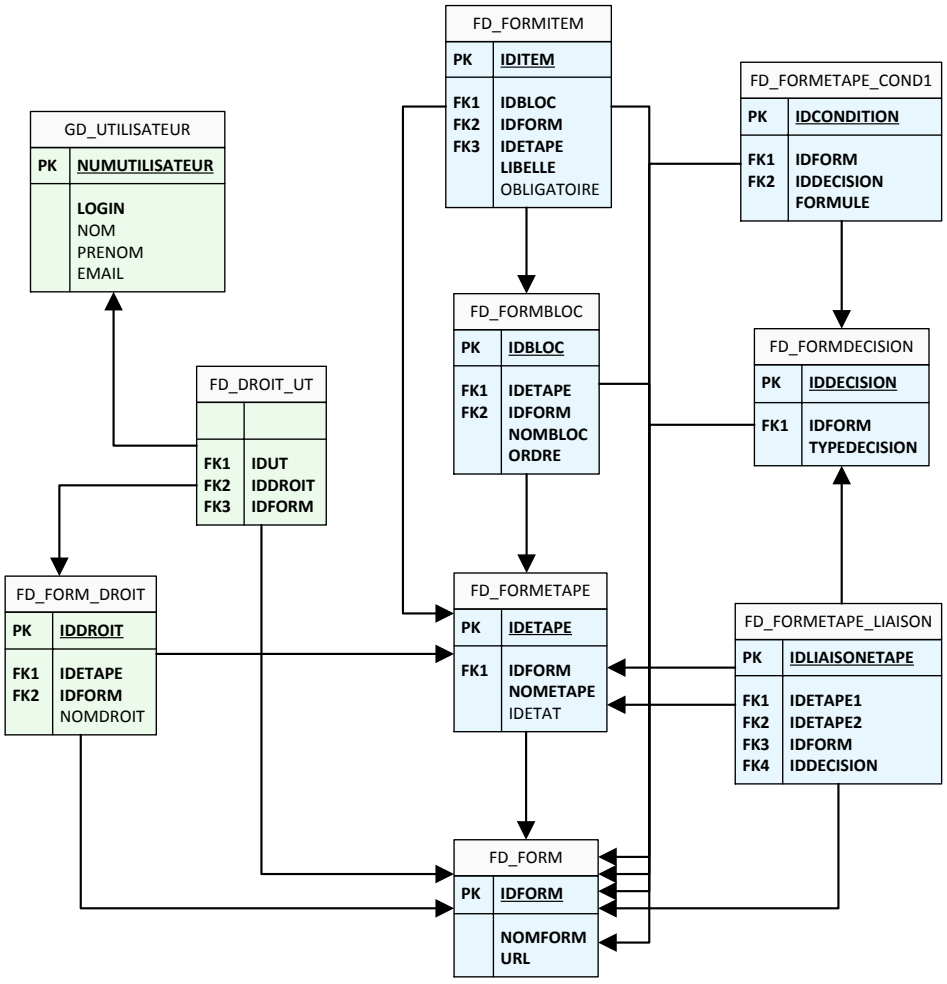

FIGURE 4.4 – Modèle de données des workflows QualNet.

GD_UTILISATEUR

Cette table modélise l'ensemble des utilisateurs de l'application. Chaque utilisateur y est représenté par un numéro d'identifiant unique (*NUMUTILISATEUR*) et un *LOGIN* de connexion sous la forme d'une chaîne de caractères. Nous y retrouvons toutes les informations relatives aux utilisateurs, et notamment leurs nom, prénom et email.

FD_FORM

Cette table représente l'ensemble des workflows de l'application avec leur nom et l'URL permettant d'y accéder. Chaque workflow possède un numéro d'identifiant unique *IDFORM*.

FD_FORMETAPE

Cette table représente toutes les étapes des workflows de l'application, qui possèdent un numéro d'identifiant unique *IDETAPE*. Chaque étape contient également un nom d'étape sous la forme d'une chaîne de caractères. Un workflow étant constitué d'un certain nombre d'étape s'exécutant chronologiquement, nous retrouvons dans cette table une clé étrangère vers le workflow auquel l'étape est rattachée, mais également une valeur entière représentant l'ordre d'exécution des étapes dans ce workflow (*IDETAT*).

FD_FORMETAPE_LIAISON

Cette table décrit les liaisons entre les différentes étapes d'un workflow c'est-à-dire l'enchainement des étapes au sein du workflow. Une liaison contient donc une étape de départ et une étape d'arrivé (*IDETAPE1* et *IDETAPE2*), associées à un workflow donné (*IDFORM*). Ces liaisons peuvent être conditionnelles. Les conditions sont représentées par les deux tables suivantes, et par la clé étrangère *IDDECISION*.

FD_FORMDECISION

Cette table représente les différents types de décisions qui peuvent être effectuées lors d'une liaison entre étapes. Ces décisions peuvent être un refus, un abandon, un court-circuit (saut d'étape), un embranchement en *OU* ou un embranchement en *ET*.

FD_FORMETAPE_COND1

Certaines décisions sont conditionnelles (par exemple les embranchement en *OU*). Cette table décrit ces conditions, qui sont représentées par une *formule*, généralement mathématique, s'appliquant sur les différents objets présents dans les formulaires.

FD_FORMITEM

Les formulaires des différentes étapes d'un workflow sont composés d'un certain nombre de ressources, appelées *items*. Ces items peuvent être de différents types (champ texte, numérique, image, etc.). Chaque item est nommé (champ *LIBELLE*), et lié à une étape particulière (champ *IDETAPE*) et donc un workflow particulier (*IDFORM*). Un item peut être obligatoire ou non.

FD_FORMBLOC

Au sein d'un formulaire, les items sont regroupés entre eux sous forme de *blocs*. Un formulaire est donc composé d'un certain nombre de blocs ordonnés (champ *ORDRE*) et nommés (champ *NOMBLOC*) regroupant des items ayant un même sens du point de vue fonctionnel.

FD_FORM_DROIT

Cette table représente les droits d'accès aux workflows (champ *IDFORM*) et aux étapes (champ *IDETAPE*). Chaque droit d'accès possède un nom, par exemple *M3_4* pour l'étape 4 du workflow 3.

FD_DROIT_UT

Enfin, cette table décrit la distribution des droits aux utilisateurs de l'application. Un droit (champ *IDDROIT*) est attribué à un utilisateur particulier (champ *IDUT*).

Fichiers générés

Le système de workflow QualNet utilise le modèle de données précédent pour générer sur le serveur d'applications Web différents fichiers physique. L'organisation de ces fichiers est présentée dans la figure 4.5. Nous notons entre accolades les données du modèle précédent qui sont utilisées pour la génération des noms des répertoires et des fichiers (sous la forme *TABLE.CHAMP*).

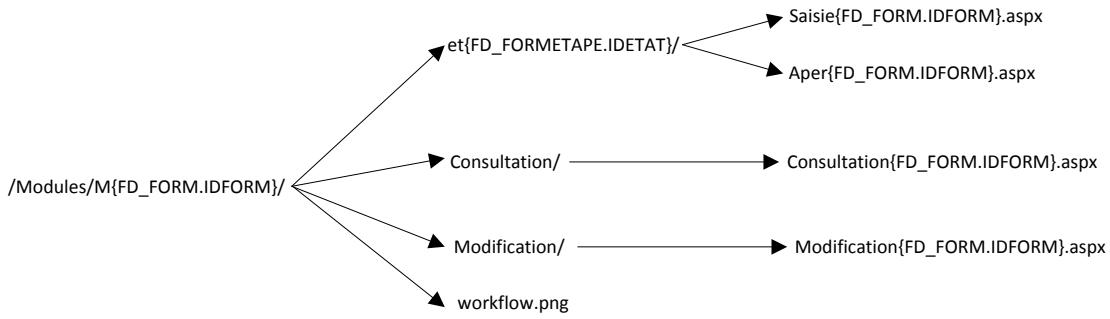

FIGURE 4.5 – Modèle de fichiers des workflows QualNet.

Les fichiers de chaque workflows sont générés dans un répertoire différent (les répertoires *M*). Pour chaque étape d'un workflow, un répertoire **et**, qui contient deux pages aspx est créé. Le fichier *Saisie.aspx* représente le formulaire de l'étape correspondante, qui contient tous les objets de types blocs et items du modèle de données. Le fichier *Aper.aspx* est utilisé comme aperçu lors de la modélisation des workflows.

Chaque répertoire de workflow contient également deux répertoires (*Consultation* et *Modification*). Les deux pages aspx contenues dans ces répertoires servent respectivement pour la consultation et la modification de fiches émises pour le workflow correspondant. Enfin, le fichier *workflow.png* est une image qui contient une représentation visuelle du workflow considéré.

4.3.2 Transposition vers le modèle spécifique de workflow

Pour rappel, la figure 4.6 décrit notre modèle spécifique d'accès pour les workflows nous permettant de calculer les politiques de sécurité. Dans le but de calculer les politiques de sécurité associées aux workflows QualNet, il est donc nécessaire de transposer le modèle de workflows QualNet vers notre modèle spécifique d'accès. Pour cela, nous allons donc présenter quelles sont, pour chaque table de notre modèle spécifique d'accès, les données du modèle de workflow QualNet qui correspondent.

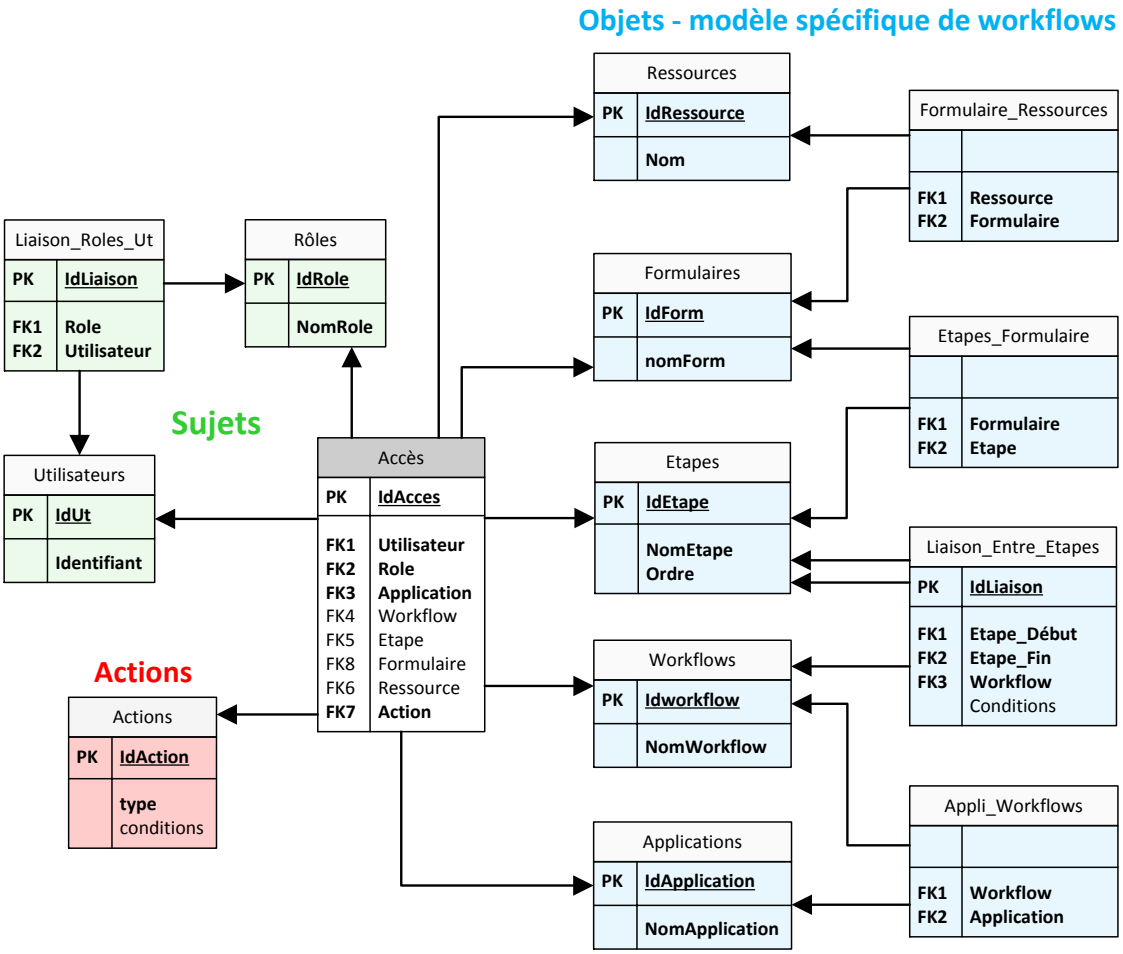

FIGURE 4.6 – Modèle spécifique d'accès pour les workflows.

Concernant les utilisateurs, la transposition du modèle QualNet vers notre modèle est simple. La clé primaire *NUMUTILISATEUR* correspond à la clé primaire de la table *Utilisateurs*, et nous utilisons le champ *LOGIN* comme identifiant de l'utilisateur (cf. figure 4.7).

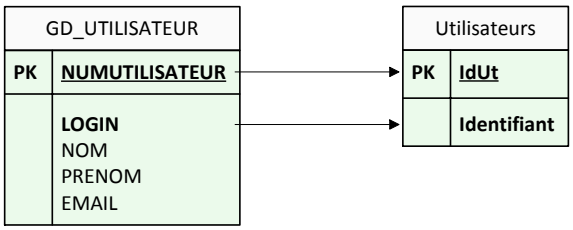

FIGURE 4.7 – Transposition vers la table Utilisateurs.

Dans le modèle de workflows QualNet, la notions de rôles est confondue avec la notion de droits. En pratique, à chaque étape de chaque workflow correspond un rôle (ou droit) particulier dont la possession autorise l'accès aux ressources de l'étape considérée. Ce rôle est noté de manière empirique $M\{IDWORKFLOW\}\_\{ETAPE.IDETAT\}$. Ainsi, les champs *IDDROIT* et *NOMDROIT* de la table *FD_FORM_DROIT* sont transposés vers la table *Rôles* de notre modèle (cf. figure 4.8).

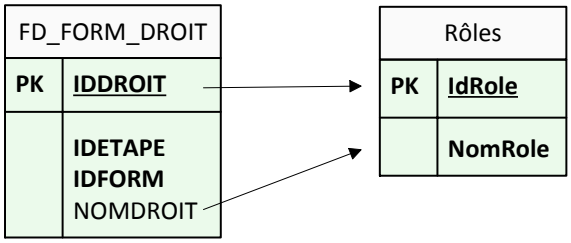

FIGURE 4.8 – Transposition vers la table Rôles.

La table *FD_FORM* du modèle de workflow QualNet contient toutes les informations concernant les workflows, et notamment un identifiant et un nom de workflow. La transposition vers la table *Workflows* de notre modèle est donc très simple (cf. figure 4.9) :

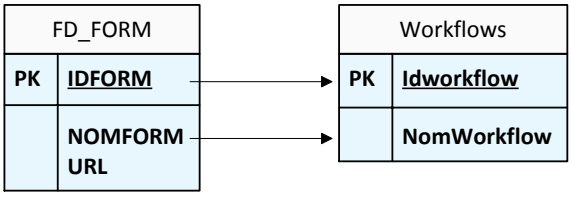

FIGURE 4.9 – Transposition vers la table Workflows.

La représentation des étapes des workflows est basée sur le même principe. Dans le modèle de workflows QualNet, la table *FD_FORMETAPE* représentent les étapes (cf. partie 4.3.1). Le champ *IDETAT* modélise l'ordre d'exécution des étapes au sein du workflow. La transposition vers la table *Étapes* de notre modèle est donc la suivante (cf. figure 4.10) :

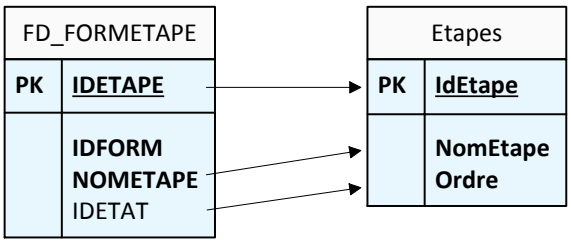

FIGURE 4.10 – Transposition vers la table Étapes.

Comme nous l'avons présenté dans la partie 4.3.1, dans le modèle de workflow QualNet à chaque étape de chaque workflow est associé un seul et unique formulaire. Chaque formulaire est composé d'un ensemble de *blocs* comprenant un certain nombres d'*items*. Les blocs et les items correspondent donc aux *ressources* de notre modèle spécifique de workflows.

Ainsi, nous regroupons les blocs et les items du modèle de workflows Qualnet dans la notion de ressources de notre modèle (cf. figure 4.11). Pour cela, nous utilisons comme clé primaire des blocs le champ *IDBLOC* et le champ *NOMBLOC* comme nom de ressource. Concernant les items, nous utilisons comme nom de ressource le champ *LIBELLE* des blocs. La clé primaire sera quant à elle la concaténation des champs *IDITEM* et *IDBLOC* de la table *FD_FORMITEM* pour éviter la duplication de clés. En effet, il est possible qu'une ressource de type bloc et une ressource de type item aient la même valeur de clé primaire.

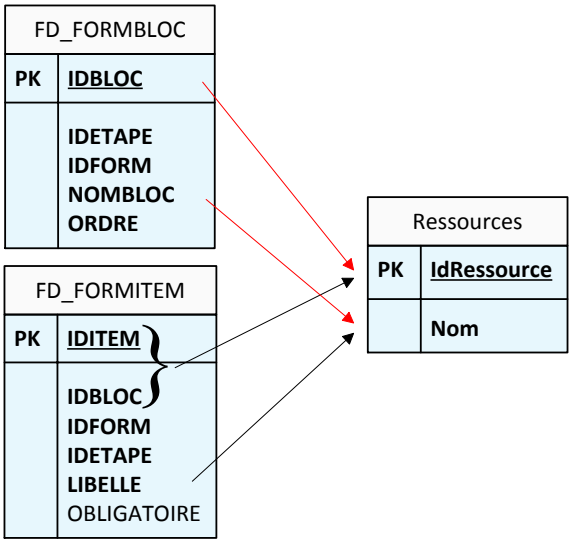

FIGURE 4.11 – Transposition vers la table Ressources.

Comme nous l'avons montré dans la partie 4.3.1, le modèle de données des workflows de QualNet ne contient pas de description des formulaires. Ceux-ci sont décrits dans le modèle de fichier (cf. figure 4.5). Ce modèle présente notamment le fichier *Saisie.aspx* qui correspond au formulaire physique associé à une étape particulière d'un workflow. Dans le modèle QualNet, un seul et unique formulaire est associé à chacune des étapes.

Ainsi, nous pouvons utiliser l'identifiant de l'étape à laquelle le formulaire est associé comme clé primaire de la table *Formulaires* de notre modèle spécifique de workflows, et le nom du fichier correspondant comme nom du formulaire (cf. figure 4.12).

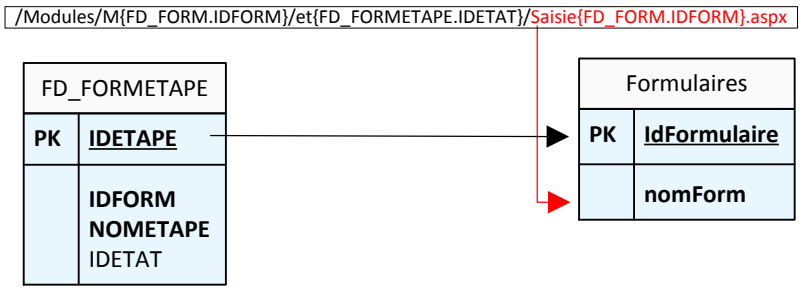

FIGURE 4.12 – Transposition vers la table Formulaires.

Dans le modèle de workflow QualNet, les conditions d'accès aux étapes sont modélisées par les tables *FD_FORMDECISION* et *FD_FORMETAPE_COND1*, et notamment le champ *FORMULE* de cette table qui représente de manière mathématique les conditions nécessaires au niveau des liaison entre les étapes d'un workflow (cf.figure 4.13).

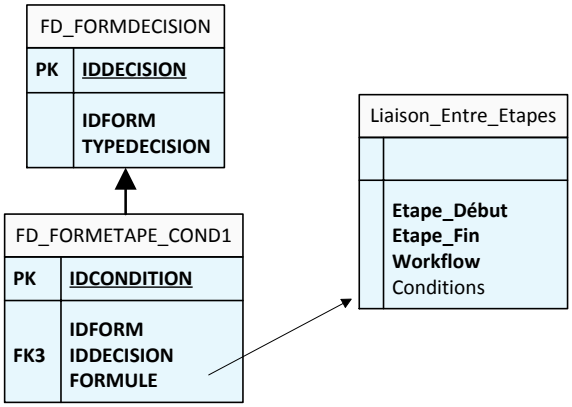

FIGURE 4.13 – Transposition des conditions.

Ainsi, par le biais de transpositions relativement simples à mettre en œuvre, nous avons pu retrouver toutes les informations nécessaires au modèle spécifique de workflows que nous proposons dans le modèle de workflows QualNet.

4.3.3 Calcul des règles

Dans la partie 3.4.1, nous avons présenté un *modèle spécifique d'applications* permettant de calculer les politiques de sécurité des applications reposant sur ce modèle. Nous avons également montré que nous pouvons transposer notre modèle spécifique de workflows (cf. figure 4.6) vers ce modèle spécifique d'application et donc calculer les politiques de sécurité pour les workflows s'appuyant sur ce modèle. Nous venons de voir que nous pouvons transposer le modèle de workflow QualNet vers notre modèle spécifique de workflows.

Ainsi, grâce à la méthode de calcul des règles à partir du modèle spécifique d'application que nous allons présenter, et par le jeu des transpositions, nous pouvons calculer les politiques de sécurité relatives aux workflows QualNet.

Pour rappel, la structure de la table représentant les accès dans notre modèle spécifique d'application est exposée dans la figure 4.14. Le modèle spécifique d'accès complet pour les applications est présenté dans la figure 3.7.

Accès	
PK	**IdAcces**
FK1	**Utilisateur**
FK2	**Rôle**
FK3	**Application**
FK4	Service
FK5	Fonctionnalité
FK6	Ressource
FK7	**Action**

FIGURE 4.14 – Table Accès du modèle spécifique d'application.

La requête SQL du listing 4.7 permet d'extraire l'ensemble des données nécessaires au calcul des règles d'autorisations associées à notre modèle spécifique d'application. Elle utilise des jointures sur les différentes tables associées aux clés étrangères de la table *Accès* pour retrouver toutes les informations nécessaires au calcul de la politique : l'identifiant de l'utilisateur, le nom du rôle qui lui est associé, le type d'action effectuée, les conditions pour effectuer cette action, et les noms des différentes entités relatives au modèle spécifique d'application.

```
1  SELECT Identifiant, NomRole, NomApplication, NomService,
2         NomFct, NomRessource, TypeAction, Conditions
3  FROM Accès
4  INNER JOIN Utilisateurs ON Utilisateurs.IdUt = Accès.Utilisateur
5  INNER JOIN Rôles ON Rôles.IdRôle = Accès.Rôle
6  INNER JOIN Applications ON Applications.IdApplication = Accès.Application
7  INNER JOIN Services ON Services.IdService = Accès.Service
8  INNER JOIN Fonctionnalités ON Fonctionnalités.IdFct = Accès.Fonctionnalité
9  INNER JOIN Ressources ON Ressources.IdRessource = Accès.Ressource
10 INNER JOIN Actions ON Actions.IdAction = Accès.Actions
```

Listing 4.7 – Requête SQL d'extraction des données.

L'algorithme 5 utilise cette requête pour calculer la politique de sécurité relative au modèle. Pour chaque enregistrement retourné par la requête, nous pouvons construire les contextes de sécurité sujets et objets, mais également définir le type d'action effectuée et retrouver les conditions relatives à cette action.

Le contexte de sécurité sujet est construit par la concaténation de l'identifiant de l'utilisateur et du rôle associé (cf. partie 3.3.1) :

$$C_S = Utilisateur : Rôle$$

Le contexte de sécurité objet est calculé à partir des paramètres permettant de représenter de manière sûre l'objet considéré. Le premier paramètre est donc le type de ressource, suivi par les noms d'application, de service, de fonctionnalité et de ressource associés à l'objet :

$$C_O = TypeRessource : NomApplication : NomService : NomFct : NomRessource$$

L'action effectuée est obtenu grâce au champ *TypeAction* de la table *Action* :

$$Action = Action.TypeAction$$

Enfin, les conditions sont obtenus à partir du champ *Conditions* de la table *Action* :

$$Conditions = Action.Conditions$$

Algorithme 5 : Calcul des politiques de sécurité.

Entrées : SQL, modèle spécifique
Résultat : Politique
Accès = retrouverAccès(modèle spécifique,SQL)
pour chaque *ligne L* ***dans*** *Accès* **faire**
    sujet = L.Identifiant + ' :' + L.NomRôle
    action = L.TypeAction
    objet = L.TypeRessource + ' :' + L.NomApplication + ' :' + L.NomService + ' :' + L.NomFct + ' :' + L.NomRessource
    condition = L.Conditions
    règle = allow(sujet,action,objet) : conditions
    ajouter règle **dans** Politique
fin
retourner Politique

4.3.4 Tests et vérification de la protection obligatoire

Dans le but de valider l'implémentation de notre protection obligatoire dynamique et de notre méthode de calcul des politiques, nous avons proposé dans la partie 3.5 une méthode générale de vérification de la protection en place. Nous détaillerons ici son implémentation, et un cas d'utilisation sera présenté dans le chapitre 5.

Architecture

Pour simuler de la manière la plus réelle possible l'application lors du test de couverture, nous utilisons une architecture client/serveur pour vérifier la couverture de notre moniteur de référence. Le client, représentant l'outil de test, effectue les requêtes HTTP associées aux accès nécessaires à l'algorithme 6 pour couvrir l'intégralité des accès nécessaires au parcours exhaustif de l'application testée (cf. figure 4.15).

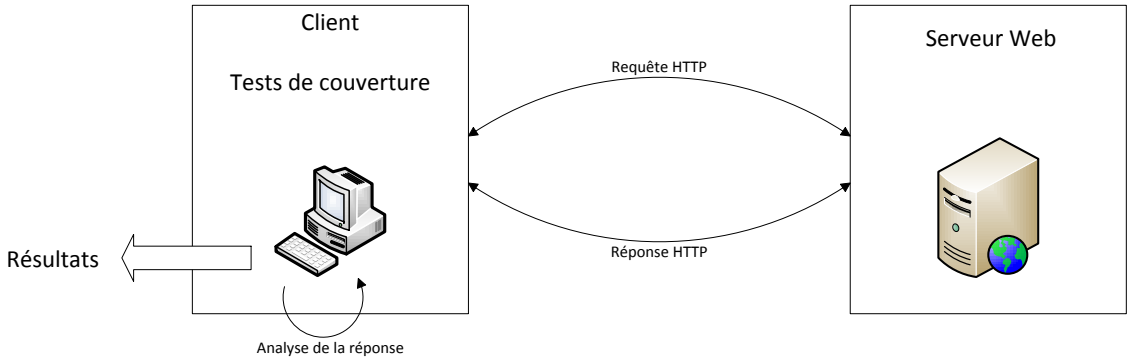

FIGURE 4.15 – Architecture Client/Serveur.

Algorithme et code

L'algorithme 6 présente une version détaillée de l'algorithme de test et de vérification présenté dans la partie 3.5. Pour son implémentation, nous avons utilisé le Framework *AutoIt*, qui permet d'automatiser de manière efficace des interactions utilisateurs dans des environnements Windows, et notamment au sein des navigateurs Web. Dans ce langage, l'implémentation de notre méthode de test est relativement simple. Le code complet (cf. listing 4.8) ne représente qu'une quarantaine de lignes.

Tout d'abord, la liste des utilisateurs (ainsi que leur mot de passe) et les rôles sont stockés dans un fichier texte. Pour chaque couple (utilisateur, rôle), notre outil commence par s'authentifier dans l'application en tant que cet utilisateur. En pratique, cela se traduit par l'accès via la méthode *_IENavigate()* à la page de connexion de l'application à laquelle il est possible de passer en paramètre d'URL le login et mot de passe de l'utilisateur voulant se connecter.

Il accède ensuite à chacun des objets en tant que cet utilisateur via une requête HTTP, et récupère le contenu HTML retourné en réponse à cette requête. Pour déterminer si l'accès est autorisé ou non par notre protection obligatoire, le système analyse le contenu de la réponse. Si celui-ci contient le message d'erreur générique retourné par le moniteur de référence, cela signifie que l'accès a été refusé.

Enfin, une fois que l'ensemble des objets a été parcouru pour l'utilisateur considéré, le système utilise la page de déconnexion de l'application pour déconnecter l'utilisateur en cours et passer au couple (utilisateur/rôle) suivant.

Algorithme 6 : Algorithme détaillé de vérification de la protection.

Entrées : FichierUtilisateursRôles, FichierObjets, modèle spécifique d'application, Politique, MessageErreur
Résultat : A_P,$A_{\overline{P}}$,N

```
Users = LireFichier(FichierUtilisateursRôles)

pour chaque (utilisateur,password,rôle) dans Users faire

    Connexion(utilisateur,password,rôle)

    Objets = LireFichiers(FichierObjets)
    pour chaque objet dans Objets faire
        accès = (utilisateur,rôle,objet)
        source = naviguer_vers(objet)
        N += 1
        si (source ne contient pas MessageErreur) alors
            si (accès est dans Politique) alors
                A_P += 1
            sinon
                A_P̄ += 1
            fin
        fin
    fin
    Déconnexion(utilisateur)
fin
```

4.4 Conclusion

Dans ce chapitre, nous avons présenté une implémentation de notre protection obligatoire dynamique sur des environnements d'applications Web Microsoft (Serveur Web IIS et Framework .Net). Cette implémentation consiste en un module HTTP réalisant une médiation avec le niveau applicatif et agissant comme moniteur de référence. Il est composé de plusieurs gestionnaires d'événements exécutant les différents composants de notre protection. La compilation et le stockage dans le cache serveur de la politique de sécurité sont effectués lors de l'événement *Init*. La décision d'autorisation est prise lors de l'événement *PostAcquireRequestState*. Nous avons présenté les algorithmes de calcul des contextes de sécurité objets, de calcul des règles potentielles d'accès et de prise de décision, et étudié leurs complexités. Nous avons notamment montré que la complexité de la prise de décision est indépendante de la taille de la politique de sécurité à appliquer, ce qui permet la mise en place de politiques de tailles importantes avec un faible impact sur les performances.

Nous avons aussi explicité dans ce chapitre le modèle de workflow de QualNet et montré comment nous pouvions transposer ce modèle de workflows vers notre modèle spécifique de workflows. Comme ce modèle est projeté automatiquement vers notre modèle spécifique d'application, nous calculons bien automatiquement les politiques de sécurité associées aux workflows de QualNet.

Enfin, nous avons présenter la mise en œuvre de notre méthode de test et de vérification de la protection. Celle-ci repose sur une architecture client/serveur qui simule bien tous les accès possibles à l'application Web et fournit des indicateurs de sûreté de la protection. Cette sûreté mesure les taux d'accès légitime et illégitime.

```
$nav = _IECreate()
$users = FileOpen("utilisateurs.txt")
$NbTests = 0
$P = 0
$PBarre = 0
;pour chaque utilisateur faire
While ($ligne = FileReadLine($users)) <> -1

 $user = extraireUser($ligne)
 $password = extrairePass($ligne)

 ;Connexion en tant qu'utilisateur
 _IENavigate($nav,"serveur/intrav61/V5/identification.aspx?nom="
                  & $user & "&password=" $password)

 $objets = FileOpen("objets.txt")
 ;Pour chaque contexte objet faire
 While ($objet = FileReadLine($objets)) <> -1

  _IENavigate($nav,"serveur/intrav61" & $objet)
  $source = _IEDocReadHTML ($nav)
  $NbTests +=1

  If StringInStr($source,"Vous n'avez pas les droits necessaires pour
                          acceder a cette ressource") Then
    $Refus +=1
  Else If (verifierAccesDansPolitique($user,$objet))
    $P +=1
  Else
    $PBarre += 1
  EndIf

  FileClose($objets)
  ;deconnexion de l utilisateur
  _IENavigate($nav,"serveur/intrav61/V5/identification.aspx?ClearSession=1")
 Next
WEnd

FileClose($users)
_IEQuit($nav)
```

Listing 4.8 – Implémentation AutoIt de la méthode de vérification de la protection.

Chapitre 5

Expérimentations

Dans le chapitre précédent, nous avons décrit l'implémentation de notre protection obligatoire dynamique sur des environnements Web Microsoft (IIS/ASP.Net) dans le but de protéger les applications Web, et notamment les workflows, développées par QualNet. Nous avons également présenté l'implémentation de notre méthode de calcul des politiques de sécurité qui se base sur un modèle spécifique d'application et de workflows pour en déduire une politique de sécurité utilisable par notre protection obligatoire. Nous montrons au travers de ce chapitre que ces implémentations sont fonctionnelles, notamment sur les environnements QualNet dans lesquels notre protection obligatoire est maintenant systématiquement intégrée.

Pour cela, nous présentons tout d'abord les politiques de sécurité que notre calculateur de politique de sécurité nous permet de déduire automatiquement sur un cas concret de workflow. Nous détaillons également la méthode utilisée pour calculer la partie de la politique de sécurité concernant certaines parties de l'application pour lesquelles il n'existait pas de spécification nous permettant d'automatiser le calcul des politiques de sécurité. Un *mode apprentissage* nous a aidé à obtenir la politique de sécurité relative aux parties des applications non spécifiées.

Ensuite, nous déroulerons une exécution de notre protection obligatoire en détaillant notamment les contextes de sécurités sujets et objets calculés, les règles potentielles d'accès obtenues et les prises de décision.

De plus, nous effectuerons des mesures de charges et de performances sur notre protection obligatoire dans le but de vérifier expérimentalement les complexités algorithmiques présentées dans le chapitre précédent. Nous verrons que l'implémentation de notre protection obligatoire respecte bien les complexités algorithmes théoriques. Nous mesurerons notamment le temps de compilation d'une politique de sécurité ainsi que la surcharge que notre protection obligatoire impose sur une application de workflow en mesurant le temps d'exécution de chaque composant de notre protection obligatoire. Nous montrerons que cette surcharge est faible vis-à-vis de la taille des environnements protégés et des politiques de sécurité appliquées. Ces éléments justifient l'utilisation en grandeur réelle qui est faite actuellement par les clients de QualNet.

5.1 Calcul des politiques de sécurité

Nous avons présenté dans la partie 4.3 le modèle de workflow de QualNet, et nous avons montré que nous pouvions le transposer vers notre modèle spécifique de workflows pour calculer les politiques de sécurité associées à ces workflows. Nous expliciterons ces politiques dans la sous-partie suivante.

Cependant, les workflows QualNet s'appuient sur un socle applicatif global. Il s'agit d'un portail collaboratif qui permet entre autres fonctionnalités l'accès aux différents workflows, la gestion des utilisateurs et l'administration de l'application. Cette partie est plus complexe à prendre en compte car elle ne présente pas une structure claire faisant l'objet d'une spécification. Pour éviter un travail fastidieux de rétro-ingénierie, nous avons préféré une méthode d'apprentissage à l'approche de modélisation. Cette méthode nous à permis de produire efficacement la politique de sécurité (qui contient plus de 900 règles) du socle applicatif global. Nous détaillerons la méthode d'apprentissage que nous avons utilisée dans un second temps.

5.1.1 Calcul pour les workflows

Calcul pour les étapes d'un workflow

Nous prendrons comme exemple le workflow de demande d'achats de la figure 5.1. Le modèle spécifique associé aux étapes de ce workflow a été présenté dans la figure 3.11.

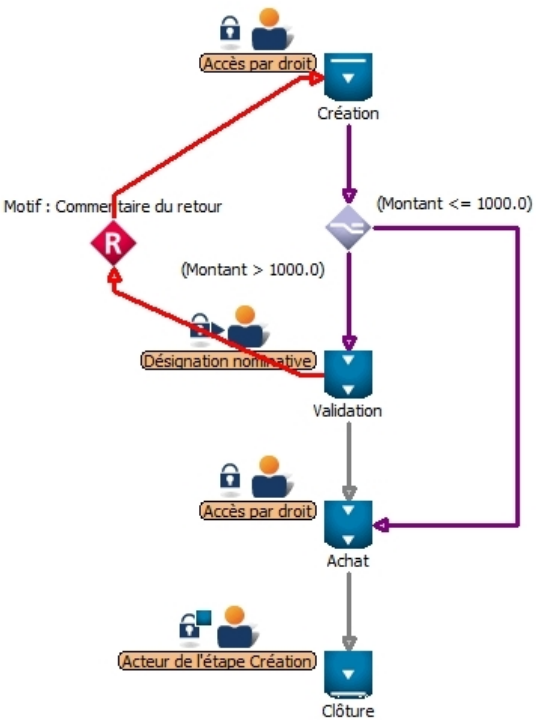

FIGURE 5.1 – Workflow de demande d'achats.

Nous avons vu que le modèle de données présenté dans la partie 4.3 permet au modéliseur de workflows QualNet de générer pour chaque workflow un ensemble de fichiers aspx (cf. figure 4.5). Ce modéliseur génère notamment pour chaque étape d'un workflow un fichier *Saisie* correspondant au formulaire de l'étape et un fichier *Aper* qui est utilisé par

le modéliseur comme aperçu du formulaire lors de sa modélisation. Nous avons également vu que dans le modèle QualNet qu'un rôle particulier est associé à chaque étape (cf. partie 4.3.2). Ce rôle etait noté de manière empirique $M\{IDWORKFLOW\}\_\{IDETAT\}$. L'accès aux fichiers de *Saisie* doit donc être autorisé pour tous les utilisateurs possédant le rôle $M\{IDWORKFLOW\}\_\{IDETAT\}$ associé à l'étape considérée. L'accès aux fichiers d'aperçu est autorisé pour tous les utilisateurs ayant le rôle de modélisation *model*.

Ainsi, pour le workflow de demande d'achats défini précédemment, nous obtenons pour les quatre étapes qui le composent les cinq règles d'autorisation du listing 5.1 :

```
1  allow(*:M4_1, execute, page:/Dynamic/Modeliseur/Modules/M4/et1/Saisie4.aspx)
2  allow(*:model, execute, page:/Dynamic/Modeliseur/Modules/M4/et1/Aper4.aspx)
3
4  allow(*:M4_2, execute, page:/Dynamic/Modeliseur/Modules/M4/et2/Saisie4.aspx)
5  allow(*:model, execute, page:/Dynamic/Modeliseur/Modules/M4/et2/Aper4.aspx)
6
7  allow(*:M4_3, execute, page:/Dynamic/Modeliseur/Modules/M4/et3/Saisie4.aspx)
8  allow(*:model, execute, page:/Dynamic/Modeliseur/Modules/M4/et3/Aper4.aspx)
9
10 allow(*:M4_4, execute, page:/Dynamic/Modeliseur/Modules/M4/et4/Saisie4.aspx)
11 allow(*:model, execute, page:/Dynamic/Modeliseur/Modules/M4/et4/Aper4.aspx)
```

Listing 5.1 – Politique de sécurité du workflow Demande d'achats.

Calcul pour les services utilisés par un workflow

Outre les différentes étapes qui composent un workflow et pour lesquelles nous calculons automatiquement les règles d'autorisations, des *services externes* sont communs à tous les workflow et à leur gestion. Il s'agit des services de *statistiques*, *recherche* et *supervision*. La figure 5.2 détaille à partir du modèle spécifique d'applications que nous avons défini dans la partie 3.4.1 le modèle applicatif relatif à ces services et présente les relations existantes entre le workflow de demande d'achats de notre exemple, les différents services qu'il utilise et les fonctionnalités et ressources qui composent ces services.

Ce modèle spécifique d'application, représentant les interactions entre un workflow et les différents services qu'il utilise, fait apparaitre dix-sept relations *indirectes* tels que nous les avons défini dans la partie 3.4.1. Dans ces relations *indirectes* peuvent intervenir jusqu'à deux paramètres pour un des quatre niveaux de structuration du modèle. Trois de ces relations sont présentées ici :

$$indirecte(Dynamic, \textbf{\textit{Demande d'achats}}, \textbf{\textit{Statistiques}}, Répartition, /Dynamic/Statistiques/*.aspx)$$
$$indirecte(Dynamic, \textbf{\textit{Demande d'achats}}, \textbf{\textit{Recherche}}, Modification, Modification/*.aspx)$$
$$indirecte(Dynamic, \textbf{\textit{Demande d'achats}}, \textbf{\textit{Supervision}}, Répartition\ mensuelle, /Dynamic/Supervision/*.aspx)$$

Le modèle spécifique représentant les services utilisés par un workflow dans les applications QualNet nous permet d'obtenir la politique de sécurité telle que présentée dans le listing 5.1. Nous y retrouvons dix règles d'autorisations correspondantes aux dix ressources utilisées par ces services et leurs fonctionnalités associées. Nous pouvons remarquer que

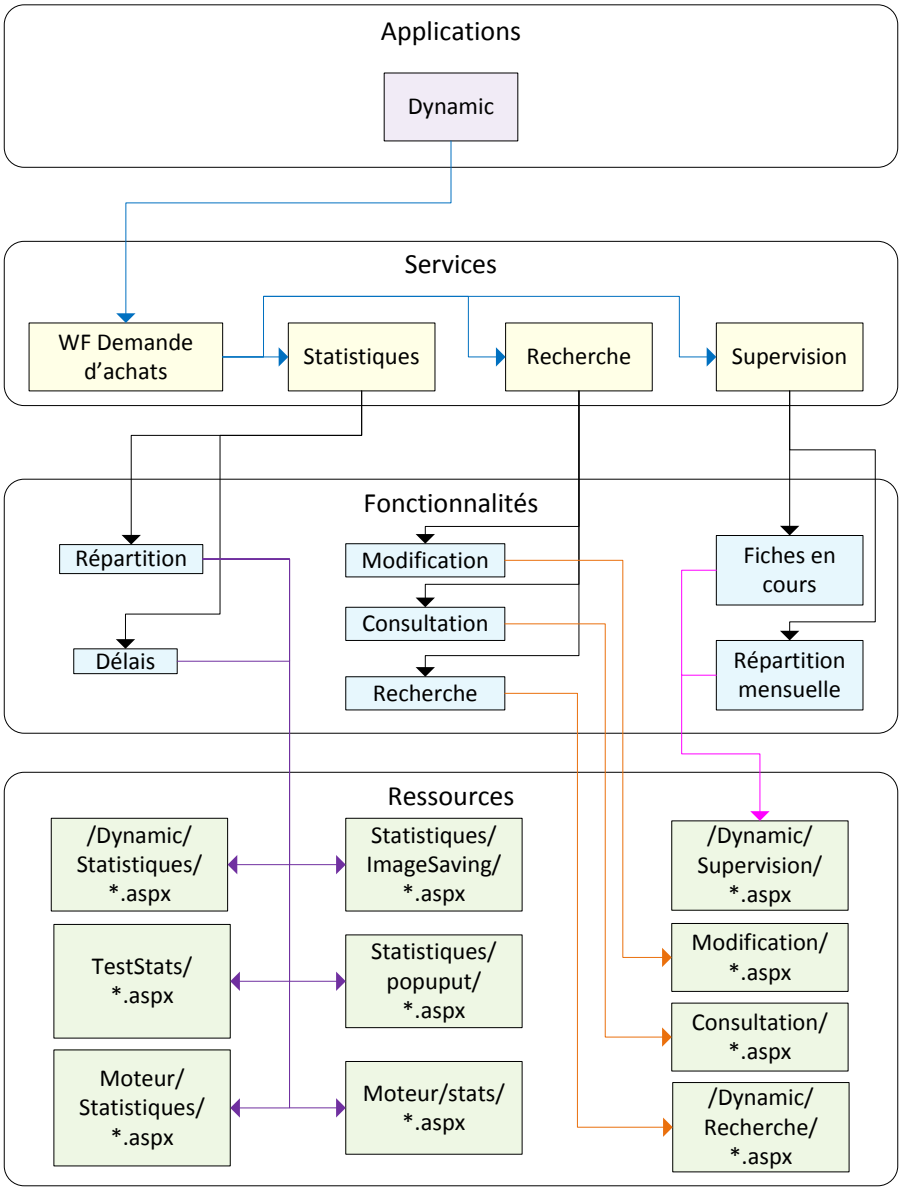

FIGURE 5.2 – Modèle QualNet : Services communs utilisés par un workflow.

les deux premières règles d'autorisations ne sont pas conditionnelles. En effet, dans le modèle QualNet, les fonctionnalités de *Consultation* et *Modification* sont dédiées à chaque workflow. Les huit autres fonctionnalités sont communes à tous les workflows, et donc conditionnées au workflow utilisé.

Par exemple, le rôle *M4_SUPERVISION* ne peut accéder aux ressources utilisées par le service de *Supervision* que pour le workflow *4*. Cette condition s'exprime de la manière suivante :

$$Request(\text{"}idform\text{"}) == 4$$

Pour chaque service utilisé par un workflow, un rôle applicatif est dédié à l'utilisation de ce service par le workflow considéré. Par exemple, le rôle *M4_SUPERVISION* est associé à l'utilisation du service *Supervision* pour le workflow M4. Ainsi, grâce à ces rôles et au modèle spécifique d'application modélisant les relations entre les différents services et les workflows, nous pouvons en déduire les enregistrements de la table *Accès* de notre modèle spécifique d'accès associés à ce workflow, et dont deux exemples sont présentés dans la figure 5.3.

Accès
*
M4_MODIF
Dynamic
Recherche
Modification
/M4/Modification/*.aspx
execute

Accès
*
M4_STATS
Dynamic
Statistiques
Répartition
/Dynamic/Statistiques/*.aspx
execute : Request("idform") == 4

FIGURE 5.3 – Exemple d'accès pour les services externes.

Grâce à ce modèle et à la méthode de calcul des règles d'autorisations présentée dans la partie 4.3.3, nous pouvons en déduire la politique de sécurité suivante :

```
1  allow(*:M4_CONSULT, execute, page:
2          /Dynamic/Modeliseur/Modules/M4/Consultation/*.aspx)
3  allow(*:M4_MODIF, execute, page:
4          /Dynamic/Modeliseur/Modules/M4/Modification/*.aspx)
5  allow(*:M4_CONSULT, execute, page:/Dynamic/Recherche/*.aspx)
6          : Request("idform") == 4
7  allow(*:M4_SUPERVISION, execute, page:/Dynamic/Supervision/*.aspx)
8          : Request("idform") == 4
9  allow(*:M4_STATS, execute, page:/Dynamic/Statistiques/*.aspx)
10         : Request("idform") == 4
11 allow(*:M4_STATS, execute, page:/Dynamic/Statistiques/TestStats/*.aspx)
12         : Request("idform") == 4
13 allow(*:M4_STATS, execute, page:/V5/Dynamic/Moteur/Statistiques/*.aspx)
14         : Request("idform") == 4
15 allow(*:M4_STATS, execute, page:
16         /V5/Dynamic/Moteur/Statistiques/ImageSaving/*.aspx)
17         : Request("idform") == 4
18 allow(*:M4_STATS, execute, page:/V5/Dynamic/Moteur/Statistiques/popuput/*.aspx)
19         : Request("idform") == 4
20 allow(*:M4_STATS, execute, page:/V5/Dynamic/Moteur/stats/*.aspx)
21         : Request("idform") == 4
```

Listing 5.2 – Politique calculée pour les services utilisés par un Workflow.

5.1.2 Politique de sécurité pour le socle applicatif

Pour obtenir la politique de sécurité associée au socle applicatif sur lequel s'appuie le système de workflow de Qualnet, nous proposons un **mode d'apprentissage** de la politique de protection.

Mode apprentissage

L'idée générale était d'utiliser l'application avec chacun des rôles applicatifs et de tracer tous les accès effectués. Cette méthode est illustrée dans l'algorithme 7. Pour cela, notre protection obligatoire était configurée de manière permissive, c'est-à-dire que le calcul d'autorisation ou de refus est effectué, mais les accès interdits ne sont pas bloqués. Cependant, les autorisations d'accès et les refus sont stockés dans des fichiers de traces. Pour obtenir une politique la plus exacte possible, nous avons collaboré avec les personnes chargées des formations aux logiciels QualNet, car elles maitrisent parfaitement tous les rouages des applications et notamment les ressources autorisées pour chacun des rôles applicatifs.

Grâce à l'ensemble des accès effectués pour un rôle donné, nous pouvons construire la politique de sécurité associée en autorisant le rôle applicatif considéré à accéder à l'ensemble des ressources qui ont été utilisées durant la phase d'apprentissage.

Nous avons commencé par recenser l'ensemble des ressources utilisées par le socle applicatif dans le but de vérifier la couverture de notre méthode. Après avoir parcouru l'ensemble des rôles applicatifs, nous pouvons vérifier si l'ensemble des ressources de l'application a été parcouru lors de la phase d'apprentissage. Le nombre de rôle applicatif étant fixé, et les rôles connus, en appliquant cette méthode pour chaque rôle applicatif, nous obtenons bien la politique de sécurité du socle applicatif.

Pour obtenir une couverture complète, plusieurs utilisations du mode d'apprentissage ont du être effectuées. Grâce à cette méthode, nous avons obtenu une couverture des ressources de 95%. Les 5% restants étaient soit des ressources qui furent oubliées lors de l'utilisation de l'application en mode apprentissage, soit des ressources toujours présentes physiquement mais qui ne sont plus utilisées par l'application. Pour ces ressources (environ quarante) nous avons dû rechercher manuellement quelles sont les fonctionnalités qui les utilisent, et en déduire les rôles applicatifs associés. L'utilisation du mode apprentissage nous a grandement facilité la création de cette politique de sécurité en contournant le problème de l'absence de spécification commune et évité un travail fastidieux de rétro-ingénierie.

Algorithme 7 : Calcul des politiques de sécurité en mode apprentissage.

```
Entrées : Ressources, modèle spécifique, SQL, Rôles
Résultat : Politique
Accès = retrouverAccès(modèle spécifique,SQL)
pour chaque rôle dans Rôles faire
    pour chaque ressource dans Ressources faire
        Traces += Tracer_accès(ressource, rôle)
    fin
    Politique += générer_politique(Traces)
fin
retourner Politique
```

Correction de la politique obtenue

Cependant, cette méthode ne garantit pas l'exactitude de la politique de sécurité obtenue car elle ne repose pas sur un modèle formel d'application mais sur la connaissance de l'application. Après avoir obtenu une politique de sécurité complète pour le socle applicatif, il est donc nécessaire de vérifier la cohérence de la politique de sécurité ainsi calculée. Pour cela, nous avons ensuite appliqué cette politique de sécurité sur l'environnement de test en configurant notre protection obligatoire en mode *strict*, c'est-à-dire que les accès refusés sont effectivement bloqués par la protection, et nous avons demandé aux utilisateurs d'utiliser l'application tout en traçant les accès refusés.

Nous avons ensuite pu corriger la politique en étudiant au cas par cas les accès ayant été refusés (environ 25 règles ont été modifiées, soit 3% d'erreur sur la politique calculée).

Optimisation de la politique obtenue

Enfin, nous avons cherché à réduire la taille de la politique en utilisant les notations factorisées présentées dans la partie 4.2.2. Dans la politique obtenue par l'utilisation du mode apprentissage n'apparait que des règles d'autorisation sans factorisations des contextes de sécurité objets. Cependant, pour chaque rôle applicatif, il est possible de réduire la taille de la politique. Nous proposons une méthode qui permet de remplacer dans la politique de sécurité calculée les règles d'autorisations qui peuvent être factorisées Si toutes les règles équivalentes à une règle factorisée sont présentes dans la politique de sécurité, alors nous pouvons remplacer cet ensemble de règles par la règle factorisée associée.

Cette méthode est exprimée dans l'algorithme 8.

Sur l'exemple suivant, nous montrons une utilisation sur un répertoire */admin* qui ne contient seulement que trois pages : *menu.aspx*, *gestion_serveur.aspx* et *config_workflows.aspx*. Étant donné qu'il existe une règle autorisant l'accès pour les trois ressources au rôle *administrateur*, ces trois règles peuvent être remplacées par la règle factorisée autorisant l'accès a l'ensemble des pages du répertoire */admin*.

$$\left\{ \begin{array}{l} allow(* : administrateur, execute, page : /admin/menu.aspx) \\ allow(* : administrateur, execute, page : /admin/gestion\_serveur.aspx) \\ allow(* : administrateur, execute, page : /admin/config\_workflows.aspx) \end{array} \right\}$$

$$\Longleftrightarrow$$

$$allow(* : administrateur, execute, page : /admin/*.aspx)$$

Algorithme 8 : Optimisation des politiques de sécurité en mode apprentissage.

```
Entrées : Ressources, Politique
Résultat : Politique
pour chaque règle dans Politique faire
    ressource_règle = ExtraireRessource(règle)
    rôle = ExtraireRôle(règle)
    P = Parent(ressource_règle)
    ok_Remplace = Vrai
    pour chaque ressource dans P faire
        si accès(ressource,rôle) n'est pas dans Politique alors
            ok_Remplace = Faux
        fin
        si ok_Remplace alors
            nouvelle_règle = accès(P, rôle)
            remplacer(règle,nouvelle_règle)
        fin
    fin
fin
Supprimer_Doublons(Politique)

retourner Politique
```

5.2 Exécution du contrôleur d'accès

Nous allons maintenant développer un cas concret d'exécution de notre protection obligatoire en détaillant les rôles applicatifs associés à l'utilisateur de notre exemple, l'ensemble des contextes de sécurité objets relatifs à la ressource accédée, les règles potentielles obtenues grâce à ces contextes de sécurité objets et le mécanisme de prise de décision qui compare ces règles potentielles d'accès à la politique de sécurité. L'accès que nous considérons ici est le suivant :

L'utilisateur *Bob* tente d'accéder à la seconde étape du workflow *M4*.

5.2.1 Rôles applicatifs

Les rôles que possède l'utilisateur *Bob* au sein de l'application considérée sont présentés dans le listing 5.3. Ces rôles sont retournés par l'adaptateur applicatif lorsque celui-ci est interrogé par le moniteur de référence.

```
1 admin dyn
2 Gestion utilisateurs
3 M4_1
4 M4_2
5 M4_STATS
```

Listing 5.3 – Liste des rôles de l'utilisateur.

5.2.2 Contextes de sécurité objets

En accord avec les modèles de données et de fichiers QualNet que nous avons présentés dans la partie 4.3.1, la ressource associée à la seconde étape du workflow M4 est donc :

page :/Dynamic/modeliseur/modules/M4/et2/Saisie4.aspx

Pour le calcul des contextes de sécurité objet associés à cette ressource, nous considérons une profondeur de calcul $p = 3$. Ainsi, grâce à l'algorithme de calcul des contextes de sécurité objets, nous obtenons les contextes de sécurité objets présentés dans le listing 5.4.

```
1 page:/Dynamic/modeliseur/modules/M4/et2/Saisie4.aspx
2 page:/Dynamic/modeliseur/modules/M4/et2/*.aspx
3 page:/Dynamic/modeliseur/modules/M4/et2/*
4 page:/Dynamic/modeliseur/modules/M4/*
```

Listing 5.4 – Liste des contextes de sécurité objets.

Nous avons vu dans la partie 4.2.6 que le nombre de contextes de sécurité objets obtenus pour une profondeur de calcul p supérieure à 1 est égale à $p+1$. Nous retrouvons bien ici, pour une profondeur de calcul $p = 3$, quatre contextes de sécurité objets.

5.2.3 Règles potentielles d'accès

Grâce à l'ensemble des rôles applicatifs et des contextes de sécurité objets précédents, nous pouvons calculer les règles potentielles d'accès associées à la requête que nous traitons. Celles-ci sont présentées dans le listing 5.5. L'ordre dans lequel celles-ci sont présentées correspond à celui de génération par l'algorithme de calcul.

```
allow(*:admin dyn,execute,page:
        /Dynamic/modeliseur/modules/M4/et2/Saisie4.aspx)
allow(Bob:admin dyn,execute,page:
        /Dynamic/modeliseur/modules/M4/et2/Saisie4.aspx)
allow(*:Gestion utilisateurs,execute,page:
        /Dynamic/modeliseur/modules/M4/et2/Saisie4.aspx)
allow(Bob:Gestion utilisateurs,execute,page:
        /Dynamic/modeliseur/modules/M4/et2/Saisie4.aspx)
allow(*:M4_1,execute,page:/Dynamic/modeliseur/modules/M4/et2/Saisie4.aspx)
allow(Bob:M4_1,execute,page:/Dynamic/modeliseur/modules/M4/et2/Saisie4.aspx)
allow(*:M4_2,execute,page:/Dynamic/modeliseur/modules/M4/et2/Saisie4.aspx)
allow(Bob:M4_2,execute,page:/Dynamic/modeliseur/modules/M4/et2/Saisie4.aspx)
allow(*:M4_STATS,execute,page:
        /Dynamic/modeliseur/modules/M4/et2/Saisie4.aspx)
allow(Bob:M4_STATS,execute,page:
        /Dynamic/modeliseur/modules/M4/et2/Saisie4.aspx)
allow(Bob:*,execute,page:/Dynamic/modeliseur/modules/M4/et2/Saisie4.aspx)
allow(*:*,execute,page:/Dynamic/modeliseur/modules/M4/et2/Saisie4.aspx)
allow(*:admin dyn,execute,page:/Dynamic/modeliseur/modules/M4/et2/*.aspx)
allow(Bob:admin dyn,execute,page:/Dynamic/modeliseur/modules/M4/et2/*.aspx)
allow(*:Gestion utilisateurs,execute,page:
        /Dynamic/modeliseur/modules/M4/et2/*.aspx)
allow(Bob:Gestion utilisateurs,execute,page:
        /Dynamic/modeliseur/modules/M4/et2/*.aspx)
allow(*:M4_1,execute,page:/Dynamic/modeliseur/modules/M4/et2/*.aspx)
allow(Bob:M4_1,execute,page:/Dynamic/modeliseur/modules/M4/et2/*.aspx)
allow(*:M4_2,execute,page:/Dynamic/modeliseur/modules/M4/et2/*.aspx)
allow(Bob:M4_2,execute,page:/Dynamic/modeliseur/modules/M4/et2/*.aspx)
allow(*:M4_STATS,execute,page:/Dynamic/modeliseur/modules/M4/et2/*.aspx)
allow(Bob:M4_STATS,execute,page:/Dynamic/modeliseur/modules/M4/et2/*.aspx)
allow(Bob:*,execute,page:/Dynamic/modeliseur/modules/M4/et2/*.aspx)
allow(*:*,execute,page:/Dynamic/modeliseur/modules/M4/et2/*.aspx)
allow(*:admin dyn,execute,page:/Dynamic/modeliseur/modules/M4/et2/*)
allow(Bob:admin dyn,execute,page:/Dynamic/modeliseur/modules/M4/et2/*)
allow(*:Gestion utilisateurs,execute,page:
        /Dynamic/modeliseur/modules/M4/et2/*)
allow(Bob:Gestion utilisateurs,execute,page:
        /Dynamic/modeliseur/modules/M4/et2/*)
allow(*:M4_1,execute,page:/Dynamic/modeliseur/modules/M4/et2/*)
allow(Bob:M4_1,execute,page:/Dynamic/modeliseur/modules/M4/et2/*)
allow(*:M4_2,execute,page:/Dynamic/modeliseur/modules/M4/et2/*)
allow(Bob:M4_2,execute,page:/Dynamic/modeliseur/modules/M4/et2/*)
allow(*:M4_STATS,execute,page:/Dynamic/modeliseur/modules/M4/et2/*)
allow(Bob:M4_STATS,execute,page:/Dynamic/modeliseur/modules/M4/et2/*)
allow(Bob:*,execute,page:/Dynamic/modeliseur/modules/M4/et2/*)
allow(*:*,execute,page:/Dynamic/modeliseur/modules/M4/et2/*)
allow(*:admin dyn,execute,page:/Dynamic/modeliseur/modules/M4/*)
allow(Bob:admin dyn,execute,page:/Dynamic/modeliseur/modules/M4/*)
allow(*:Gestion utilisateurs,execute,page:/Dynamic/modeliseur/modules/M4/*)
allow(Bob:Gestion utilisateurs,execute,page:/Dynamic/modeliseur/modules/M4/*)
allow(*:M4_1,execute,page:/Dynamic/modeliseur/modules/M4/*)
allow(Bob:M4_1,execute,page:/Dynamic/modeliseur/modules/M4/*)
allow(*:M4_2,execute,page:/Dynamic/modeliseur/modules/M4/*)
allow(Bob:M4_2,execute,page:/Dynamic/modeliseur/modules/M4/*)
allow(*:M4_STATS,execute,page:/Dynamic/modeliseur/modules/M4/*)
allow(Bob:M4_STATS,execute,page:/Dynamic/modeliseur/modules/M4/*)
allow(Bob:*,execute,page:/Dynamic/modeliseur/modules/M4/*)
allow(*:*,execute,page:/Dynamic/modeliseur/modules/M4/*)
```

Listing 5.5 – Liste des règles potentielles d'accès.

Dans la partie 4.2.6, nous avons vu que le nombre de règle potentielles d'accès calculées est égal à $(p+1)*(2r+2)$, avec p la profondeur de calcul ($p=3$) et r le nombre de rôles applicatifs retournés par l'adaptateur applicatif ($r=5$). Nous obtenons bien ici un nombre de règles potentielles d'accès égal à $4*(2*5+2)=4*12=48$.

5.2.4 Prise de décision

Les règles potentielles d'accès étant calculées, le moteur de décision peut maintenant comparer ces règles potentielles à la politique de sécurité pour décider d'autoriser ou de refuser l'accès associé à la requête traitée. Pour cela, chaque règle potentielle d'accès est recherchée dans la politique de sécurité. La première règle présente dans la politique de sécurité autorise l'accès. Si des conditions sont exprimées pour cette règle, l'accès n'est autorisé uniquement si ces conditions sont respectées.

La politique complète associée au workflow M4 est présentée dans les listings 5.1 et 5.2. Pour illustrer notre exemple, une sous-partie de cette politique est exposée dans le listing 5.6.

```
1  allow(*:M4_1, execute, page:/Dynamic/Modeliseur/Modules/M4/et1/Saisie4.aspx)
2  allow(*:M4_2, execute, page:/Dynamic/Modeliseur/Modules/M4/et2/Saisie4.aspx)
3  allow(*:M4_3, execute, page:/Dynamic/Modeliseur/Modules/M4/et3/Saisie4.aspx)
4  allow(*:M4_4, execute, page:/Dynamic/Modeliseur/Modules/M4/et4/Saisie4.aspx)
5  allow(*:M4_CONSULT, execute, page:
6          /Dynamic/Modeliseur/Modules/M4/Consultation/*.aspx)
7  allow(*:M4_MODIF, execute, page:
8          /Dynamic/Modeliseur/Modules/M4/Modification/*.aspx)
9  allow(*:M4_CONSULT, execute, page:/Dynamic/Recherche/*.aspx)
10         : Request("idform") == 4
```

Listing 5.6 – Extrait de la politique de sécurité du workflow M4.

Le moteur de décision parcourt les règles potentielles d'accès présentées dans le listing 5.5 de manière itérative. Les six premières règles potentielles ne sont pas présentes dans la politique, elle ne permettent donc pas au moteur de décision d'autoriser l'accès. La septième règle potentielle (ligne 11 du listing 5.5) est présente dans la politique (ligne 2 du listing 5.6). Le moteur de décision a donc trouvé une correspondance entre les règles potentielles calculées et la politique de sécurité. Dans la politique, cette règle ne comporte pas de conditions. L'accès est donc directement autorisé et les règles potentielles suivantes ne sont pas recherchées au sein de la politique.

5.3 Vérification des complexités

Dans cette partie, nous allons vérifier expérimentalement les complexités des différents algorithmes que nous avons proposés dans le chapitre 4. Pour cela, nous avons mesuré le temps d'exécution de ces algorithmes en fonction de différents paramètres. Pour effectuer ces mesures, nous avons utilisé des objets *Stopwatch* fournis par le framework .Net. Ces objets permettent de mesurer précisément le temps écoulé avec une précision de l'ordre de la microseconde.

```
1  using System.Diagnostics;
2
3  //instanciation d'un objet stopwatch
4  Stopwatch chrono = new Stopwatch();
5  chrono.Start();//Demarrage du chronometre
6
7  Decisionner.takeDesicion(); //le code a mesurer
8  ...
9
10 chrono.Stop();  //arret du chronometre
```

Listing 5.7 – Exemple - Mesure du temps de prise de décision.

5.3.1 Calcul des contextes de sécurité objets

Dans le but de vérifier la complexité de notre méthode de calcul des contextes de sécurité objets associés à une ressource, nous avons mesuré le temps d'exécution de l'algorithme de calcul en fonction de la profondeur de calcul configurée (cf. partie 4.2.6). Pour différentes ressources, nous mesurons le temps moyen d'exécution du calcul des contextes de sécurités objets sur cinquante requêtes à la même ressource. Les conditions de mesure sont explicitées dans la table 5.1.

Utilisateur	fixé
Nombre de rôles	fixé
Profondeur de calcul	**variable**
Taille de la politique	fixée
Ressource accédée	**variable**
Nombre d'itérations	50

TABLE 5.1 – Conditions expérimentales pour la vérification du calcul des contextes de sécurité objets.

Nous présentons les résultats de ces mesures pour trois ressources distinctes dans la figure 5.4. Nous pouvons remarquer que, pour une profondeur de calcul faible (ici inférieure à 4), le temps de calcul moyen est le même pour les trois ressources. Comme nous l'avions prévu dans notre calcul de complexité pour l'algorithme de calcul des contextes de sécurité objets, pour une profondeur de calcul donnée, le temps de calcul est constant et indépendant de la ressource.

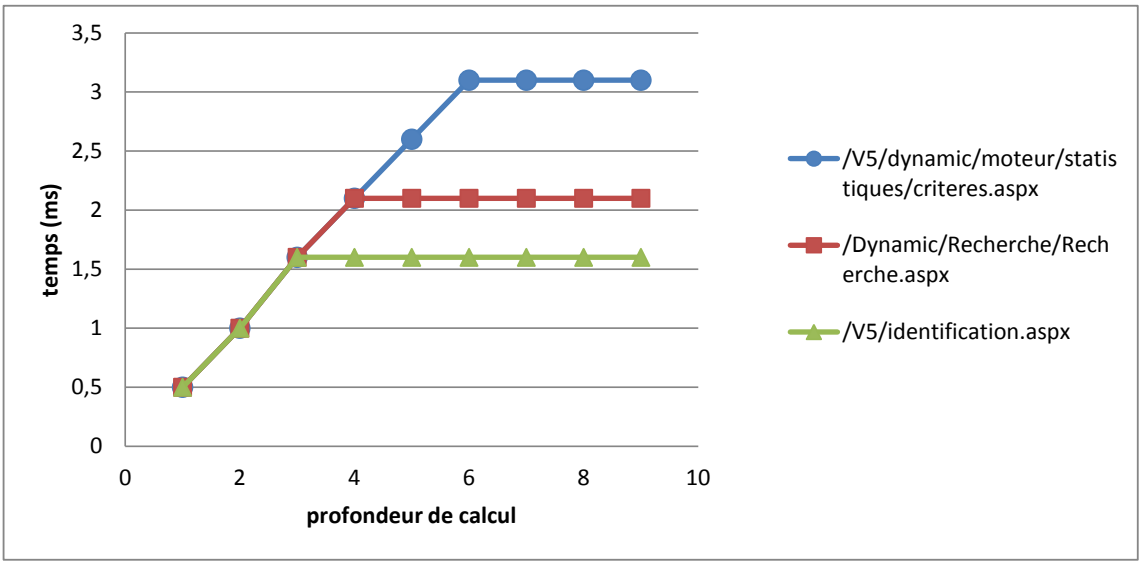

FIGURE 5.4 – Temps de calcul des contextes de sécurité objets en fonction de la profondeur.

De plus, nous remarquons que pour une ressource donnée le temps de calcul des contextes de sécurité objets associés augmente linéairement avec la profondeur de calcul configurée. Cependant, nous voyons que celui-ci devient constant à partir d'une certaine profondeur de calcul. En effet, notre algorithme de calcul possède deux conditions d'arrêt : l'algorithme s'arrête soit lorsque la profondeur de calcul est atteinte, soit lorsque l'algorithme est remonté jusqu'au niveau racine de la ressource. Dans le cas d'une grande profondeur de calcul, c'est cette seconde condition qui prédomine. Par exemple, pour la ressource */V5/dynamic/moteur/statistiques/criteres.aspx*, une profondeur de calcul de 6 permet de remonter à la racine */*. Donc pour une profondeur de calcul supérieure, le calcul s'arrête lorsque la racine est atteinte.

5.3.2 Calcul des règles potentielles d'accès

Pour vérifier la complexité de notre algorithme de calcul des règles potentielles d'accès, nous avons mesuré le temps moyen de génération des règles potentielles d'accès pour une ressource donnée, en fonction du nombre de rôles applicatifs de l'utilisateur accédant à cette ressource. Pour cela, nous avons configuré quinze utilisateurs différents, chacun possédant un rôle de plus que le précédent. Ainsi, le premier utilisateur ne possédait qu'un seule rôle, le second utilisateur en possédait deux, et ainsi de suite.

Chacun de ces utilisateurs ainsi configurés accède à une ressource fixée, et nous mesurons le temps de calcul des règles potentielles d'accès pour deux valeurs de profondeur de calcul p. Les conditions de ces mesures sont explicitées dans la table 5.2.

Utilisateur	**variable**
Nombre de rôles	**variable**
Profondeur de calcul	**variable**
Taille de la politique	fixée
Ressource accédée	fixée
Nombre d'itérations	50

TABLE 5.2 – Conditions expérimentales pour le calcul des règles potentielles d'accès.

Nous présentons les résultats de ces mesures pour une profondeur de calcul $p = 6$ dans la table 5.3. La figure 5.5 représente sous forme graphique ces résultats pour les profondeurs de calcul $p = 3$ et $p = 6$. La variation entre les valeurs minimales et maximales mesurées est ici très faible (inférieure à 5 %) et calculée de la manière suivante :

$$Variation = \frac{V_{max} - V_{min}}{V_{moy}}$$

Rôles	**Valeur min**	**Moyenne**	**Valeur max**	**Variation**
1	0.60	**0.61**	0.63	4.92 %
2	0.96	**0.99**	1.01	5.05 %
3	1.30	**1.35**	1.36	4.44 %
4	1.66	**1.72**	1.72	3.49 %
5	2.04	**2.09**	2.08	1.91 %
6	2.41	**2.43**	2.46	2.07 %
7	2.77	**2.80**	2.81	1.43 %
8	3.13	**3.18**	3.20	2.20 %
9	3.49	**3.53**	3.55	1.70 %
10	3.84	**3.85**	3.90	1.56 %
11	4.22	**4.25**	4.28	1.41 %
12	4.57	**4.61**	4.64	1.52 %
13	4.95	**4.99**	5.03	1.60 %
14	5.29	**5.34**	5.38	1.69 %
15	5.65	**5.70**	5.74	1.58 %

TABLE 5.3 – Résultats des mesures pour $p = 6$.

Ces résultats nous montrent que le temps de calcul des règles potentielles d'accès est linéairement dépendant du nombre de rôles applicatifs retournés par l'adaptateur applicatif. Le temps de calcul apparait également dépendant de la profondeur de calcul configurée au sein de l'application. Dans la partie 4.2.6, nous proposons une complexité de notre algorithme de calcul des règles potentielles d'accès de O(c(2r+2)). Ces résultats nous permettent de confirmer cette complexité, le temps de calcul des règles potentielles d'accès étant bien linéairement dépendant du nombre de rôles et de la profondeur de calcul.

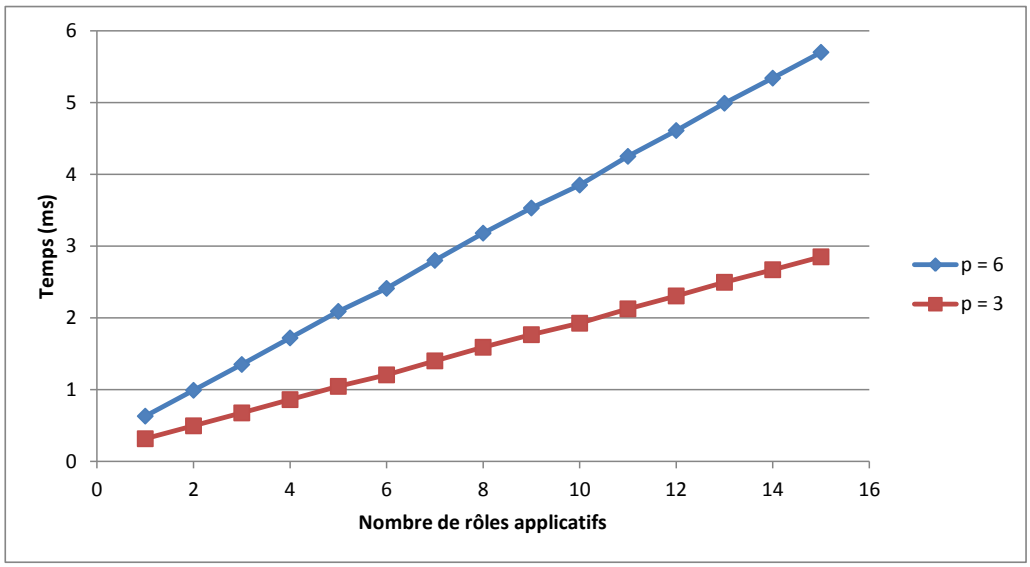

FIGURE 5.5 – Temps de calcul des règles potentielles d'accès en fonction du nombre de rôles applicatifs.

Comme nous l'avons proposé dans la partie 4.2.7, ces résultats montrent qu'il est possible de réduire le temps de calcul des règles potentielles d'accès et donc le temps d'autorisation de deux manières différentes :

- Diminuer la profondeur de calcul (au détriment de l'utilisation de règles d'autorisation factorisées).
- Réduire le nombre de rôles en utilisant un adaptateur applicatif intelligent limitant le nombre de rôles retournés.

5.3.3 Prise de décision

Pour vérifier la complexité de notre algorithme de prise de décision, nous avons mesuré le temps moyen de décision d'autorisation pour une ressource donnée, en fonction du nombre de rôles applicatifs de l'utilisateur accédant à cette ressource. Pour cela, nous avons utilisé une méthode identique à celle que nous avons utilisé pour vérifier la complexité de l'algorithme de calcul des règles potentielles d'accès. Les conditions de ces mesures sont explicitées dans la table 5.4.

Utilisateur	**variable**
Nombre de rôles	**variable**
Profondeur de calcul	**variable**
Taille de la politique	fixée
Ressource accédée	fixée
Nombre d'itérations	50

TABLE 5.4 – Conditions expérimentales pour la vérification de la prise de décision.

Les résultats de ces mesures sont exposés dans le graphique de la figure 5.6.

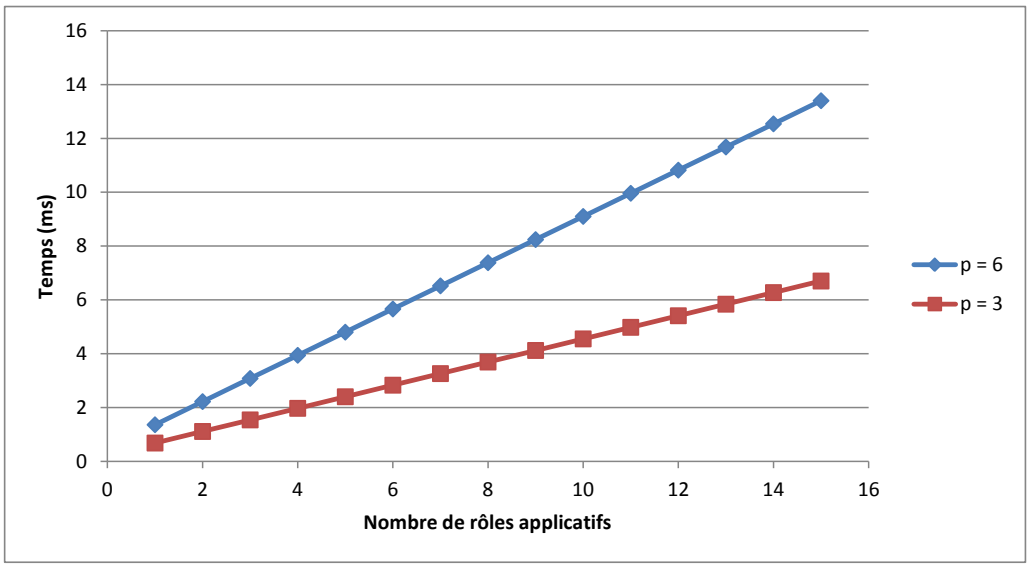

FIGURE 5.6 – Temps de prise de décision en fonction du nombre de rôles applicatifs.

Les résultats obtenus montrent que le temps de prise de décision est, comme le temps de calcul des règles potentielles d'accès, linéairement dépendant du nombre de rôles retournés par l'adaptateur applicatif, mais également de la profondeur de calcul.

Ces résultats sont cohérents vis-à-vis des complexités que nous avons obtenues dans les parties 4.2.6 et 4.2.7, montrant que les algorithmes de calcul des règles potentielles d'accès et de prise de décision possèdent les même complexités. En effet, nous avons montré que le temps de prise de décision dépend uniquement du nombre de règles potentielles d'accès. Nous retrouvons donc par l'expérience la même complexité pour les deux algorithmes, c'est-à-dire une complexité $O(c(2r+2))$.

Taille de la politique de sécurité

Nous avons également mesuré le temps d'exécution du moteur de décision pour une ressource ciblée en fonction de la taille de la politique appliquée mais également de la position de la règle autorisant cet accès dans la politique. Le but ici est de vérifier l'indépendance du moteur de décision vis-à-vis de la taille de la politique de sécurité. En effet, une politique de sécurité est constituée de règles d'autorisations consécutives. Nous avons vu que le moteur de décision recherche de manière itérative dans la politique de sécurité chaque règle potentielle calculée en rapport avec la ressource accédée.

Nous voulons vérifier ici que le parcours de la politique de sécurité est indépendant de la taille de la politique. Par exemple, pour une politique P_1 contenant mille règles, et une politique P_2 contenant trois mille règles, si la règle autorisant l'accès est la centième, le temps de décision sera le même pour les deux politiques. Cependant, nous voulons aussi vérifier que le temps de décision reste le même, que la règle d'autorisation soit placée en première ou en dernière position dans la politique.

Pour cela, nous avons généré différentes politiques de sécurité de tailles différentes (de 100 à 12500 règles d'autorisations), dans lesquelles nous avons à chaque fois placé en dernière position la règle autorisant l'accès que nous allons effectuer. Ainsi, nous pouvons vérifier que la taille de la politique et la position des règles dans la politique de sécurité n'a pas d'influence sur la prise de décision d'autorisation (cf. figure 5.7).

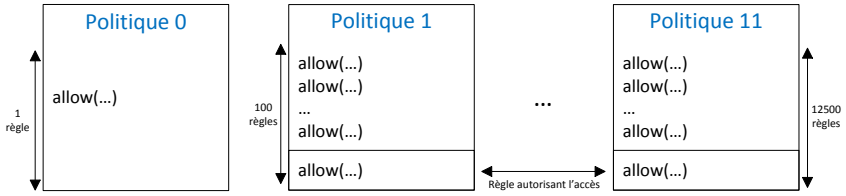

FIGURE 5.7 – Politiques de tailles différentes.

Les conditions de mesures sont explicitées dans la table 5.5. Les résultats de ces mesures sont présentés dans la figure 5.8.

Utilisateur	fixé
Nombre de rôles	fixé
Profondeur de calcul	fixé
Taille de la politique	**variable**
Ressource accédée	fixé
Nombre d'itérations	10

TABLE 5.5 – Conditions expérimentales pour la vérification de la prise de décision.

Ces résultats nous montrent que le temps de prise de décision reste constant lorsque la taille de la politique de sécurité augmente. En effet, pour les onze politiques de tailles différentes utilisées, le temps d'autorisation se situe entre 13.3 et 13.4 millisecondes. La variation du temps de prise de décision est ici inférieure à un pour cent :

$$Variation = \frac{V_{max} - V_{min}}{V_{moy}} = \frac{13.40 - 13.31}{13.35} = 0.7\ \%$$

De plus, celui-ci est également indépendant de la position de la règle autorisant l'accès au sein de la politique de sécurité. En effet, le temps de prise de décision est le même pour la politique 0 correspondant à la règle d'autorisation seule et donc en première position et pour les autres politiques de sécurité dans lesquelles la règle d'autorisation se trouve à la fin.

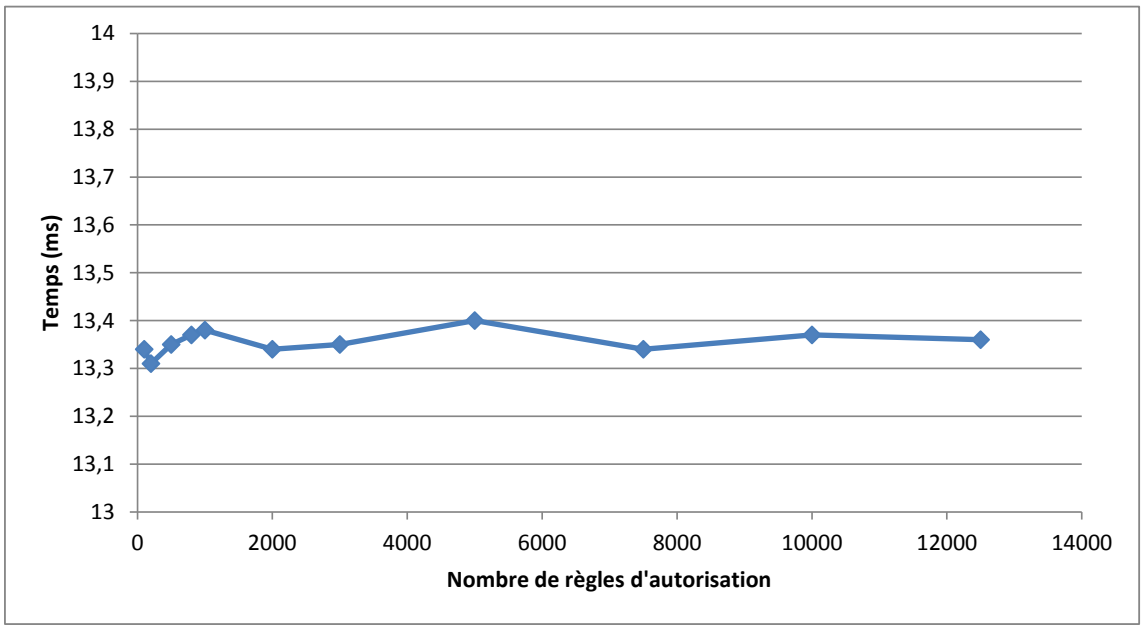

FIGURE 5.8 – Temps de prise de décision en fonction de la taille de la politique de sécurité.

5.4 Performances

Afin d'évaluer les performances de notre protection obligatoire, nous avons mesuré le temps de compilation d'une politique de sécurité en fonction du nombre de règles de la politique de sécurité. Nous avons également mesuré le temps d'exécution des différents composants de notre protection obligatoire sur deux environnements QualNet pour évaluer la surcharge imposée par notre protection et mettre en évidence le composant le plus consommateur de temps.

5.4.1 Compilation des politiques de sécurité

Nous avons mesuré le temps de compilation pour différentes politiques de sécurité dont le nombre de règles varie de 100 à 12500 règles. Les résultats de ces mesures sont présentés dans la table 5.6 et le graphique de la figure 5.9.

Nombre de règles	**temps de compilation (ms)**
100	1.3
200	3.0
500	10.1
800	17.3
1000	22.0
2000	38.3
3000	58.7
5000	104.9
7500	150.0
10000	199.3
12500	250.8

TABLE 5.6 – Temps de compilation d'une politique.

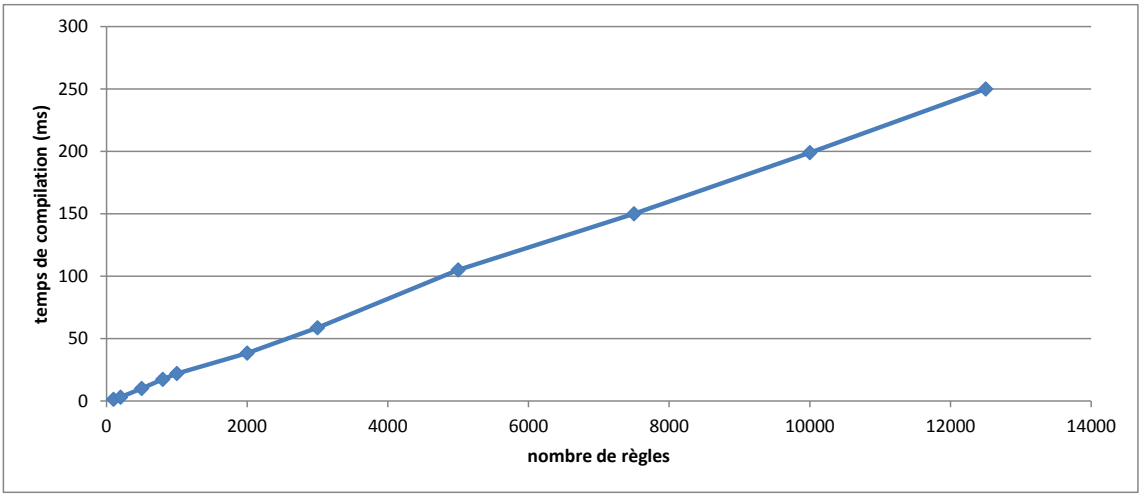

FIGURE 5.9 – Temps de compilation d'une politique en fonction du nombre de règles.

Ces mesures nous permettent d'affirmer que le temps de compilation d'une politique est linéairement dépendant du nombre de règles d'autorisations qu'elle contient. Pour un

environnement QualNet, la politique de sécurité complète pour une utilisation en vrai grandeur comporte généralement moins de 3000 règles. Ainsi, la surcharge imposée par la compilation de la politique de sécurité sur le chargement de l'application QualNet par le serveur d'application Web sera inférieure à 58 millisecondes.

5.4.2 Temps d'exécution des différents composants.

Nous avons effectué des mesures de temps d'exécution des différents composants de notre protection obligatoire sur deux environnements distincts. Le premier environnement est une copie de l'application utilisée en interne par QualNet pour la gestion de ses processus métiers. Le second environnement est un environnement à grande échelle utilisé par un des clients de QualNet. Ainsi, nous pouvons évaluer les performances pour une utilisation à grande échelle par un client. Les spécifications de ces environnements sont présentées dans la table 5.7.

	Environnement 1	**Environnement 2**
Utilisateurs	25	548
Rôles	176	455
Workflows	41	110
Pages Web	164	440
Nombre de règles	1644	2886

TABLE 5.7 – Spécifications des environnements de tests.

Le but de cette expérimentation est de mesurer le temps d'exécution du moniteur de référence et d'évaluer sa surcharge sur le temps de traitement d'une requête par le serveur Web. Nous mesurons le temps d'exécution de chaque composant du moniteur de référence, dans le but de mettre en évidence le composant qui consomme le plus de temps d'exécution du moniteur de référence. Dans ce but, nous comparons le temps d'exécution moyen d'un grand nombre de pages Web sur les deux environnements, avec notre protection obligatoire activée puis désactivée. Nous mesurons ensuite le temps moyen d'exécution du calculateur des règles potentielles et du moteur de décision avec deux types d'adaptateur applicatif :

- Un adaptateur applicatif simple qui retourne à chaque requête tous les rôles de l'utilisateur.
- Un adaptateur applicatif intelligent qui ne retourne que les rôles relatifs au service accédé.

La table 5.8 présente les résultats des tests de performances effectués sur les deux environnements présentés.

Temps moyen d'exécution (ms)	**Environnement 1**	**Environnement 2**
Sans protection obligatoire	29.3	330.2
Avec protection obligatoire	31.8	349.2
moniteur de référence	2.5	19.0
Surcharge (%)	8.5	5.8

TABLE 5.8 – Temps moyen d'exécution d'une page Web avec et sans protection obligatoire.

Le calcul de la surcharge imposée par l'utilisation de notre protection obligatoire est le suivant :

$$Surcharge = \frac{t_{moniteur}}{t_{sans\ protection}} = \frac{19}{330.2} = 5.8\%$$

Ces résultats montrent que pour un environnement complet, la surcharge due à notre protection obligatoire est inférieure à 9%. De plus, quand la taille de l'application augmente, la surcharge due à notre protection obligatoire diminue. En effet, le temps d'exécution des pages Web de l'application augmente plus fortement que le temps d'exécution du moniteur de référence. Ce résultat est particulièrement intéressant car il montre que l'approche de protection obligatoire fonctionne à grande échelle.

Temps moyen d'exécution (ms)	**Environnement 1**	**Environnement 2**
Moniteur de référence		
Adaptateur simple	2.5	19.0
Adaptateur intelligent	2.2	17.0
Calcul des règles potentielles		
Adaptateur simple	0.75	5.65
Adaptateur intelligent	0.65	5.15
Moteur de décision		
Adaptateur simple	1.75	13.35
Adaptateur intelligent	1.55	11.85

TABLE 5.9 – Temps d'exécution moyen des différents composants.

Les temps moyen d'exécution des différents composants du moniteur de référence sur les deux environnements de tests sont présentés dans la table 5.9. Ces résultats nous montrent que le moteur de décision est le composant qui prend le plus de temps et qu'il représente environ 70% du temps global d'exécution du moniteur de référence :

$$\frac{tmoteur\ décision}{tmoniteur\ référence} = \frac{13.35}{19.0} = 70.3\%$$

$$\frac{tmoteur\ décision}{tmoniteur\ référence} = \frac{11.85}{17.0} = 69.7\%$$

De plus, cette proportion reste la même en utilisant un adaptateur applicatif simple ou intelligent. Elle est donc indépendante du nombre de rôles et du nombre de règles potentielles calculées (l'adaptateur applicatif intelligent permet de réduire le nombre de rôles retournés et donc le nombre de règles potentielles calculées).

Nous pouvons également remarquer à travers ces résultats que l'utilisation d'un adaptateur applicatif intelligent (c'est-à-dire filtrant les rôles retournés au moniteur de référence en fonction de la requête entrante) permet de réduire le temps d'exécution global d'environ 11% :

$$\frac{t_{adaptateur\ simple} - t_{adaptateur\ intelligent}}{t_{adaptateur\ simple}} = \frac{19.0-17.0}{19.0} = 11\%$$

Si ce gain est appréciable, nous voyons qu'il n'est pas indispensable pour le passage à l'échelle puisque les performances s'améliorent sans cela pour des grands environnements.

5.5 Test de validation de la protection

Nous avons proposé dans la partie 3.5 une méthode de vérification de notre protection obligatoire, dont nous avons décrit l'implémentation dans la partie 4.3.4. Cette méthode repose sur une architecture client/serveur et effectue un parcours exhaustif de toutes les actions qui peuvent être effectuées sur l'intégralité des ressources d'une application. Nous allons ici appliquer cette méthode de vérification sur le premier environnement décrit dans la partie 5.4.

Le nombre de cas de tests à effectuer est le produit du nombre de sujets S par le nombre de relations R. Ici, nous considérons que les sujets correspondent aux rôles et les relations aux ressources. Ainsi, le nombre de cas de tests à effectuer est le produit du nombre de rôles applicatifs $N_{rôles}$ par le nombre de ressources $N_{ressources}$:

$$C = S * R = N_{rôles} * N_{ressources} = 176 * 164 = 28864$$

Pour rappel, la politique de sécurité associée à cet environnement de test contenait 1644 règles.

Nous avons explicité dans la définition 13 (cf. partie 3.5) les conditions de validation de la protection à partir des résultats retournés par notre méthode de vérification. Cette méthode de vérification nous permet de connaître le nombre d'accès refusés, le nombre A_P d'accès légitimement autorisés (c'est-à-dire les accès pour lesquels une règle d'autorisation existe dans la politique de sécurité) et le nombre $A_{\overline{P}}$ d'accès illégitimement autorisés (pour lesquels il n'existe pas de règle d'autorisation dans la politique de sécurité).

Temps d'exécution

L'exécution complète du test de vérification de notre protection obligatoire a pris sur cet environnement *2 heures et 41 minutes*. La durée de ce test complet semble acceptable vis-à-vis de la taille de l'environnement. Celle-ci montre qu'un tel test est en pratique réalisable. Même si nous n'avons pas effectué ce test sur le second environnement à grande échelle, nous pouvons estimer par proportionnalité que la durée de test serait de *18 heures et 36 minutes*.

L'architecture client/serveur sur laquelle repose cette méthode permet d'assurer un test de bout en bout, c'est-à-dire de la requête HTTP émise par le client, du traitement de cette requête par le serveur Web et le renvoi de la réponse HTTP au client par le serveur Web. Le temps de test moyen d'une requête (c'est-à-dire le rapport de la durée d'exécution du test complet par le nombre de tests effectués), est ici égal à 330 ms. Dans ce temps de test rentre en compte le temps de traitement effectif de la requête par le serveur, mais également le temps de transfert réseau de la requête et de la réponse et l'analyse de la réponse par notre outil de test. Le temps moyen de traitement était pour cet environnement de 31.8ms (cf. table 5.8).

Résultats

Les résultats obtenus lors de l'exécution de notre méthode de test sur le premier environnement sont les suivants :

$$N = 28864,\ A_P = 1644,\ A_{\overline{P}} = 0$$

En appliquant la définition 13 pour vérifier les conditions de validation de la protection, nous obtenons :

$$T_C = \frac{N}{C} = \frac{28864}{28864} = 1$$

$$T_P = \frac{A_P}{N_P} = \frac{1644}{1644} = 1$$

$$T_I = \frac{A_{\overline{P}}}{N_P} = \frac{0}{1644} = 0$$

Ces valeurs vérifient bien les condition de sûreté définies pour notre protection obligatoire. Notre protection obligatoire est donc mise en œuvre de bout en bout et de façon sûre.

5.6 Conclusion

Dans ce chapitre, nous avons présenté sur un exemple réel de workflow le calcul automatisé de politiques de sécurité à partir d'un modèle spécifique de workflows, que ce soit pour les différentes étapes composant un workflow que pour les services externes utilisés par les workflows. Nous avons également décrit la méthode utilisée pour obtenir les politiques de sécurité du socle applicatif QualNet, pour lequel il n'existait pas de spécification clairement définie. Cependant, nous avons défini un mode d'apprentissage qui permet de faciliter l'obtention de la politique de sécurité pour ce socle applicatif.

Dans une seconde partie, nous avons déroulé un cas concret d'autorisation en décrivant à partir des rôles applicatifs d'un utilisateur les contextes de sécurité objets et les règles potentielles d'accès obtenus par l'application des algorithmes présentés dans le chapitre 4, ainsi que la prise de décision.

Nous avons mis en place des protocoles de vérifications expérimentales des complexités algorithmiques que nous avions calculés dans le chapitre 4. Ceux-ci reposent sur la mesure du temps d'exécution des différents composants de notre protection obligatoire, en fonction de certains paramètres comme le nombre de rôles, la profondeur de calcul ou la taille de la politique de sécurité. Ainsi, nous confirmons expérimentalement les complexités algorithmiques théoriques que nous avions obtenues. Le calcul des règles potentielles d'accès et la prise de décision sont bien uniquement dépendants de la profondeur de calcul défini et du nombre de rôles applicatifs de l'utilisateur. Le point le plus important est l'indépendance de la prise de décision d'autorisation vis-à-vis de la taille de la politique de sécurité appliquée.

De plus, nous avons effectué des mesures de performances de notre protection obligatoire sur deux environnements de workflows QualNet. Nous obtenons une surcharge en terme de temps de traitement des requêtes HTTP de l'ordre de 5 à 8 % selon la taille de l'environnement. Nous avons remarqué que cette surcharge diminue lorsque la taille de l'environnement augmente, le temps de traitement global des requêtes augmentant plus rapidement que celui de notre protection. Nous avons également mis en évidence que le composant le plus important en terme de temps de traitement est le mécanisme de prise de décision, et qu'il est possible sur ces environnements de réduire de 10 % cette surcharge par l'utilisation d'un adaptateur applicatif intelligent.

Enfin, nous avons vérifié la sûreté de notre protection et des politiques de sécurité calculées pour un environnement de workflows QualNet.

Chapitre 6

Conclusion

La contribution majeure de cette thèse est la proposition d'un système de protection obligatoire qui permet de garantir de manière dynamique des objectifs de sécurité sur des serveurs d'applications Web. La solution présentée combine, d'une part, une modélisation des applications Web qui permet de représenter tous types d'applications Web et de workflows, et d'autre part, une protection obligatoire dynamique s'appuyant sur un langage dédié. Ce langage permet de décrire les besoins en terme de contrôle d'accès et de représenter les autorisations d'un sujet sur un objet du serveur d'applications. Les modèles d'applications et de workflows proposés permettent de calculer automatiquement les politiques de contrôle d'accès requises. Ce résultat est important car il permet que la solution soit utilisable en pratique par des personnes non spécialistes de la sécurité. Il répond à la difficulté intrinsèque aux approches de contrôle d'accès obligatoire de calcul des politiques de sécurité. Notre solution répond bien aux contraintes d'étude que nous avions présentées en introduction. Celle-ci est suffisamment extensible pour prendre en compte différents modèles d'applications et de workflows. Elle n'est pas liée à un langage de programmation particulier. Une implémentation fonctionnelle sur les environnements de type Windows (serveur Web IIS et Framework .Net) a été proposée. Une automatisation du calcul des politiques est proposée. Une intégration industrielle à grande échelle au sein des applications de workflows de QualNet a été réalisée.

Tout d'abord, nous avons établi dans le chapitre 2 un état de l'art des modèles de contrôle d'accès, notamment pour les serveurs d'applications Web et les systèmes de workflows. Cet état de l'art nous permet de conclure à l'absence de solutions efficaces traitant de contrôle d'accès obligatoire pour les serveurs d'applications Web et les workflows. Le besoin d'une méthode de calcul automatisé des politiques de sécurité pour faciliter l'administration des systèmes de contrôle d'accès obligatoire est également mis en avant.

Dans le chapitre 3 nous avons proposé la formalisation d'un modèle abstrait permettant de représenter tous types d'applications Web. Nous avons également défini un langage de protection dédié permettant d'exprimer les besoins en terme de contrôle d'accès au sein d'un serveur d'application Web. Ce langage s'appuie sur le modèle d'application proposé. Il est suffisamment extensible pour pouvoir représenter les sujets (les entités actives du système) et les objets (c'est-à-dire des ressources hébergées sur le serveur d'applications Web). De plus, nous avons proposé une architecture de protection obligatoire dynamique permettant d'appliquer de manière efficace des politiques de sécurité, exprimées dans notre langage, sur un serveur d'applications Web. Enfin, nous avons exprimé une méthode permettant de calculer de manière automatisée les politiques de sécurité à appliquer sur une application Web à partir du modèle d'accès abstrait de cette application. Cette méthode de calcul facilite grandement l'obtention de la politique de sécurité d'une application dès

lors que son modèle abstrait est établi. L'administration de notre protection obligatoire en est de fait grandement facilitée.

Dans le chapitre 4 nous présentons une implémentation de notre protection obligatoire sur des environnements Windows utilisant un serveur Web IIS et le framework .Net. Cette implémentation est indépendante des applications Web protégées car elle repose sur l'utilisation d'un adaptateur applicatif pour s'interfacer avec n'importe quelle application, et peut donc notamment fonctionner sur les applications de Workflows développées par QualNet mais également tout autre type d'application Web. Les algorithmes de calcul des contextes objets relatifs à une ressource, de calcul des règles potentielles d'accès et de prise de décision sont présentés et leur complexité y est étudiée. Nous avons montré que la prise de décision d'autorisation d'une requête est indépendante de la taille de la politique de sécurité appliquée, ce qui permet l'utilisation de notre protection obligatoire sur des environnements à grande échelle. Nous avons également détaillé dans ce chapitre l'implémentation de la méthode de calcul automatisé des politiques de sécurité proposée dans le chapitre 3. Nous avons présenté le modèle de workflows de QualNet et détaillé la méthode permettant de transposer ce modèle de workflow vers notre modèle abstrait d'application pour en déduire ensuite les politiques de sécurité requises. Ainsi, nous avons montré qu'il est possible de calculer de manière automatisée les politiques de sécurité associées aux systèmes de workflows de Qualnet.

Enfin, dans le chapitre 5 nous avons détaillé sur un exemple la méthode calcul automatisé des politique de sécurité. Nous avons également déroulé un cas d'exécution concret de notre protection obligatoire en détaillant les contextes de sécurité et les règles potentielles d'accès relatives à notre exemple. De plus, nous avons vérifié de manière expérimentale les complexités algorithmiques explicitées dans le chapitre 4 et prouvé que celles-ci correspondent bien aux résultats attendus. Pour finir, nous avons effectué des tests de charges et de performances sur notre protection obligatoire dans le but d'évaluer la surcharge sur le temps de traitement d'une requête par le serveur web. Cette surcharge, inférieure à 5 %, reste faible vis-à-vis de la taille des environnements protégés et des politiques de sécurités appliquées. De plus, les performances s'améliorent lorsque la taille de l'application augmente.

Perspectives

Projection des règles de contrôle d'accès au niveau intergiciel

Maintenant que nous pouvons exprimer les besoins en terme de contrôle d'accès au niveau applicatif, il peut être intéressant de pouvoir projeter ces besoins à un plus bas niveau. Ainsi, il serait possible de contrôler au niveau intergiciel les interactions entre les différents objets qui composent une page web par exemple. Cette approche serait intéressante car elle permettrait de contrôler les flux entre les différents composants hébergés pour une même ressource Web dynamique telle qu'une page ASP.NET.

Appliquer notre solution à d'autre modèles d'applications et de workflows

Nous avons présenté une utilisation de notre protection obligatoire et de notre méthode de calcul des politiques de sécurité pour les applications et le modèle de workflows de QualNet. Étant donné le niveau d'extensibilité du modèle que nous proposons, il est envisageable de pourvoir l'appliquer à des modèles de workflows standardisés tels que BPEL

ou BPMN par exemple. Le modèle de workflows que nous proposons reprend la majorité des notions proposées dans ces modèles, et est suffisamment extensible pour s'adapter à ces modèles.

Coupler notre solution à des outils avancés de détection et de prévention d'intrusions

Policy interaction Graph Analysis (PIGA, [Briffaut, 2007, Briffaut *et al.*, 2009]) est un mécanisme de contrôle d'accès obligatoire et de détection d'intrusions qui est capable de contrôler des propriétés de sécurité avancées mêlant flux directs et indirects. En effet, actuellement notre approche contrôle uniquement les flux directs (la notion de relation indirecte de notre approche représente aussi des flux directs entre un sujet et un objet) au moyen d'une politique de contrôle d'accès obligatoire. Ce contrôle direct est un prérequis à l'approche PIGA qui nécessite des règles de contrôle d'accès obligatoire de type $Sujet -> Permission -> Objet$ telles que nous les proposons pour le Web. Ainsi, nous pourrions garantir des propriétés globales entre différents workflows par exemple.

Implémentation sur d'autres environnements

Dans cette thèse, nous avons présenté une implémentation de notre modèle de protection obligatoire sur des environnements Web Microsoft (IIS et framework .Net). Une implémentation de cette protection sur des environnements de type Apache/PHP et Java est envisageable car notre approche globale de protection n'est pas dépendante de l'architecture d'applications Web utilisée sur les environnements Microsoft. Ainsi, nous pourrions par exemple contrôler les flux entre les composants Java d'une page JSP.

Bibliographie

[Anderson, 1980] Anderson, J. P. (1980). Computer security threat monitoring and surveillance. Rapport technique, James P Anderson Co., Fort Washington, PA.

[Atluri et Huang, 1996] Atluri, V. et Huang, W. (1996). An authorization model for workflows. *In Proceedings of the 4th European Symposium on Research in Computer Security*, pages 44–64. Springer-Verlag.

[Bell et LaPadula, 1973] Bell, D. E. et LaPadula, L. J. (1973). Secure computer systems : Mathematical foundations and model. Rapport technique M74-244, The MITRE Corporation, Bedford MA.

[Betgé-Brezetz *et al.*, 2013] Betgé-Brezetz, S., Kamga, G.-B., Dupont, M.-P. et Guesmi, A. (2013). End-to-end privacy policy enforcement in cloud infrastructure. *In* Fu, X., Sharma, P., Huang, D. et Medhi, D., éditeurs : *CLOUDNET*, pages 25–32. IEEE.

[Biba, 1977] Biba, J. K. (1977). Integrity considerations for secure computer systems.

[Bishop, 2003] Bishop, M. (2003). *Computer Security – Art and Science*. Addison Wesley Professional.

[Boebert et Kain, 1985] Boebert, W. E. et Kain, R. Y. (1985). A practical alternative to hierarchical integrity policies. *In 8th National Computer Security Conference*.

[Bouchaudy, 2013] Bouchaudy, J. (2013). *Linux administration t.4*. Eyrolles.

[Briffaut, 2007] Briffaut, J. (2007). *Formalisation et garantie de propriétés de sécurité système : application à la détection d'intrusions*. Thèse de doctorat, Université d'Orléans. SDS.

[Briffaut *et al.*, 2009] Briffaut, J., Lalande, J. F. et Toinard, C. (2009). Formalization of security properties : enforcement for mac operating systems and verification of dynamic mac policies. *International journal on advances in security*, 2(4):325–343.

[Cherrueau *et al.*, 2013] Cherrueau, R.-A., Douence, R., Royer, J.-C., Südholt, M., De Oliveira, A. S., Roudier, Y. et Dell'Amico, M. (2013). Reference monitors for security and interoperability in OAuth 2.0. *In SETOP 2013, 6th International Workshop on Autonomous and Spontaneous Security, 12-13 September 2013, Rhul, Egham, UK*, Rhul, ROYAUME-UNI.

[Clark et Wilson, 1987] Clark, D. D. et Wilson, D. R. (1987). A Comparison of Commercial and Military Computer Security Policies. *IEEE Symposium on Security and Privacy*, page 184.

[Cvrcek, 2000] Cvrcek, D. (2000). Mandatory access control in workflow systems. *In Proceedings of the JCKBSE Conference*, pages 247–254.

[Debricon *et al.*, 2009] Debricon, S., Bouquet, F. et Legeard, B. (2009). From business processes to integration testing. *In* Zendra, O., éditeur : *IDM'09, 5èmes journées sur l'Ingéniérie Dirigée par les Modèles*, volume 1, Nancy, France.

[Dell'Amico *et al.*, 2012] Dell'Amico, M., Serme, G., Idrees, M. S., de Oliveira, A. S. et Roudier, Y. (2012). HiPoLDS : A hierarchical security policy language for distributed systems. *Information Security Technical Report*.

[Ferraiolo et Kuhn, 1992] Ferraiolo, D. et Kuhn, R. (1992). Role-based access controls. *In 15th National Computer Security Conference*, pages 554–563, Baltimore, MD, USA.

[Fielding, 2000] Fielding, R. T. (2000). *REST : Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine.

[Gros *et al.*, 2012] Gros, D., Toinard, C. et Briffaut, J. (2012). Contrôle d'accès mandataire pour Windows 7. *In SSTIC 2012*, pages 266–291, Rennes, France.

[Group, 2012] Group, I. W. A. O. W. (2012). The oauth 2.0 authorization. technical report rfc 6749.

[Harrison *et al.*, 1976] Harrison, M. A., Ruzzo, W. L. et Ullman, J. D. (1976). Protection in operating systems. *Commun. ACM*, 19(8):461–471.

[Hu *et al.*, 2007] Hu, V. C., Martin, E., Hwang, J. et Xie, T. (2007). Conformance checking of access control policies specified in xacml. *In COMPSAC (2)*, pages 275–280. IEEE Computer Society.

[Huang, 1998] Huang, W.-K. (1998). *Incorporating Security into Workflow Management Systems*. Thèse de doctorat, Rutgers The State University of New Jersey.

[ITSEC, 1991] ITSEC (1991). Information technology security evaluation criteria v1.2. Rapport technique, ITSEC.

[Kim, 2009] Kim, Y. J. (2009). Access control service oriented architecture security. *Washington Universityin St. Louis, Student Reports project on Recent Advances in Network Security*.

[Kolovski *et al.*, 2007] Kolovski, V., Hendler, J. et Parsia, B. (2007). Analyzing web access control policies. *In Proceedings of the 16th international conference on World Wide Web*, pages 677–686, Banff, Alberta, Canada. ACM.

[Koshutanski et Massacci, 2003] Koshutanski, H. et Massacci, F. (2003). An access control framework for business processes for web services. *In* Jajodia, S. et Kudo, M., éditeurs : *XML Security*, pages 15–24. ACM.

[Lampson, 1969] Lampson, B. W. (1969). Dynamic protection structures. *In Proceedings of the November 18-20, 1969, Fall Joint Computer Conference*, volume 35 de *AFIPS '69 (Fall)*, pages 27–38, Las Vegas, Nevada. AFIPS Press.

[Lampson, 1971] Lampson, B. W. (1971). Protection. *In Proceedings of the 5th Symposium on Informations Sciences and Systems*, pages 437–443, Princeton University.

[Lampson, 1973] Lampson, B. W. (1973). A note on the confinement problem. *Communications of the ACM*, 16(10):613–615.

[Leitner *et al.*, 2011] Leitner, M., Rinderle-Ma, S. et Mangler, J. (2011). Aw-rbac : Access control in adaptive workflow systems. *In ARES*, pages 27–34. IEEE.

[Lindholm *et al.*, 2013] Lindholm, T., Yellin, F., Bracha, G. et Buckley, A. (2013). *The Java Virtual Machine Specification, Java SE 7 Edition*. Pearson Education.

[Liu *et al.*, 2011] Liu, A. X., Chen, F., Hwang, J. et Xie, T. (2011). Designing fast and scalable xacml policy evaluation engines. *IEEE Trans. Computers*, 60(12):1802–1817.

[Loscocco *et al.*, 1998] Loscocco, P. A., Smalley, S. D., Muckelbauer, P. A., Taylor, R. C., Turner, S. J. et Farrell, J. F. (1998). The inevitability of failure : The flawed assumption of security in modern computing environments. *In In Proceedings of the 21st National Information Systems Security Conference*, pages 303–314, Alington, Virginia, USA.

[Masi *et al.*, 2012] Masi, M., Pugliese, R. et Tiezzi, F. (2012). Formalisation and implementation of the xacml access control mechanism. *In* Barthe, G., Livshits, B. et Scandariato, R., éditeurs : *ESSoS*, volume 7159 de *Lecture Notes in Computer Science*, pages 60–74. Springer.

[Microsoft, 2013a] Microsoft (2013a). *Code transparent de sécurité*. `http://msdn.microsoft.com/fr-fr/library/ee191569(v=vs.110).aspx`.

[Microsoft, 2013b] Microsoft (2013b). *Common Language Runtime*. `http://msdn.microsoft.com/fr-fr/library/8bs2ecf4(v=vs.110).aspx`.

[Microsoft, 2013c] Microsoft (2013c). Windows integrity mechanism design. *In The Microsoft Developper Network*.

[Nair *et al.*, 2008] Nair, S. K., Simpson, P. N. D., Crispo, B. et Tanenbaum, A. S. (2008). Trishul : A policy enforcement architecture for java virtual machines.

[Nielsen *et al.*, 2003] Nielsen, H. F., Mendelsohn, N., Moreau, J. J., Gudgin, M. et Hadley, M. (2003). SOAP version 1.2 part 1 : Messaging framework. W3C recommendation, W3C.

[Noseevich et Petukhov, 2011] Noseevich, G. et Petukhov, A. (2011). Detecting insufficient access control in web applications. *In Proceedings of the 2011 First SysSec Workshop*, SYSSEC '11, pages 11–18, Washington, DC, USA. IEEE Computer Society.

[OASIS, 2007] OASIS (2007). Web Services Business Process Execution Language Version 2.0. Rapport technique, OASIS Web Services Business Process Execution Language (WSBPEL) TC.

[OASIS, 2012] OASIS (2012). Reference architecture foundation for service oriented architecture version 1.0. Rapport technique, OASIS.

[OASIS, 2013] OASIS (2013). *XACML*. `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`.

[Olivier *et al.*, 1998] Olivier, M. S., van de Riet, R. P. et Gudes, E. (1998). Specifying application-level security in workflow systems. *In* Wagner, R., éditeur : *Proceedings of the Ninth International Workshop on Security of Data Intensive Applications (DEXA 98)*, pages 346–351. IEEE.

[(OMG), 2011] (OMG), O. M. G. (2011). Business process model and notation (bpmn) version 2.0. Rapport technique, Object Management Group (OMG).

[Platt, 2001] Platt, D. S. (2001). *Découvrir Microsoft .Net*. Dunod.

[Saltzer et Schroeder, 1975] Saltzer, J. et Schroeder, M. (1975). The protection of information in computer systems. *Proceedings of the IEEE*.

[Sandhu, 1988] Sandhu, R. S. (1988). The schematic protection model : Its definition and analysis for acyclic attenuating schemes. *J. ACM*, 35(2):404–432.

[Sandhu, 1990] Sandhu, R. S. (1990). Separation of duties in computerized information systems. *In DBSec*, pages 179–190.

[Sandhu, 1992] Sandhu, R. S. (1992). The typed access matrix model. *In Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 122–136, Oakland, CA, USA.

[Spencer *et al.*, 1999] Spencer, R., Smalley, S., Loscocco, P., Hibler, M., Lepreau, J. et Andersen, D. (1999). The flask security architecture : System support for diverse security policies. *In Proceedings of the Eighth USENIX Security Symposium*, pages 123–139.

[Splengler, 2002] Splengler, B. (2002). Detection, prevention and containment : a study of grsecurity. *In Libre software Meeting*, Bordeaux.

[TCSEC, 1985] TCSEC (1985). Trusted computer system evaluation criteria. Rapport technique, 5200.28-STD, DoD Computer Security Center.

[Thobois, 2013] Thobois, L. (2013). *Internet Information Services (versions 7 et 7.5)*. Eyrolles.

[Venelle *et al.*, 2013] Venelle, B., Briffaut, J., Clevy, L. et Toinard, c. (2013). Mandatory access control for the java virtual machine. *In 16th IEEE Computer Society Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2013)*. IEEE Computer Society.

[Wainer *et al.*, 2001] Wainer, J., Barthelmess, P. et Kumar, A. (2001). W-rbac a workflow security model incorporating controlled overriding of constraints.

[Ward et Hamblin, 2006] Ward, R. et Hamblin, J. (2006). P.b. mandatory integrity control.

[WFMC, 2012] WFMC (2012). Workflow management coalition workflow standard : Process definition interface – xml process definition language version 2.2. Rapport technique, Workflow Management Coalition, Lighthouse Point, Florida, USA.

[Wright *et al.*, 2002] Wright, C., Cowan, C., Morris, J., Smalley, S. et Kroah-Hartman, G. (2002). Linux security modules : General security support for the linux kernel. *In Proceedings of the 11th USENIX Security Symposium*, pages 17–31, Berkeley, CA, USA.

Maxime FONDA

Protection obligatoire des serveurs d'applications Web : Application aux processus métiers.

Dans cette thèse, nous nous intéressons au contrôle d'accès obligatoire dans les serveurs d'applications Web. Nous présentons une approche de protection obligatoire fondée sur un modèle abstrait d'applications Web. Les modèles d'applications Web existants, comme par exemple SOA peuvent être représentés par ce modèle abstrait d'application. Notre protection obligatoire s'appuie sur un langage de protection dédié permettant d'exprimer les besoins en terme de contrôle d'accès au sein d'un serveur d'application Web. Ce langage de protection utilise notre modèle d'application pour contrôler de manière efficace les accès des sujets aux objets de l'applications Web. Nous établissons également une méthode de calcul automatisé des politiques de sécurité qui facilite donc l'administration de la protection obligatoire proposée. Une implémentation sur des environnements Microsoft basés sur le serveur Web IIS et le canevas .Net est présentée. La solution est indépendante des applications Web protégées car elle repose sur l'utilisation d'un adaptateur applicatif pour s'interfacer avec n'importe quelle application. Celle-ci est fonctionnelle sur des environnements de workflow de la société QualNet ayant co-financée cette thèse. Les expérimentations menées montrent que notre protection obligatoire supporte des environnements à grande échelle et impose une élévation faible du temps de traitement, de l'ordre de 5%, qui diminue lorsque la taille des applications augmente.

Mots clés : Sécurité, contrôle d'accès, serveurs Web, protection obligatoire, workflows.

Mandatory protection of Web applications servers : Usage for the Workflow environments.

This thesis focuses on mandatory access control in Web applications server. We present a novel approach of mandatory protection based on an abstract Web application model. Existing models of Web applications such as SOA fit with our abstract model. Our mandatory protection uses a dedicated language that allows to express the security requirements of a Web application. This dedicated protection language uses our Web application model to control efficiently the accesses of the subjects to the objects of a Web application. We establish a method to automatically compute the requested security policies facilitating thus the administration of the mandatory protection. An implementation on Microsoft-based environments uses the IIS Web server and the .Net Framework. The solution is independent from the Web applications to protect since it uses an application adaptor to interface our mandatory protection with the applications. This implementation is fully running on the workflow environments from the QualNet society, that cofunded this Ph.D thesis. Experiments show that our mandatory protection supports large scale environments since the overhead is near to 5 % and decreases when the size of the application increases.

Keywords : Security, access control, Web servers, mandatory protection, workflows.

QualNet.

Laboratoire d'Informatique Fondamentale d'Orléans
Rue Léonard de Vinci
B.P. 6759 F-45067 ORLEANS Cedex 2

QualNet.