Fédération de modèles pour l'analyse de cybersécurité du point de vue d'un attaquant

Bastien Drouot

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

ENSTA Bretagne

THESE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPERIEUR
DE TECHNIQUE AVANCEES BRETAGNE

ECOLE DOCTORALE N°601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

Bastien DROUOT

Fédération de modèles pour l'analyse de cybersécurité du point de vue d'un attaquant.

Thèse présentée et soutenue à Brest, le 14 avril 2021
Unité de recherche : Laboratoire Lab-STICC, UMR, 6285

Rapporteurs avant soutenance :

Mireille BLAY-FORNARINO	Professeur des Universités, IUT de Nice
Luis FERREIRA PIRES	Associate Professor, University of Twente Enschede, Pays-Bas

Composition du Jury :

Président :	Antoine BEUGNARD,	Professeur, IMT-Atlantique
Examinateurs :	Mireille BLAY-FORNARINO	Professeur des Universités, IUT de Nice
	Frédéric BONIOL,	Professeur des Universités, ONERA Toulouse
	Luis FERREIRA PIRES,	Associate professor , University of Twente Enschede, Pays-Bas
Dir. de thèse :	Philippe DHAUSSY	Professeur, ENSTA Bretagne, Lab-STICC, Brest
Enc. de thèse :	Joël CHAMPEAU	Enseignant chercheur, ENSTA Bretagne, Lab-STICC, Brest

Abstract et Résumé

Abstract

Cybersecurity encompasses multiple problem areas. One of those, "Cyber Threat Analysis" (CTA), intersects various domains, system modeling, attacker modeling and threat description, where each one evolves independently. CTA is covered by multiple approaches, such as penetration testing or fault injection testing. To consider a-priori analysis methods, models are required. In this manuscript, we focus on the attacker viewpoint modeling approachs.

Abundance of cybersecurity's standards must be taken into account in our work. Indeed, using various Domain Specific Modeling Languages (DSMLs), the CTA's experts describe their models in a language relevant to their problem area.

In this modeling context, we must consider several open research questions. Firstly, how to address the interoperability problem between heterogeneous DSMLs. Secondly, how to handle, in our modeling context, the evolving nature of CTA's methods and knowledge due to the effervescence of the research domain.

To tackle these issues, in this manuscript we propose a role-based federation approach to create dynamic and adaptive attacker's viewpoints linked to existing source models.

First, we define and model the attacker's viewpoint in a CTA context. Based on cybersecurity standards as source models, our definition covers all attacker's concerns in this scope. To take into account the necessity to adapt and to evolve this definition, we propose a modeling framework based on several viewpoints representing different concerns. Additionaly, we construct viewpoints on viewpoints in order to represent specific concerns or interpret specific attacker behavior.

Then, we present our modeling language to express viewpoints, named Role4All. The concept of role is central to our approach, allowing us to represent and to federate various DSMLs in order to generate viewpoints. Interoperability between the domain models is crucial to apply dedicated algorithms and interpretations on the attacker's viewpoints. In our work, dedicated viewpoints are used by security analysis tools to simulate attacks on a system.

A case study on development and on interpretation of attacker's viewpoint validates our approach. Our case study is based on the federation of multiple domain models describing, as an example, vulnerability definition, architectures description or configuration management. In particular, we highlight the use of roles to dynamically adapt attacker's perception on the attacked system.

Résumé

La cybersécurité est un domaine de recherche très étendu qui couvre de nombreux aspects. L'un d'entre eux, la modélisation et l'analyse de menaces, regroupe différents domaines qui évoluent indépendamment les uns des autres : la modélisation de systèmes, la modélisation d'attaquants et la description de menaces. Il existe un grand nombre d'approches dédiées à la modélisation et à l'analyse de la menace, telles que, des tests de pénétration ou des tests d'injection de fautes. Cependant, si l'on souhaite utiliser des approches d'analyses proactives, il est nécessaire d'utiliser des modèles pour raisonner sur ceux-ci. Dans ce manuscrit, nous nous sur concentrons sur l'une de ces approches : la modélisation du point de vue d'un attaquant.

La littérature autour de la cybersécurité comporte un grand nombre de normes et de standards qui doivent être pris en considération dans nos travaux. Cette prolifération de solutions est particulièrement notable lorsque des experts, soumis à des problématiques spécifiques, décrivent différents modèles pour un même système. Et ce, à l'aide de langages de modélisation spécifiques à leurs domaines (DSML).

Dans ce contexte de modélisation, nous devons considérer plusieurs questions de recherche, premièrement, comment résoudre le problème d'interopérabilité entre des DSMLs hétérogènes, et deuxièmement, comment gérer, dans notre contexte de modélisation, la nature évolutive des méthodes et des connaissances propres à ce domaine de recherche.

Pour résoudre ces problématiques, nous proposons dans ce manuscrit une approche de fédération fondée sur les rôles. Cette dernière permet de créer des points de vue d'attaquants qui sont dynamiques, adaptatifs et en lien avec les modèles existants.

Tout d'abord, nous définissons et modélisons le point de vue d'un attaquant dans le domaine de modélisation et d'analyse de la menace. La définition proposée découle des normes et des standards de la cybersécurité, elle couvre, l'ensemble des préoccupations d'un attaquant dans notre domaine d'étude. Pour tenir compte de la nécessité d'adapter et de faire évoluer notre définition, nous proposons un cadre de modélisation basé sur plusieurs points de vue représentant différents ensembles de préoccupations. De plus, nous permettons la construction de points de vue sur des points de vue, cette fonctionnalité est utilisée pour représenter des préoccupations spécifiques et pour interpréter le comportement d'un attaquant spécifique.

Par la suite, nous présentons Role4All, notre langage de modélisation de points de vue. Le concept de rôle est au cœur de notre approche, car il nous permet de générer des points de vue représentant et fédérant différents DSMLs. L'interopérabilité entre les modèles est indispensable pour appliquer des algorithmes et des interprétations sur le point de vue d'un attaquant. C'est pourquoi, dans notre approche, des points de vue dédiés sont utilisés par des outils d'analyse de sécurité afin de fédérer les informations nécessaires pour simuler des attaques sur un système.

Nous validons notre approche au travers d'une étude de cas sur le développement et l'interprétation du point de vue d'un attaquant. Cette étude est basée sur la fédération de nombreux modèles décrivant, par exemple, des vulnérabilités, des architectures ou des configurations. En particulier, nous mettons en évidence l'utilisation des rôles pour adapter dynamiquement la perception qu'un attaquant a d'un système.

Remerciements

Je tiens à commencer ces remerciements par mon encadrant Joël CHAMPEAU et mon directeur de thèse Philippe DHAUSSY. Ils m'ont accompagné tout au long de ma thèse et, sans eux, il m'aurait été difficile, voire impossible, de réaliser ce projet. J'ai apprécié la confiance qu'ils ont placé en moi et l'autonomie qu'ils m'ont accordé tout au long de mes travaux. Je remercie particulièrement Joël CHAMPEAU pour son expertise, tant dans le domaine de la modélisation que dans la conduite de projet. Cela m'a permis de comprendre les enjeux de ma thèse et d'appréhender mon domaine de recherche. Je remercie aussi ma référente DGA, Véronique SERFATY, qui m'a accompagnée tout au long de cette thèse.

Je remercie également les membres de mon CSI, Frédéric BONIOL et Antoine BEUGNARD. Bien que nous ayons peu interagi, leurs conseils éclairés m'ont permis d'orienter mes recherches et de définir plus clairement mon sujet. Je remercie ensuite mes rapporteurs de thèse : Mireille BLAY-FORNARINO et Luis FERREIRA PIRES pour leurs relectures attentives.

D'une manière plus générale, je tiens à remercier l'ensemble de mes collègues. Ciprian TEODOROV et Luka LE ROUX pour nos échanges autour de l'analyse formelle et de la modélisation en général. Pascale GAUTRON, Michel INIZAN et Loïc LAGADEC pour les discussions partagées au cours de ces années, tant professionnelles que politiques ou personnelles. Les doctorants et étudiants que j'ai côtoyés, dont certains sont devenus mes amis : Cyrielle FERON, Louis MORGE-ROLLET, Gregoire DE BROGLIE, Tithnara SUN, Emilien FOURNIER et bien d'autres. Je tiens à remercier tout particulièrement Vincent LEILDE, Fahad GOLRA et Sylvain GUERIN qui ont été d'une grande aide au cours de ces années, que ce soit pour la découverte de mon sujet, la rédaction d'articles ou encore l'implémentation de Role4All. Je mesure la chance qu'il m'a été donné de faire une thèse au sein d'une équipe aussi compétente et bienveillante.

J'ai une affection particulière pour toutes les personnes qui ont relu et corrigé mon manuscrit, à savoir, mes collègues doctorants et amis proches. Il serait bien trop long de tous les énumérer ici, tant la tâche fut ardue. Sans leur travail collectif, ce manuscrit n'aurait jamais pu prendre forme, je les remercie donc tous chaleureusement. Je tiens tout de même à remercier spécifiquement Michèle HOFMANN qui a effectué la dernière relecture du manuscrit dans son intégralité.

Je remercie aussi tous ceux que j'ai oubliés lors de l'écriture de ces lignes, qu'ils soient des proches ou des collègues. Sachez que, même si je ne vous cite pas directement, je n'oublie pas les bons moments passés ensemble, autour d'une galette, lors de matches de foot, de parties de JDR ou encore de vacances partagées.

Je dédie la fin de mes remerciements aux personnes qui me sont le plus chères, à savoir, ma famille. Ma mère, qui m'a soutenue moralement durant toutes mes années d'études, loin du foyer familial, mon père, qui n'a pas hésité à faire des milliers de kilomètres pour venir faire le tour des thalassos bretonnes avec moi et mon frère qui a su me faire rire malgré la distance et a toujours gardé contact durant toutes ces années.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Introduction générale

L'ingénierie dirigée par les modèles (IDM) est utilisée dans différentes phases du cycle de vie des systèmes tels que les systèmes embarqués, les systèmes critiques de sécurité, etc [83]. L'IDM est également utilisée pour la gestion des connaissances, l'analyse de systèmes (à la fois structurelle et comportementale), et l'évaluation de performances (analyse du pire des cas, planification, etc.) [130]. Dans ce manuscrit, nous présentons une approche et un cadre de modélisation appliqués au domaine de la cybersécurité. La cybersécurité est un domaine de recherche actif où des modèles sont utilisés, en particulier pour représenter ce qu'un attaquant connaît et perçoit du système qu'il attaque.

1.1.1 Cybersécurité

La cybersécurité est une priorité pour les entreprises, les dirigeants politiques et les citoyens du monde entier. La quantité de littérature universitaire et technique couvrant le sujet est révélatrice de la nature complexe et multidisciplinaire du domaine. Ce domaine n'est pas encore clairement délimité d'un point de vue académique mais combine une multiplicité de disciplines allant du technique au comportemental et culturel. De plus l'évolution rapide des menaces et le rythme des changements techniques font de la cybersécurité un domaine d'étude vaste, complexe et en constante évolution.

Il n'existe pas aujourd'hui de définition mondialement acceptée et standardisée de la cybersécurité, cependant nous pouvons retenir cette définition : "les mesures de sauvegarde et les actions auxquelles il est possible de recourir pour protéger le cyberespace, dans les domaines civil et militaire, des menaces associées à ses réseaux interdépendants et à son infrastructure informatique ou susceptibles de leur porter atteinte. La cybersécurité vise à préserver la disponibilité et l'intégrité des réseaux et de l'infrastructure ainsi que la confidentialité des informations qui y sont contenues." [96].

Dans ce manuscrit nous nous focalisons sur les apports de la modélisation au domaine de la cybersécurité, en particulier pour la modélisation et l'analyse de menaces. Contrairement à d'autres approches de cybersécurité, tel que la détection d'intrusions, l'analyse et la modélisation des menaces est une approche pro-active. Elle consiste en l'utilisation de modèles pour analyser les problèmes de sécurité. Ces modèles sont utilisés pour développer des abstractions du système, des menaces, des attaquants et peuvent également contenir des détails sur les éléments d'un système tel que leurs configurations [146]. L'objectif est alors d'anticiper les crises de cybersécurité en recherchant les vulnérabilités pertinentes pour un système ou une organisation particulière. Les approches de modélisation et d'ana-

lyse de menaces se basent sur une distinction entre le système attaqué, l'attaquant et les menaces :

— La modélisation du système décrit la structure et le comportement du système, ses caractéristiques, ses limites, etc. Les éléments du système sont conceptualisés pour produire une abstraction de ce dernier.

— La modélisation de l'attaquant caractérise les objectifs de l'attaquant, ses compétences et ses scénarios d'attaque. La connaissance qu'un attaquant a d'un système dépend de ce qu'il perçoit du système mais aussi de ses capacités d'interaction avec ce dernier.

— La description de la menace détaille les vulnérabilités présentes sur un système. Une vulnérabilité peut être exploitée par un attaquant afin d'atteindre ses objectifs. L'exploitation successive de différentes vulnérabilités permet de définir un scénario d'attaque dépendant du système attaqué mais aussi des capacités et connaissances de l'attaquant.

Au final, la modélisation et l'analyse de menaces requièrent de représenter un système, un attaquant, des menaces mais aussi ce que l'attaquant perçoit du système et ce qu'il connaît des menaces existantes. Elles permettent de produire des modèles de sécurité [81] tels que des arbres et des graphes d'attaque. Ces modèles sont utilisés comme sources d'information pour des analyses de coût de sécurisation, de probabilité ou de dangerosité d'attaque, etc.

1.1.2 Modélisation

Loucopoulos and Zicari [115] caractérisent la modélisation conceptuelle comme "l'activité de description formelle de certains aspects du monde physique et social qui nous entoure à des fins de compréhension et de communication" [1]. Un modèle conceptuel est alors une description formelle de parties d'un domaine. Les modèles conceptuels sont utilisés à la fois par les humains, mais aussi par les ordinateurs afin de permettre, par exemple, la validation formelle et la génération d'artefacts.

Diverses recherches [154, 14, 69, 112] soulignent les lacunes des langages de modélisation classiques, tel que le modèle entité-relation [36] ou le langage de modélisation unifié (UML) [124], lorsqu'ils sont utilisés pour modéliser des domaines complexes, dépendants du contexte et dynamiques. Nous constaterons dans ce manuscrit que la modélisation et l'analyse des menaces de sécurité font partie de ces domaines.

Lors de la modélisation d'un système complexe, l'information est souvent fragmentée entre différents modèles et ceux-ci sont souvent exprimés dans différents langages de modélisation. De plus, afin de rassembler les informations pertinentes pour une partie prenante particulière, il est nécessaire de combiner et de réorganiser ces modèles en fonction des préoccupations de la partie prenante concernée.

Les points de vue sont une solution permettant de répondre aux problèmes soulevés par la modélisation de systèmes complexes. Il existe deux stratégies majeures pour la modélisation de point de vue :

— Les points de vue fixes [53]. Il existe alors un nombre fini de points de vue prédéfinis à utiliser dans différents domaines d'application ou scénarios. Dans ce cas, l'ensemble des vues possibles est alors limité aux points de vue existants.

— Les points de vue adaptatifs [140, 46]. Il est alors possible de construire une infinité de points de vue en fonction des métamodèles sources. Dans ce cas, l'ensemble des

1. En version original : "the activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication."

vues possibles n'est pas limité et une vue peut s'adapter aux changements du système observé.

Une vue, conforme à un point de vue, est une représentation d'un système satisfaisant un ensemble de préoccupations. La mise en place d'une vue, schématisée sur la figure 1.1, peut se découper en trois étapes :

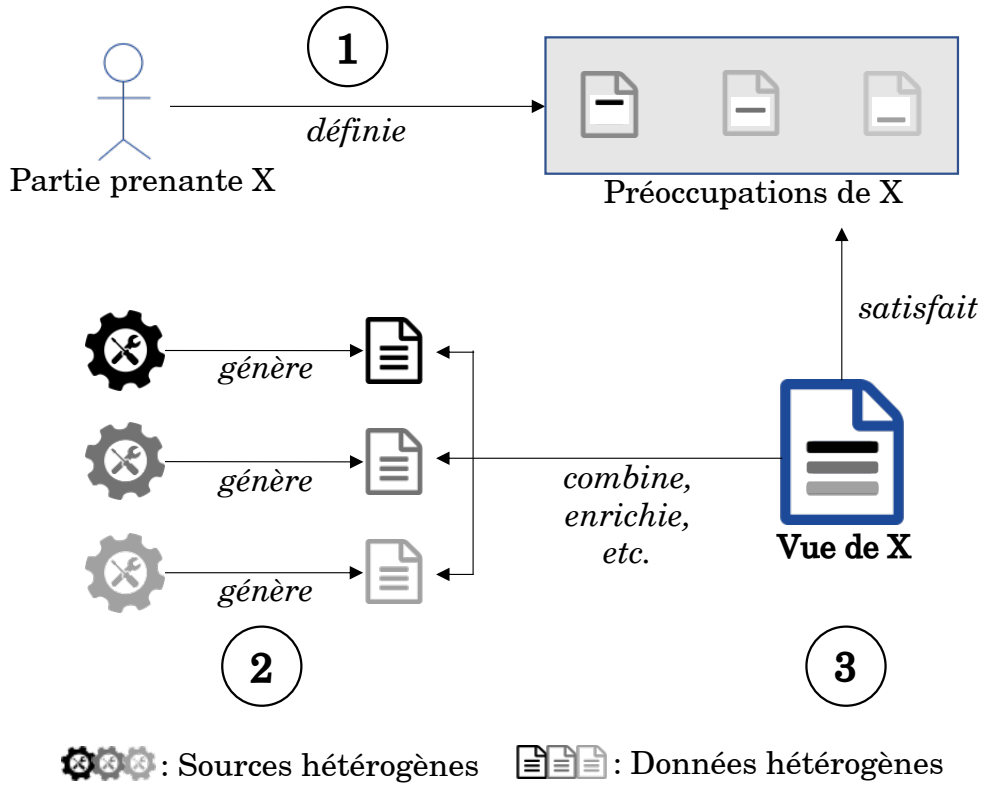


FIGURE 1.1 – Schématisation de la construction d'une vue

① Définition des préoccupations de la partie prenante : une partie prenante définit un ensemble de préoccupations relatives à un système. Par exemple, un comptable se focalise sur le coût de fabrication et le prix de vente d'un produit mais ne s'intéresse pas à son processus de fabrication, au contraire d'un technicien.

② Génération de données hétérogènes : un ensemble d'outils distincts génère des données hétérogènes représentant un système.

③ Création d'une vue : la vue de la partie prenante est construite comme une représentation des données décrivant le système. Cette représentation satisfait les préoccupations définies en ①.

Par construction il est possible de créer différentes vues pour un même système. Par exemple, Matsuda et al. [119] propose deux vues différentes (complétion et vérification) afin d'interagir avec un système.

La modélisation de points de vue est alors une solution permettant de représenter des systèmes complexes en rassemblant dans une même vue diverses informations pertinentes pour une partie prenante. La création d'une vue requière la combinaison et la réorganisation de données contenues dans des modèles hétérogènes.

1.2 Problématiques et contributions du manuscrit

La question abordée dans ce manuscrit est la modélisation et l'analyse de menaces via le point de vue d'un attaquant. Les objectifs sont de faciliter la compréhension des problèmes de sécurité par un humain via la représentation d'un point de vue et de produire des données utilisables par des outils d'analyse de sécurité, via des simulations ou de l'analyse formelle.

Comme nous l'avons introduit dans la section 1.1.1, la modélisation et l'analyse de menaces nécessitent la modélisation de systèmes complexes et dynamiques, ainsi que la prise en compte du comportement d'un attaquant. Nous caractérisons alors ce domaine d'études comme étant complexe, dépendant du contexte et dynamique. Comme présenté dans la section 1.1.2, la modélisation par point de vue est une solution permettant de couvrir ce type de domaine. Elle requière cependant la combinaison et la réorganisation de données hétérogènes.

Les problématiques abordées dans ce manuscrit sont alors la modélisation du point de vue d'un attaquant, la création de ce dernier via l'interopérabilité de modèles hétérogènes et la production de modèles de sécurité.

Les contributions apportées dans ce manuscrit permettent, *in fine*, à un humain de se représenter ce qu'un attaquant perçoit d'un système lors d'une attaque et à une machine de produire des modèles utilisables par les outils d'analyse de sécurité. La perception d'un attaquant est représentée grâce à un point de vue construit à partir de ses compétences, de ses objectifs, du système attaqué et des vulnérabilités exploitables. La production de modèles de sécurité est rendue possible via la fédération de données hétérogènes, la construction de points de vue dynamiques et l'interprétation de comportements d'attaquants.

Les contributions présentées dans ce manuscrit sont donc les suivantes :

1. Tout d'abord, une étape de définition et d'identification des préoccupations d'un attaquant. En partant de la littérature et des normes de cybersécurité, tel que la norme ISO 27000 [9], nous proposons une définition du terme d'attaquant dans le domaine de la modélisation et de l'analyse de menaces. De plus, nous énumérons les principales préoccupations d'un attaquant durant un scénario d'attaque.
2. Par la suite nous construisons des points de vue satisfaisants l'ensemble des préoccupations identifiées. Pour ce faire, nous construisons deux types de point de vue permettant respectivement de définir des concepts de référence communs à tous les types d'attaquants et des concepts spécifiques propres à un type d'attaquant particulier.
3. Une fois le point de vue d'un attaquant créé, nous le caractérisons grâce au diagramme de caractéristiques proposé par Bruneliere et al. [27].
4. Dans une seconde partie, nous proposons et formalisons Role4All, une solution de modélisation par les rôles permettant de construire des points de vue via la fédération de données hétérogènes.
5. Nous interprétons le comportement d'un attaquant lors d'une attaque au travers de son point de vue.
6. Enfin notre approche de modélisation du point de vue d'attaquant est validée sur un cas d'étude.

1.3 Structure du manuscrit

Ce manuscrit est divisé en six chapitres présentant l'état de l'art du domaine considéré, l'ensemble de nos contributions et la validation de notre approche, conformément au schéma présenté en figure 1.2 qui positionne les contributions dans les différents chapitre du manuscrit.

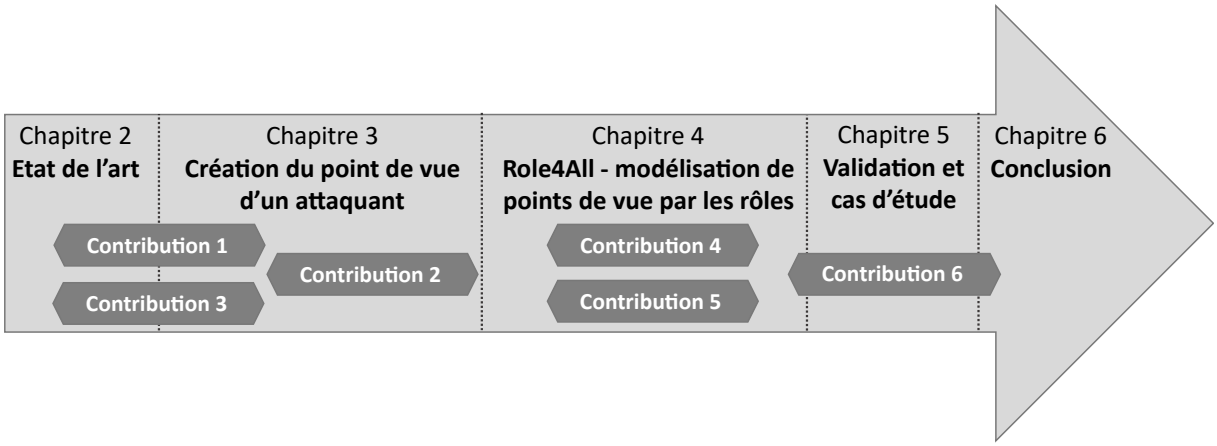


FIGURE 1.2 – Vue d'ensemble de ce manuscrit

Le chapitre 2 fait l'état de l'art du domaine. Tout d'abord concernant la cybersécurité et en particulier la modélisation et l'analyse de menaces. Ensuite, relatif à l'interopérabilité de modèles hétérogènes et principalement les approches de fédération de modèles. Une troisième section présente l'état de l'art touchant à la modélisation par point de vue et nommément la formalisation de cette notion. Enfin, une dernière section se focalise sur la modélisation par les rôles et l'utilisation de ces derniers dans divers domaines.

Le chapitre 3 présente une approche permettant de modéliser le point de vue d'un attaquant et d'effectuer des analyses à partir de ce dernier. Pour commencer nous décrivons la démarche de notre approche. Puis, nous construisons un ensemble de concepts permettant de décrire le point de vue d'un attaquant. Pour finir nous détaillons les étapes de construction et les contraintes propres au point de vue d'un attaquant.

Le chapitre 4 présente Role4All, notre contribution de modélisation par les rôles permettant de modéliser le point de vue d'un attaquant. Dans une première section nous introduisons le métamodèle de Role4All, formalisons notre utilisation de la notion de rôles et présentons une implémentation de Role4All. Dans une deuxième section nous décrivons les étapes de modélisation du point de vue d'un attaquant dans Role4All. Pour conclure ce chapitre, nous détaillons les étapes de construction d'un point de vue d'interprétation basé sur les rôles permettant d'interpréter le comportement d'un attaquant lors d'une attaque.

Le chapitre 5 permet de valider notre approche sur un cas d'étude. Dans un premier temps, nous présentons et modélisons un cas d'étude réel à l'aide d'outils variés. Par la suite nous construisons, à l'aide de Role4All, le point de vue d'un attaquant spécifique sur le système sélectionné. Pour finir, nous identifions un scénario d'attaque sur le système modélisé et nous validons ce dernier sur le système réel à l'aide de vrais outils de piratage.

Le chapitre 6 conclut ce manuscrit. Il récapitule les contributions apportées afin de répondre aux problématiques de ce manuscrit.

Chapitre 2

État de l'art

Sommaire

Compte tenu du contexte identifié lors de l'introduction du document, nous présentons, dans ce chapitre, l'état de l'art des recherche en terme de cybersécurité, d'intéropérabilité de formalismes et de modèles, de création de point de vue de modélisation et de modélisation par les rôles.

2.1 Cybersécurité

L'étude proposée dans cette section se focalise rapidement sur la modélisation et l'analyse du point de vue d'un attaquant.

En effet, la cybersécurité est un domaine de recherche couvrant de nombreux aspects. Il semble important voire incontournable de se reposer sur un ensemble de vocables sémantiquement bien définis et partagés. Dans ce cadre, une initiative de l'Union Européenne propose d'aligner les terminologies, définitions et domaines de la cybersécurité dans une taxonomie cohérente [125]. Elle comprend 15 domaines, 15 secteurs d'applications et 23 axes technologiques, chacun de ces éléments étant composé d'une dizaine de sous-domaines. Dans ce manuscrit, nous nous focalisons sur le domaine de la modélisation et de l'analyse de la menace de système d'information dans une optique de sécurité. Dans une première section nous définissons donc les principaux termes de notre domaine d'étude. Dans une seconde section nous présentons les standards et les méthodologies permettant de modéliser les menaces d'un système. La troisième partie se focalise sur les approches permettant l'analyse des menaces d'un système. Pour finir nous présentons notre axes de recherche en terme de cybersécurité.

2.1.1 Présentation générale

L'analyse et la modélisation de la menace interviennent tout le long du cycle de vie d'un système notamment pour la gestion des risques, l'élaboration d'exigences de sécurité, la conception sécurisée, la hiérarchisation de problèmes de sécurité, les tests de sécurité, la mise en production d'applications et la mise à jour d'applications.

L'analyse de la menace est articulée selon plusieurs axes et à différents niveaux : antiterrorisme [8], processus [160], etc. Pour bien caractériser notre domaine d'étude nous définissons l'analyse de la menace, dans ce manuscrit, comme un processus dans lequel la connaissance des vulnérabilités et des informations internes et externes pertinentes pour une organisation particulière sont comparées aux cyberattaques du monde réel. [160]

L'analyse de la menace a pour but d'évaluer des menaces et de renforcer la sécurité d'un système, elle peut se décomposer en quatre étapes distinctes :

— Acquisition de connaissances : cette étape de capture d'informations permet de définir ce qui fait ou non partie du système analysé.

— Mise en relation d'informations : les informations concernant un système sont associées et comparées à des bases de données afin d'identifier les vulnérabilités de ce système.

— Analyse de donnée : les informations collectées sont testées en vue de déterminer le niveau d'exposition du système.

— Mise en œuvre de solution : cette étape vise à proposer des mesures de sécurité visant à atténuer ou anticiper l'émergence de cyberattaque sur le système.

L'acquisition de connaissances est une étape de haut niveau de technicité pour laquelle il existe de nombreux outils dédiés (Nessus, SAINT, NeXpose, Qualys Scanner, Nikto, OpenVAS, Retina, NMap, etc.). Par ailleurs la mise en œuvre de solutions dépend principalement du système attaqué et de la politique de sécurité appliquée. La corrélation ou la mise en

relation des informations est une étape critique et importante, de même que l'analyse de ces informations restructurées. C'est pourquoi, nous nous focaliserons dans ce manuscrit sur les étapes de mise en relation et d'analyse.

La modélisation de la menace intervient lors de ces deux étapes, UcedaVelez and Morana [160] définissent la modélisation de la menace comme "un processus stratégique visant à examiner les scénarios d'attaque et les vulnérabilités possibles dans un environnement d'application proposé ou existant dans le but d'identifier clairement les niveaux de risque et d'impact" [1].

Ce processus peut être abordé selon divers méthodologies, STRIDE, DREAD, PASTA [160]. La méthodologie STRIDE proposée par Microsoft est la plus répandue [164], elle permet de simplifier la compréhension des menaces de sécurité d'un système. STRIDE n'est pas une méthodologie applicative car elle ne définit pas le processus dans lequel le modèle de menace doit être implanté, suivi et livré. Elle permet cependant d'organiser les menaces de sécurité contre un système en les classant dans les six domaines suivants : usurpation d'identité, falsification, répudiation, divulgation d'informations, déni de service et élévation de privilège.

La cybersécurité et en particulier l'analyse et la modélisation de la menace, se focalise sur un ensemble de propriétés de sécurité définies dans la norme ISO-27000 [9]. Cette norme liste les 6 propriétés de sécurité suivantes :

— Confidentialité : propriété selon laquelle l'information n'est pas diffusée ni divulguée à des personnes, des entités ou des processus non autorisés.

— Intégrité : propriété d'exactitude et de complétude.

— Disponibilité : propriété d'être accessible et utilisable à la demande par une entité autorisée.

— Authenticité : propriété selon laquelle une entité est ce qu'elle revendique être.

— Non-répudiation : capacité à prouver l'occurrence d'un événement ou d'une action donnée et des entités qui en sont à l'origine.

— Fiabilité : propriété relative à un comportement et à des résultats prévus et cohérents.

Kordy et al. [97] recensent 31 approches de modélisation de la menace utilisant pour la plupart le standard STRIDE et la terminologie ISO-27000 comme référence. Parmi les approches étudiées, plus de la moitié se focalisent sur le point de vue de l'attaquant, ce choix permet de centrer la modélisation sur la perception que l'attaquant a du système.

Nous retenons de cette présentation générale qu'il existe un foisonnement important des approches, des outils et des langages de modélisation et d'analyse de la menace. Ce foisonnement est pris en compte dans les travaux présentés dans ce manuscrit.

2.1.2 Modélisation d'un attaquant et des vulnérabilités d'un système

La modélisation et l'analyse de la menace appliquées au point de vue d'un attaquant se répartissent comme suit : la description de l'attaquant et de ses interactions possibles avec le système, et la modélisation du comportement de l'attaquant lors d'une attaque. La première partie est abordée dans la section 2.1.2 et la seconde dans la section 2.1.3.

Le standard ISO-27000 [9] définit une vulnérabilité comme une "faille dans un actif ou dans une mesure de sécurité qui peut être exploitée par une ou plusieurs menaces". Les actions possibles pour un attaquant sont alors, en plus de celles permises par le système, l'exploitation de vulnérabilités.

1. En version originale : Threat Modeling is a strategic process aimed at considering possible attack scenarios and vulnerabilities within a proposed or existing application environment for the purpose of clearly identifying risk and impact levels.

Dans [159] les auteurs déplorent un manque de standard pour la catégorisation de vulnérabilité et donnent un ensemble de lignes directrices permettant la création de taxonomie plus efficaces. Les auteurs préconisent de baser les taxonomies sur des standards tel que CVSS [3] et soulignent l'importance de la prise en charge de nouvelles vulnérabilités. Pour ce faire, ils privilégient les approches directement connectées au dictionnaire CVE [1], ce dernier regroupe un grand nombre de vulnérabilités et est mis à jour régulièrement.

En 2018, une enquête est réalisée couvrant la littérature associée aux modèles de sécurités [164]. L'enquête montre que les standards de sécurités les plus utilisés sont STRIDE, CVSS et CVE. De plus, elle souligne que parmi l'ensemble des propriétés de sécurité, seules la confidentialité, l'intégrité et la disponibilité sont couramment utilisées. Les observations réalisées lors de cette enquête concordent alors avec les préconisations proposées dans [159], pour être efficace une taxonomie doit se baser sur des outils et des propriétés standardisées.

Dans cette section, nous présenterons tout d'abord les standards utilisés pour la création de taxonomie de sécurité, puis nous nous focaliserons sur les taxonomies de vulnérabilité et les taxonomies d'attaquant.

2.1.2.1 Les standards : NVD, CWE, CVE et CVSS

NVD (*National Vulnerability Database*) [6] est une base de données de vulnérabilité gérée par l'institut états-unien des normes et de la technologie. Elle répertorie plus de 145 000 vulnérabilités nommées d'après la nomenclature CVE. La base de données spécifie les informations sous les titres : identifiant, date de sortie d'origine, date de dernière révision, source, description, score CVSS détaillé, références, type et logiciel vulnérable. NVD associe également une faiblesse aux vulnérabilités répertoriées en référençant le schéma de classification CWE.

CVSS (*Common Vulnerability Scoring System*) [3] est un système de métriques permettant de calculer un score entre 0 et 10 évaluant la dangerosité d'une vulnérabilité, 10 signifiant que la vulnérabilité est critique et 0 signifiant qu'elle est anodine. CVSS permet également de créer un vecteur présentant les caractéristiques de la vulnérabilité évaluée. Les mesures CVSS sont divisées en trois groupes : celles de base mesurent les caractéristiques intrinsèques et fondamentales d'une vulnérabilité, elles ne changent ni dans le temps et ni selon l'environnement. Les métriques temporelles mesurent les attributs qui changent dans le temps mais qui restent indépendantes de l'environnement. Les mesures environnementales mesurent les caractéristiques pertinentes et uniques à un environnement particulier.

CWE (*Common Weakness Enumeration*) [4] est une liste hiérarchique des faiblesses de sécurité des logiciels et du matériel. La structuration hiérarchique permet la définition de plusieurs niveaux d'abstraction, nommés du plus général au plus spécifique : pilier, classe, base et variante. CWE propose également de regrouper les faiblesses selon différentes vues. Par exemple la vue CWE-1003, utilisée par NVD, liste les faiblesses les plus courantes (124 éléments).

CVE (*Common Vulnerabilities and Exposures*) [1] est un dictionnaire énumérant les vulnérabilités de cybersécurité connues du public. Chaque CVE contient un numéro d'identification, une description et au moins une référence publique. L'identification s'effectue suivant le schéma CVE-AAAA-XXXX, avec AAAA l'année où la vulnérabilité a été rendue publique et XXXX est un nombre aléatoire. Le standard CVE propose également de regrouper les vulnérabilités selon différents types dérivés de CWE, [2] classe l'ensemble des entrées CVE de 1999 à 2019 selon 13 types de vulnérabilité :

— Dénis de service (*DoS*).

— Exécution de code (*Code Execution*).
— Débordement de tampon (*Overflow*).
— Corruption de mémoire (*Memory Corruption*).
— Injection SQL (*Sql Injection*).
— Script intersite (*XSS*).
— Traversement de répertoires (*Directory Traversal*).
— Fractionnement de réponses HTTP (*Http Response Splitting*).
— Contournement (*Bypass something*).
— Obtention d'informations (*Gain Information*).
— Obtention de privilège (*Gain Privileges*).
— Contrefaçon de demande intersite (*CSRF*).
— Inclusion de fichier (*File Inclusion*) .

Cette section présente une partie des nombreux standards utilisés en cybersécurité. Nous verrons dans la suite de ce manuscrit qu'il est nécessaire de manipuler plusieurs de ces standards afin de construire le point de vue d'un attaquant.

2.1.2.2 Les taxonomies de vulnérabilité

La cybersécurité couvre de nombreux domaines et les acteurs de ces domaines utilisent la notion de vulnérabilité à différents niveaux d'abstraction. Ce constat explique en partie pourquoi, à l'heure actuelle, il n'existe pas de taxonomie de vulnérabilité adoptée comme standard, au contraire, il en existe plusieurs centaines construites spécifiquement pour couvrir un domaine particulier [163]. Il est possible de regrouper l'ensemble des taxonomies existantes en deux grandes catégories, celle s'appuyant sur des standards et les autres.

Dans cette seconde catégorie, on retrouve les propositions les plus spécifiques telles que [72] qui présentent une taxonomie en quatre dimensions se focalisant sur le type de cible. Les auteurs décrivent une cible en différenciant les aspects matériels (disque, périphérique, etc.) et logiciels (système d'exploitation, application, services), chaque aspect étant décrit sur six niveaux d'abstraction. De même, Yongzheng and Xiaochun [168] proposent une taxonomie focalisée sur l'élévation de privilège, une vulnérabilité devenant alors uniquement un moyen d'obtenir de nouveaux droits et accès. Ce type d'approche permet de définir une taxonomie à la fois fournie et simple d'utilisation, cependant elle reste très restrictive et spécialisée à un seul type d'analyse.

Les taxonomies basées sur les standards proposent des approches plus généralistes. Par exemple, Singh et al. présentent dans [149] et [148] un cadre estimant un niveau de risque à partir d'une fréquence d'exploitation. Les auteurs calculent cette fréquence à partir des données de base et temporelles de CVSS et de la maturité d'une vulnérabilité, déterminées à partir des informations fournies par NVD. Dans cet exemple, les auteurs regroupent des informations provenant de deux standards distincts afin de calculer une nouvelle métrique, ici une fréquence. On retrouve cette approche dans [60], les auteurs font également la distinction entre système d'exploitation et application/service. Leur taxonomie comprend en plus les champs : causes de la vulnérabilité, techniques d'attaques exploitées et niveaux de menace. Ces trois champs sont dérivés des informations textuelles contenues dans les descriptions CVE de la vulnérabilité et de sa note CVSS. Par ailleurs, certaines recherches proposent d'améliorer un standard existant, par exemple Chen et al. [37] proposent un cadre de catégorisation qui transforme le dictionnaire CVE en un classificateur qui catégorise les CVE et qui évalue les tendances générales d'évolution des vulnérabilités. Pour

construire leur classificateur, les auteurs ont étudié plus de 20 000 descriptions de vulnérabilités via des algorithmes de machine learning, et ont identifié 11 causes, 11 impacts, 15 types de services vulnérables et 5 niveaux de sévérités. Ce type d'approche propose de déduire de nouvelles informations à partir de l'interconnexion de données provenant de différents standards.

Finalement, les taxonomies de vulnérabilité proposent, soit des métriques spécialisées à un domaine et définies manuellement, soit de nouvelles métriques construites grâce à l'association de données provenant de différents standards. Dans l'objectif d'obtenir une taxonomie la plus générique possible, nous pouvons identifier que, au minimum, il faudrait prendre en compte le type, les causes et les impacts des vulnérabilités.

2.1.2.3 Les taxonomies d'attaquants

Dans la littérature un attaquants est catégorisé, selon ses capacités [80], ses objectifs [160] ou encore un ensemble de caractéristiques. Il existe un grand nombre de solutions permettant de décrire un attaquant, tels que :

- Un niveau global : l'utilisation d'un niveau global permet de définir des attaquants plus ou moins compétents. Cette approche est utilisée, par exemple, dans le standard CVSS [3] avec la création, pour chaque vulnérabilité, d'une complexité d'attaque. Cette complexité représente le niveau de compétences nécessaires à un attaquant afin d'exploiter la vulnérabilité. Elle est évaluée selon trois niveaux : haut, moyen et bas. Par ailleurs, Paulauskas and Garsva [134] définissent un critère de temps moyen de compromission, calculé à partir des informations de NVD sur un ensemble de vulnérabilités et des caractéristiques d'un attaquant hypothétique. Les auteurs proposent de déterminer ce critère en se basant sur une distribution normale du niveau de compétence dans un groupe d'attaquant.
- La création de types d'attaquants : un attaquant est alors représenté par un ou plusieurs types. L'approche STIX [7] définit cinq types d'attaquants : innovateur, expert, praticien, novice et aspirant. Chaque type d'attaquants décrit un ensemble de capacités et d'aptitudes.
- La création de profils d'attaquants : un profil d'attaquants décrit l'ensemble des techniques et des outils utilisés par l'attaquant. Le concept de profil d'attaquants a été initialement proposé dans le projet "Know Your Enemies" [150]. Lenin et al. [110] construisent des profils d'attaquants en fonction d'un budget, de compétences et de temps alloué à l'attaque.
- La création d'un persona d'attaquants : un persona d'attaquants définit un ensemble d'objectifs, de motivations, d'attitudes et de compétences. Faily and Fléchais [54] utilisent ce concept afin identifier des menaces sur un système.
- La définition d'attributs d'attaquants : un attaquant est alors défini via un ensemble d'attributs qui peuvent être de différents types : booléen, numérique, etc. Heckman [75] proposent cinq variables (niveaux de compétence, ressources, intentions, motivations et probabilités) variant de 1 à 10, utilisées pour définir 9 types d'attaquants. Fraunholz et al. [59] proposent quatre attributs permettant de décrire un attaquant. Ces attributs sont : ressources, compétences, intentions et motivations. Les auteurs comparent également 15 approches de description d'attaquants et définissent une correspondance entre la notion de types et celle d'attributs, montrant alors qu'il est possible de passer d'une approche à l'autre.

Finalement, une taxonomie d'attaquants prend en compte trois types de critères : ce qu'un attaquant sait faire (capacités, compétences, aptitudes, etc.), ce qu'il veut faire (objectifs, intentions, motivations, etc.) et ce qu'il peut faire (ressources, budget, temps, accès,

droits, etc.). Tout comme les taxonomies de vulnérabilité, les informations sont en majorité dérivées des standards, seules les plus spécifiques à un domaine sont définies manuellement.

Les travaux [73, 84, 91] comparent tous un grand nombre de taxonomies de vulnérabilité et d'attaquants. Chacun de ces travaux propose une liste de critères permettant de correctement définir une taxonomie. Les critères communs aux différentes études sont : simplicité, réutilisabilité, conformité avec les standards et flexibilité. Il est à noter que la création d'une taxonomie est un problème difficile de caractérisation et d'identification de concepts. De nos jours, il n'existe pas de définition globalement acceptée et la convergence des différentes définitions sera longue voire impossible car trop de préoccupations sont à prendre en compte.

A notre sens, les définitions utilisées doivent être à la fois génériques et reposées sur une approche souple et ouverte afin d'intégrer les spécialisations nécessaires. Dans ce but nous présentons dans le chapitre 3 une approche permettant de décrire un attaquant et les vulnérabilités d'un système de manière flexible, simple, réutilisable et basée sur l'interconnexion de différents standards.

2.1.3 Analyses du comportement d'un attaquant

L'objectif des analyses de menaces, centrées sur le point de vue d'un attaquant, est d'anticiper le comportement de l'attaquant lors d'une attaque. De nombreux modèles graphiques de sécurité permettent de représenter ce comportement, Lallie et al. [107] en présentent 16, regroupés en deux grands groupes : les arbres d'attaque et les graphes d'attaque. Un arbre d'attaque [120] est un diagramme conceptuel des menaces d'un système et des attaques possibles pour atteindre ces menaces. Il est utilisé pour identifier différents chemins conduisant à un objectif prédéfini et pour construire une arborescence qui décrit comment une vulnérabilité aide un attaquant à atteindre son objectif. Un graphe d'attaque [135] représente une vue détaillée de la sécurité d'un système, il permet de déterminer si un attaquant peut atteindre des états finaux à partir d'un état initial. Il est composé de nœuds et de relations, les nœuds représentent des états et les relations font référence à des transitions entre différents états.

Les travaux [97, 81, 107] énumèrent et caractérisent l'ensemble des modèles graphiques de sécurité. Ils soulignent particulièrement la diversité des analyses permises par les arbres d'attaque.

LeMay et al. [109] proposent de générer et d'analyser des arbres d'attaque via l'approche ADVISE permettant d'imiter la façon dont un attaquant sélectionne sa prochaine étape d'attaque. Les auteurs proposent un modèle exécutable représentant le comportement de l'attaquant, ce modèle est construit à partir d'un profil d'adversaire et d'un graphe d'exécution des attaques. Un graphe d'exécution des attaques est créé par un expert en sécurité, il est composé de :

- Un ensemble d'étapes d'attaque : chaque étape représente un objectif ou une progression vers un objectif modifiant les accès ou les connaissances qu'a un profil adversaire. Une étape d'attaque comporte une pré-condition, une durée, un coût, une probabilité de succès, une probabilité de détection et une post-condition.
- Un ensemble d'accès : il permet de modéliser les accès de l'attaquant sur l'ensemble des constituants du système.
- Un ensemble d'élément connus : il modélise ce que l'attaquant connaît du système à un instant donné.
- Un ensemble de compétences pertinentes pour attaquer le système.

— Un ensemble d'objectifs pertinents pour un attaquant.

Un profil d'adversaire est composé de :

— Une situation initiale, représentant les droits et accès de l'attaquant avant l'attaque.

— Une fonction décrivant les compétences de l'attaquant. A chaque compétence pertinente pour attaquer le système est associé un booléen signifiant si l'attaquant maîtrise ou non la compétence.

— Une fonction de valeur d'objectif permettant d'associer à chaque objectif une valeur que lui attribue l'attaquant.

— Un ensemble d'attributs modélisant les préférences de l'attaquant en terme de coût, gain et chance d'être détecté. Ces attributs permettent de spécifier l'attaquant, par exemple un attaquant voulant être discret privilégiera une faible chance d'être détecté au détriment des gains et coûts éventuels.

— Une variable d'"horizon", elle représente la profondeur de l'arbre de prédiction utilisé pour sélectionner l'étape d'attaque.

L'approche ADVISE [109] permet de simuler pas à pas le comportement d'un attaquant en fonction de son profil et du graphe d'exécution des attaques associés au système.

Jajodia et al. [87] proposent une approche d'analyse de vulnérabilité sur un réseau nommé TVA (*Topological Vulnerability Analysis*). Leurs propositions reposent sur l'utilisation de *Nessus*, un outil de découverte de réseau et de détection de vulnérabilité. Les auteurs décrivent une vulnérabilité comme un tuple contenant une pré-condition et une post-condition représentant respectivement les conditions nécessaires à l'exécution d'une vulnérabilité et les effets de cette dernière sur le système. L'outil proposé s'articule autour de 3 composants :

— Une base de connaissance de vulnérabilité : définie manuellement, elle contient l'ensemble des vulnérabilités prisent en compte et décrites via des pré-conditions et de post-conditions.

— Une description réseau du système étudié : les auteurs utilisent directement les informations issues de Nessus afin de modéliser un réseau.

— Une spécification du scénario d'attaque : le scénario d'attaque définit la cible de l'attaque ainsi que la configuration du système et une situation initiale.

L'utilisation de ses trois composants permet aux auteurs de générer des chemins d'attaque.

Certaines approches proposent d'aller plus en profondeur dans l'analyse en proposant des solutions de vérification de modèle (*model checking*). Les auteurs de [144] s'appuient sur les techniques d'analyse formelle pour générer et analyser automatiquement un arbre d'attaque. Cependant des travaux [11, 132, 62] mettent en avant les limites de l'utilisation de la vérification de modèle, en particulier lors du passage à l'échelle pour des systèmes complexes. Diverses techniques sont alors proposées, utilisant des optimisations graphiques [132] ou encore de l'intelligence artificielle [62].

En plus des analyses effectuées durant la construction d'un arbre d'attaque, il est possible d'effectuer des analyses sur un arbre préexistant. Cela nécessite d'enrichir l'arbre d'attaque avec de nouvelles métriques de sécurité. Ces dernières peuvent représenter un coût, un profit, une chance de détection, une chance de réussite, une complexité, etc. La souplesse apportée par les arbres d'attaque permet de réutiliser un même arbre afin de mener différentes analyses. Par exemple, dans [111] les auteurs évaluent le temps moyen de compromission d'un système à partir d'un arbre d'attaque, ce temps est réutilisé dans [134] afin de calculer la dangerosité d'un type d'attaquant pour un système.

Finalement, la généricité apportée par les arbres d'attaque permet une grande diversité des analyses et une réutilisabilité des résultats. L'approche proposée dans [109] dissocie le

système de l'attaquant. Cette séparation permet de réutiliser la modélisation d'un attaquant sur différents systèmes et d'étudier la sécurité d'un même système par rapport à diverses stratégies d'attaque. Cependant l'approche ADVISE [109] se concentre sur des systèmes statiques et ne modélise pas le cycle de vie de ce dernier, de plus la création du graphe d'exécution nécessite l'intervention d'un expert à la fois du système modélisé et de l'approche ADVISE. Dans le chapitre 4 nous présentons une approche gardant la flexibilité et la richesse des approches existantes et qui prend en compte le cycle de vie du système et l'existence de nombreuses solutions d'analyse basés sur l'étude d'arbre d'attaque.

2.1.4 Axes de recherche : Modélisation et analyse du point de vue de l'attaquant

Parmi l'ensemble des domaines de la cybersécurité, nous avons couvert, dans cette section, les premières phases de l'analyse de la menace. En conséquence, notre état de l'art s'est focalisé sur la modélisation et l'analyse du point de vue d'un attaquant. La figure 2.1 donne la vision d'ensemble de ces travaux avec notamment la partie centrale de la figure qui montre la phase de modélisation comprenant à la fois la modélisation du système, des vulnérabilités et de l'attaquant.

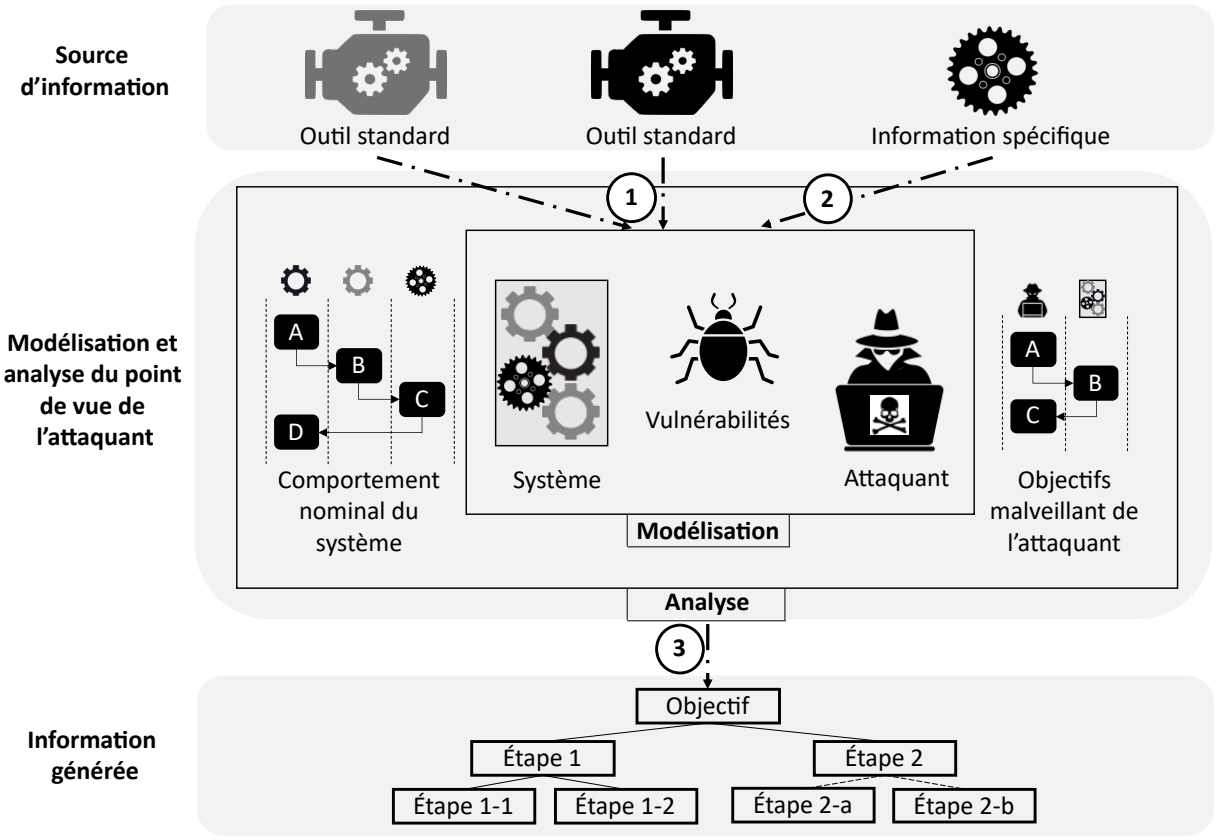


FIGURE 2.1 – Modélisation et analyse du point de vue de l'attaquant

Cet état de l'art fait ressortir notamment, à notre sens, les problématiques suivantes :

— Deux grandes étapes de modélisation sont principalement utilisées, ① la mise en commun d'informations provenant de standards et ② la modélisation manuelle d'informations spécifiques au contexte. Bien qu'un grand nombre de solutions reposent sur l'utilisation de standards ou proposent une modélisation spécifique à un domaine,

peu d'entre elles permettent d'effectuer les deux à la fois pour obtenir une modélisation holistique du point de vue d'un attaquant.

— Cette modélisation holistique nécessite une mise en relation d'informations et par conséquent une interopérabilité de différents formalismes issus des différents outils [10]. L'interopérabilité des modèles reste un sujet à part entière et c'est pourquoi nous discutons de cette problématique dans la section suivante de l'état de l'art (§2.2). De plus, nous présentons en section 2.4 la modélisation par rôle comme une solution d'interopérabilité non intrusive et adaptative.

— L'analyse du point de vue de l'attaquant nécessite des approches de simulation ou/et de vérification de modèle. Elle requiert, en plus des modèles du système, des vulnérabilités et de l'attaquant, la prise en compte du comportement du système et des objectifs de l'attaquant. Cette étape permet de générer ③ un ensemble d'informations hétérogènes tel que des arbres ou des graphes d'attaques. Ces informations peuvent être utilisées afin d'effectuer des analyses de sécurité approfondies utilisant des métriques spécifiques telles qu'une analyse de coût. Cependant, nous pouvons noter que dans la littérature la génération d'arbre d'attaque s'effectue généralement sans prendre en compte ni le comportement, ni l'évolution du système.

— La prise en compte des modifications apportées au système requiert une approche de construction de point de vue flexible et offrant des relations bidirectionnelles entre les points de vue et les modèles originels. Ce problème est abordé dans la section 2.3 avec la modélisation de point de vue et il est détaillé dans la section 2.4 via la modélisation par rôle, étudiée pour ses qualités d'abstraction et de flexibilité.

Ce manuscrit porte sur la modélisation de point de vue dans un contexte en évolution permanente, à savoir la modélisation et l'analyse de menaces. Pour se faire, nous présentons dans la suite de ce chapitre les problématiques de modélisation se rapportant à notre contexte.

2.2 Interopérabilité de modèles

Nous avons identifié dans la section précédente (§2.1) que l'interopérabilité des modèles et formalismes est crucial à notre approche. Dans cette section, nous définissons donc tout d'abord la notion d'interopérabilité, puis nous nous focalisons sur une des approches d'interopérabilité : la fédération de modèles. Par la suite, nous développons les notions de transformation bidirectionnelle et de synchronisation de modèles. Pour finir, nous concluons la section et discutons du futur des approches d'interopérabilité.

2.2.1 Définition de l'interopérabilité de modèles

L'interopérabilité est un sujet multi-facettes qui présente donc de nombreuses définitions. Par exemple, le standard IEEE-100 [50] donne les quatre définitions suivantes :

1. La capacité de deux ou plusieurs systèmes ou éléments à échanger des informations et à utiliser les informations échangées.
2. La capacité des unités d'équipement à travailler ensemble pour effectuer des fonctions utiles.
3. La capacité, promue mais non garantie par la conformité avec un ensemble de normes, qui permet à des équipements hétérogènes, généralement construits par différents fournisseurs, de fonctionner ensemble dans un réseau.

4. La capacité de deux ou plusieurs systèmes ou composants à échanger des informations dans un réseau hétérogène et à utiliser ces informations.

La notion d'interopérabilité peut être abordée sous différents angles. Chen et al. [35] identifient trois dimensions d'interopérabilité qu'ils nomment préoccupation, barrière et approche. La figure 2.2 schématise ces trois dimensions, chacune étant divisée en plusieurs catégories.

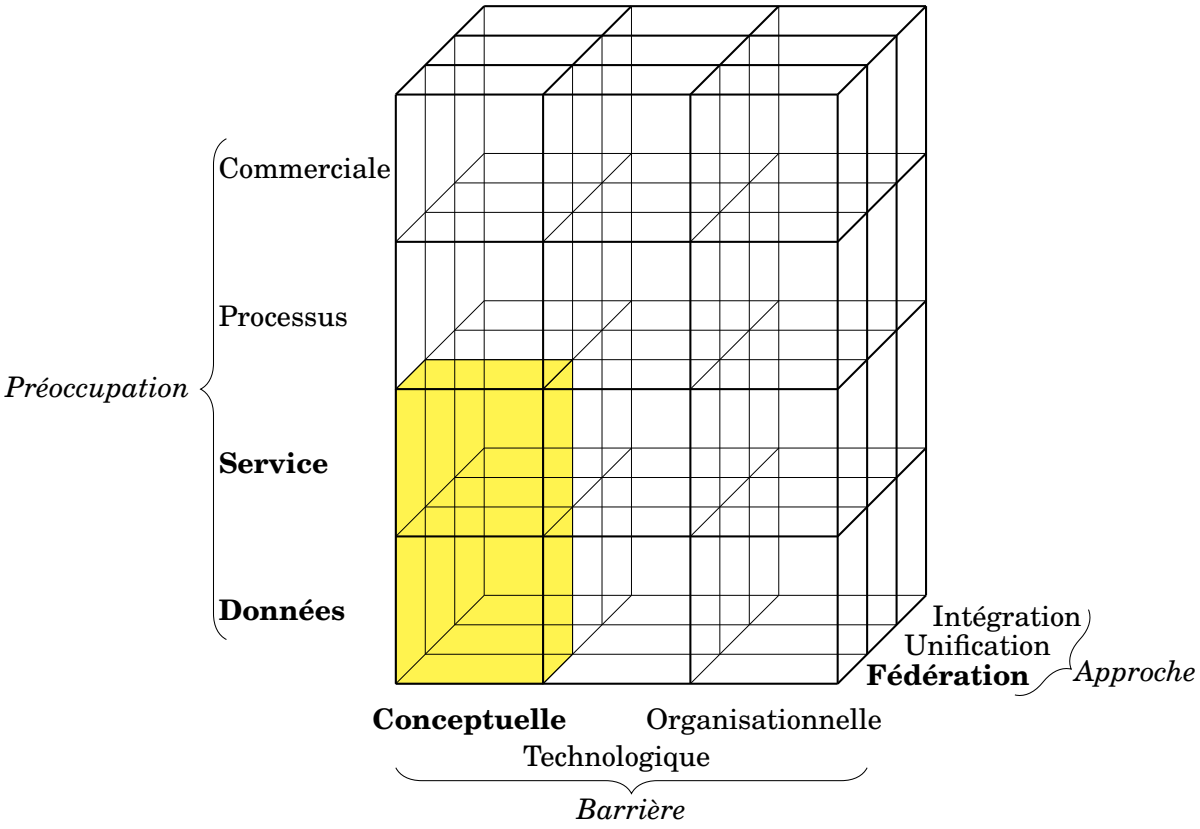


FIGURE 2.2 – Représentation tridimensionnelle des approches d'interopérabilité

Les préoccupations identifiées sont :

— Interopérabilité de données : approche visant à faire fonctionner ensemble différents modèles de données (hiérarchiques, relationnelles, etc.).

— Interopérabilité de service : approche visant à composer et faire fonctionner ensemble diverses applications (conçues et implémentées indépendamment) en résolvant les différences syntaxiques et sémantiques ainsi qu'en assurant les connexions à des données hétérogènes.

— Interopérabilité de processus : approche visant à faire fonctionner ensemble différents processus, un processus définissant dans quel ordre les services sont fournis en fonction des besoins du système.

— Interopérabilité commerciale : approche visant à faire collaborer différentes entreprises.

Dans cette approche, les barrières représentent les différentes incompatibilités ou décalages qui font obstacle au partage et à l'échange d'informations. Elles peuvent être conceptuelles, technologiques ou organisationnelles. Les barrières conceptuelles concernent les incompatibilités syntaxiques et sémantiques des informations. Celles technologiques existent

en raison de l'absence d'un ensemble de normes compatibles permettant l'utilisation de techniques informatiques hétérogènes pour partager et échanger des informations entre deux ou plusieurs systèmes. Les barrières organisationnelles représentent les incompatibilités de la structure et des techniques de gestion mises en œuvre dans différentes entreprises.

Les approches d'interopérabilité reflètent la technique utilisée pour mettre en place une interopérabilité entre différents systèmes. La norme ISO-14258 [85] définit trois manières pour lier entre eux différent modèles, ils peuvent être intégrés, unifiés ou fédérés.

L'**intégration** se base sur l'existence d'un modèle standard construit autour de l'union de tous les concepts des différents formalismes/langages intégrés [52]. Ce format commun n'est pas nécessairement une norme internationale mais doit être accepté par toutes les parties pour élaborer des modèles et construire des systèmes. La définition d'un tel métamodèle n'est pas envisageable pour tous les systèmes. Cette approche est envisageable uniquement si les formalismes intégrés ne varient pas au cours du temps, car toute création ou modification d'un élément intégré nécessite la redéfinition complète du standard.

L'**unification** nécessite la création d'un métamodèle qui fournit une structure de méta-niveau commune procurant un moyen d'établir des équivalences sémantiques. Ce métamodèle est généralement nommé métamodèle pivot. Après la définition de la correspondance, des transformations peuvent être appliquées entre tous les formalismes et le langage pivot, et aussi en sens inverse. Cependant, la création du langage pivot reste une tâche difficile. Si ce dernier est trop abstrait, cela conduit à une perte d'information, mais, s'il prend en compte toutes les propriétés des langages unifiés, cela peut produire un langage pivot trop large. Dans ce cas, l'unification devient alors équivalente à l'approche d'intégration. L'unification est à privilégier si le besoin d'interopérabilité concerne un nombre fixe d'outils hétérogènes, la prise en compte de nouvelles sources d'informations nécessitant de redéfinir le métamodèle pivot.

La **fédération de modèles** se concentre sur la modélisation de la sémantique des liens entre les concepts de langages hétérogènes. Cette approche se base sur la mise en relation dynamique des modèles fédérés plutôt que sur la définition d'un métamodèle pivot ou d'un standard. Dans le cadre de l'utilisation d'une approche fédérée, il n'y a aucun format commun. Les approches par fédération peuvent également utiliser des métamodèles de référence pour créer des liens entre différents modèles. La différence par rapport à l'unification est que ce métamodèle n'est pas prédéfini mais établi et évolue dynamiquement. Dans une approche fédérée, les transformations sont appliquées uniquement à la demande.

Le principal avantage de cette approche est la nette séparation entre l'élément source qui a son type d'origine et la modélisation sémantique relative au contexte de fédération. Un concept source peut avoir plusieurs sémantiques au sein d'une même fédération et plusieurs concepts source peuvent partager la même sémantique. Les approches de fédération sont valorisables si le besoin d'interopérabilité concerne des éléments hétérogènes et variant, la fédération permet d'ajouter, de modifier ou de supprimer des sources d'information dynamiquement. Comme présenté dans la section précédente (§2.1), la modélisation et l'analyse du point de vue d'un attaquant requièrent ce type d'interopérabilité appliqué à un ensemble de données ou de services hétérogènes conceptualisés dans des modèles sources. Finalement, comme le représentent les carrés colorés sur la figure 2.2, nous nous focaliserons dans ce manuscrit sur les approches d'interopérabilité mettant en œuvre une solution de fédération conceptuelle de données et de services.

Les approches d'unification et de fédération nécessitent l'échange d'informations hétérogènes. Dans un contexte de modélisation, cet échange est réalisé par des approches de transformation de modèles. De nombreux travaux proposent une formalisation de cette notion, par exemple Diskin et al. [46] et Feldmann et al. [55] définissent un cadre permettant

de spécifier les chevauchements entre des modèles partiels. Diskin et al. [46] proposent une solution ne nécessitant pas de fusion des métamodèles partiels en un métamodèle global, alors que Feldmann et al. [55] basent leurs approche sur la modélisation explicite des inconsistances entre les modèles sources. D'autres recherches telles que, [122] ou [26] proposent des solutions de transformation de modèles basées sur la création de vues spécifiques. Meier et al. [122] proposent d'assurer la cohérence des informations provenant de diverses sources via l'utilisation d'un modèle sous-jacent modélisé sous la forme de vues générées à la demande. Bruneliere et al. [26] proposent une approche permettant de définir des vues sur des modèles potentiellement hétérogènes. Les vues générées ne sont pas matérialisées mais redirigent toutes les demandes d'accès et de manipulation vers les modèles sur lesquelles elles sont appliquées.

Il existe une grande variété des approches de transformations de modèles. Cette variété est étudiée, par exemple, par Czarnecki and Helsen [39] qui fournissent un diagramme de caractéristiques représentant la variabilité de ces approches et dans une étude publiée en fin d'année 2018 [92] recensant plus de 60 outils de transformations de modèles.

Pour conclure, on peut noter que parmi le grand nombre d'approches existantes, la fédération de modèles est la plus adaptée à notre domaine d'étude.

2.2.2 Fédération de modèles

Dans cette section, nous cherchons à préciser la fédération et son utilisation.

Golra et al. [64] présentent la fédération de modèles comme un mécanisme d'interopérabilité de modèles hétérogènes. Ce mécanisme peut servir à développer de nouveaux points de vue en croisant plusieurs modèles existants ou encore à synchroniser ces mêmes modèles afin de concevoir un système. Les auteurs soulignent que les approches de fédération permettent aux outils interopérants de conserver leur propre paradigme, formalisme et outillage au sein de leur espace de modélisation privé. Cette distinction entre espace des outils et espace de fédération se retrouve également dans [116]. La fédération est ici définie comme un quintuple contenant, entre autres, un ensemble d'algorithmes indépendants des domaines sources et un ensemble d'algorithmes spécifiques aux domaines sources.

Les approches de fédération sont utilisées dans de nombreux domaines, par exemple pour la construction de chaînes d'outils [34], pour modéliser un ensemble d'activités décrites dans des modèles hétérogènes [147] ou encore pour assurer la collaboration de systèmes de systèmes [141]. Dans le domaine de la cybersécurité, la fédération peut être utilisée par exemple, pour interconnecter différents simulateurs Van Tran et al. [162].

Il existe divers techniques et outils permettant de créer un modèle d'information fédéré en reliant entre eux des modèles hétérogènes. Niemöller et al. proposent dans [128] l'utilisation d'un "*Glue-model*". Cette solution permet de créer un modèle d'information fédéré en reliant de manière flexible des éléments de modèles spécifiques à différents domaines. Afin d'inter-relier des éléments de divers modèles existants, le Glue-model utilise des relations abstraites. Les points de départ et d'arrivée de ces relations sont des références à des éléments d'autres modèles, ainsi le Glue-model lui-même est une ontologie de référence. L'approche proposée peut être décrite sur la base d'un formalisme mathématique avec une sémantique précise ce qui permet les analyses formelles. Les auteurs utilisent cette caractéristique afin de générer un réseau bayésien qui, une fois optimisé, permet d'effectuer diverses analyses.

D'autres auteurs proposent des solutions de fédération telles que Golra et al. [64] qui proposent une plate-forme collaborative générique pour la modélisation multiforme nommé "*Openflexo*". OpenFlexo permet la fédération de modèles dans plusieurs espaces technologiques et la prise en charge des modèles graphiques et textuels. L'objectif de cette solution de fédération est de créer des ponts entre différents paradigmes et de maintenir un lien

dynamique afin que des modèles hétérogènes puissent être synchronisés, permettant ainsi une cohérence globale de l'information. Openflexo permet également le développement de points de vue et de modèles de préoccupations croisées.

En pratique Openflexo repose sur la division de l'espace de modélisation en deux, un espace technologique et un espace conceptuel. L'espace technologique est l'environnement dans lequel sont définis des modèles hétérogènes utilisant leur propre paradigme, formalisme et outillage au sein de leur espace de modélisation privé. L'espace conceptuel est l'endroit où un ensemble de modèles peuvent être fédérés pour développer un nouveau point de vue / modèle de préoccupation croisée, appelé "modèle virtuel". Pour la conception d'un système complexe, un espace conceptuel peut être entouré de nombreux espaces technologiques. Chaque espace technologique contient alors des modèles qui suivent un paradigme de modélisation spécifique pour répondre à des préoccupations spécifiques. La connexion entre les espaces conceptuels et technologiques est bidirectionnelle. Un modèle virtuel dans l'espace conceptuel peut être mis à jour lorsqu'un modèle technologique correspondant évolue et inversement.

Les approches décrites ici, utilisant une ontologie de référence (Glue-model) ou une plate-forme spécialisée (OpenFlexo), illustrent différentes approches de fédération. Dans la suite de ce manuscrit nous détaillons les besoins en terme d'interopérabilité propres à la création de points de vue d'attaquant (§3.4.3). Nous présentons ensuite (chapitre 4) notre approche de fédération basée sur les rôles permettant la fédération de modèles hétérogènes et la mise en place de communication bi-directionnelle ainsi que, comme le permettent les Glue-model, l'exécution d'analyses basées sur les modèles fédérés.

2.2.3 Transformation bidirectionnelle et synchronisation de modèles

Pour clarifier le positionnement de la fédération par rapport aux transformations bidirectionnelles, nous explicitons ce mécanisme permettant de maintenir la cohérence entre diverses sources d'informations interconnectées [40]. Ce type de transformation est utilisé dans un grand nombre de domaines tels que la synchronisation des données répliquées dans différents formats [58], la conception d'interfaces utilisateurs [121], ou la mise à jour de vues [17]. Stevens [156] propose une formalisation de ce mécanisme et fait une distinction claire entre les transformations bidirectionnelles et les transformations bijectives. Une transformation bijective est bidirectionnelle mais l'inverse n'est pas toujours vrai, pour être bijective une transformation doit associer à chaque élément du modèle source un unique élément du modèle cible, et vice versa. Hidaka et al. [78] proposent un diagramme de fonctionnalités regroupant l'ensemble des caractéristiques des transformations bidirectionnelles en quatre macro-fonctionnalités :

— Espace technique : macro-fonctionnalité regroupant l'ensemble des caractéristiques propre à la représentation d'artefact. Cette catégorie caractérise les approches en fonction de la représentation d'artefact à laquelle elle se réfère.

— Correspondance : cette macro-fonctionnalité différencie les approches en fonction du type de relations qu'elle permet.

— Changements : macro-fonctionnalité facultative qui analyse le type de mises à jour autorisées.

— Exécution : cette macro-fonctionnalité structure les choix possibles dans les sémantiques d'exécutions.

Une transformation bidirectionnelle est donc caractérisée par la manière dont elle représente les éléments de modèles, par le type de relations qu'elle propose et par sa sémantique d'exécution. De plus, elle peut permettre, ou non, la mise à jour des éléments interconnectés.

La littérature compte de nombreuses approches permettant de mettre en œuvre des transformations bidirectionnelles. Certaines approches se fondent sur la création d'un modèle intermédiaire tel que *AMW2ATL* [117]. D'autres, tel que *Boomerang* [24], permettent de définir des transformations complexes à partir de combinaisons de transformations plus simples. Il existe également des approches utilisant les notions de vue [82] et d'autres utilisant celle de rôles [165, 143]. Werner and Aßmann [165] utilisent la notion de rôles afin de permettre la création de transformations durant l'exécution. La dynamicité des rôles permet l'ajout de nouvelles transformations pour des outils non pris en compte lors de la création du système [143]. De plus, les rôles permettent de synchroniser simplement des éléments de modèles en évitant la duplication de données [165]. La gestion de la synchronisation et la génération de transformations bidirectionnelles via les rôles sont discutées dans la section 2.4.2.

Les techniques de transformation bidirectionnelle sont utilisées pour la mise à jour de vues dans le domaine des bases de données. Lorsqu'une modification intervient sur un élément référencé par une vue, il existe deux grands types de solutions, la re-génération complète de la vue et la mise à jour partielle de la vue. La première solution permet d'éviter des problèmes de cohérence et simplifie l'implantation. Cependant elle est difficilement applicable sur des systèmes complexes car une re-génération totale entraîne un coup matériel et temporel importants. La seconde solution permet de réduire ce coup mais nécessite une synchronisation plus complexe. Par exemple, [43] proposent l'utilisation de modèles de décoration afin d'automatiser les mises à jours partielles de vue. D'autres approches, telles que [82] ou [119], permettent de modifier un modèle à partir d'une vue. Matsuda et al. [119] proposent la création de 2 vues distinctes, une vue de complétion permettant de mettre à jour un modèle et une vue de vérification permettant de s'adapter aux modifications apportées à ce dernier. Une vue peut alors être mise à jour en cas de changements survenant sur un modèle et elle peut être utilisée pour apporter des modifications sur ce dernier. La notion de vue et en particulier son utilisation pour la synchronisation et la transformation bidirectionnelle est détaillée dans la section 2.3.3

2.2.4 Le futur de l'interopérabilité

L'interopérabilité de modèles, les transformations bidirectionnelles et la synchronisation, font partie des défis majeurs de l'ingénierie dirigé par les modèles [29]. Une étude effectuée en 2019, [31] montre que les langages de transformation de modèles sont peu utilisés industriellement au profit des langages plus génériques tels que Java. Selon cette étude, l'une des raisons est que les langages traditionnels permettent la création de transformations plus complexes. Les auteurs concluent que les langages de transformations de modèles sont destinés à être utilisés dans des domaines spécifiques, les domaines plus généraux seraient alors couverts par d'autres types de solutions. Bruneliere et al. [28] proposent comme alternative au langage dédié l'utilisation de modèles de vues. Les auteurs mettent en avant l'hétérogénéité des sources d'information ainsi que la nécessité d'adapter les vues au changement survenant dans les modèles. Une des solutions mise en avant est donc la gestion de vue dynamique permettant de capter les changements du système et d'interagir avec ce dernier comme la fédération le propose. Nous envisageons cette approche dans la section suivante (§2.3).

2.3 Modélisation de point de vue

Dans la section 2.1 nous spécifions que le domaine d'étude de ce manuscrit est la modélisation et l'analyse de la menace à partir du point de vue d'un attaquant. Afin de pouvoir

correctement construire et modéliser le point de vue d'un attaquant, dans cette section, nous définissons puis formalisons les notions de vue et de point de vue. Par la suite, nous présentons puis discutons les techniques de modélisation par point de vue.

2.3.1 Définition des notions de vue et de point de vue

La notion de vue est antérieure à celle de point de vue, on la retrouve par exemple dans [145] où elle est définie comme "une abstraction simplificatrice d'une structure complexe" [1]. Les auteurs présentent la notion de vue comme une extension du paradigme de la programmation orientée objet permettant de définir plusieurs interfaces dans les classes d'objets, de contrôler la visibilité des variables d'instance et de permettre à plusieurs copies d'une variable d'instance d'exister dans une même instance d'objet. Une vue possède alors plusieurs caractéristiques lui permettant de :

— Cacher une partie des informations d'un objet.

— Partager l'identité d'un objet.

— Observer et manipuler un objet.

— Créer plusieurs vues sur le même objet.

Il est alors possible de créer, modifier ou détruire une variable ou un objet au travers d'une vue. Un ensemble de vues fixes permet, pour un utilisateur, d'observer et de manipuler des parties d'un système complexe et, pour un développeur, de faciliter le développement des outils dédiés à un type d'utilisateur.

La notion de point de vue est explicitée dans [57] comme "une combinaison des notions d'acteur, de source de connaissances, de rôle ou d'agent dans le processus de développement et de celles de vue ou de perspective qu'un acteur maintient. En termes de logiciel, il s'agit d'un objet géré localement et faiblement couplé qui contient une connaissance partielle du système et du domaine" [2].

Le cadre exposé dans [57] a été étendu dans [129]. Les auteurs décrivent l'utilisation de relations explicites entre des points de vue pour gérer le développement de logiciels possédant plusieurs perspectives. Ils mettent également en avant des problèmes de gestion de l'hétérogénéité pour les systèmes complexes et proposent une solution d'intégration basée sur la définition de plusieurs points de vue interconnectés.

De nombreux domaines utilisent la notion de point de vue. Par exemple, la gestion de l'architecture d'entreprise les utilise pour représenter la perception qu'un acteur a d'un système complexe, conformément à un ensemble de préoccupations. Dans ce contexte [151] définit les points de vue comme "des abstractions sur l'ensemble des modèles représentant l'architecture d'entreprise, chacun visant un type particulier de parties prenantes et répondant à un ensemble particulier de préoccupations". Les points de vue sont ici aussi un ensemble fini et fixe de modèles à partir desquels sont définis des vues individuelles. Cet ensemble de modèles est présenté dans [53]. Les auteurs de [151] proposent un cadre classifiant l'ensemble de ces modèles selon deux axes : les objectifs constitués des objectifs de conceptions, de décisions et d'informations et le contenu pouvant être un contenu détaillé, cohérent ou une vue d'ensemble.

La norme ISO-42010 [86] définit la notion de point de vue dans le domaine de l'ingénierie des systèmes et des logiciels, et en particulier pour la description d'architecture de systèmes. Les notions de vue et de point de vue y sont définis comme :

1. En version originale : A view is a simplifying abstraction of a complex structure.
2. En version originale : A viewpoint can be thought of as a combination of the idea of a “actor”, “knowledge source”, “role” or “agent” in the development process and the idea of a “view” or “perspective” which an actor maintains. In software terms it is a loosely coupled, locally managed object which encapsulates partial knowledge about the system and domain.

— Vue : produit de travail exprimant l'architecture d'un système conformément à un ensemble de préoccupations spécifiques à ce système.

— Point de vue : produit de travail établissant les conventions pour la construction, l'interprétation et l'utilisation de vues relativement aux préoccupations spécifiques du système.

La norme ISO-42010 ainsi que l'ensemble des autres définitions présentées dans cette section sont synthétisées dans les travaux de Bruneliere et al. [27]. Les auteurs affirment que dans un contexte industriel, les modèles doivent couvrir l'infrastructure matérielle, le déploiement, les scénarios d'utilisation ainsi que les propriétés non fonctionnelles du système. Ces domaines sont comparables à la classification proposée dans [151], l'infrastructure matérielle correspondant à un point de vue de conception détaillée, le déploiement à une décision vue d'ensemble, les scénarios d'utilisation à des informations détaillées et les propriétés non fonctionnelles du système à une information vue d'ensemble.

Comme le schématise la figure 2.3 Bruneliere et al. [27] proposent une définition des points de vue construits à partir de l'ensemble des caractéristiques, compositions et classifications présentées dans cette section. Pour ce faire ils définissent six termes en relation

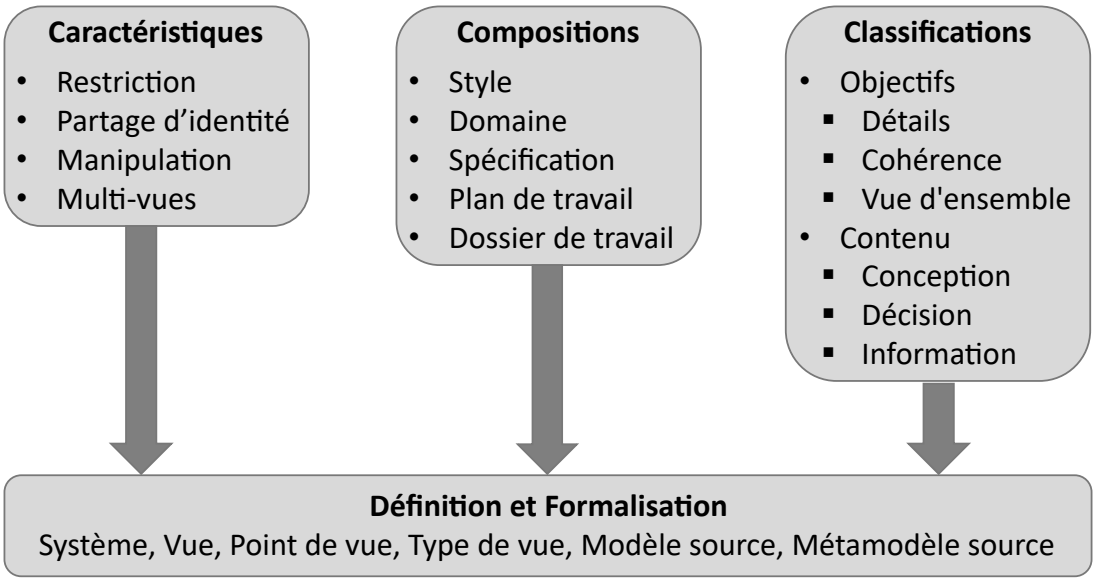


FIGURE 2.3 – Cadre de classification des points de vue

avec la notion de point de vue :

— *Système* : un *système* est une unité composée de plusieurs composants interdépendants, qui sont conçus et mis en œuvre par des ingénieurs. Un *système* englobe des logiciels, du matériel, des exigences, ainsi qu'un ensemble d'autres artefacts créés au cours du processus de développement. Dans un contexte de modélisation, un *système* est modélisé par un ensemble de *modèles sources* conformément à des *métamodèles sources*.

— *Point de vue* : Un *point de vue* est la description d'une combinaison, d'une agrégation, d'un partitionnement et / ou d'une restriction des préoccupations à partir desquelles les *systèmes* peuvent être observés. Dans un contexte de modélisation, il consiste en un ensemble de concepts issus d'un ou plusieurs métamodèles, éventuellement complétés par de nouvelles interconnexions et caractéristiques.

— *Type de vue* : un *type de vue* est un métamodèle qui décrit les types d'éléments qui peuvent apparaître dans une *vue*, c'est-à-dire le formalisme / langage réellement utilisé. Un élément dans un *type de vue* peut faire partie de l'un des *métamodèles sources*

ou peut être spécifiquement défini pour un *type de vue*. Un *type de vue* donné peut être pertinent pour plusieurs *points de vue*, et un *point de vue* définit généralement plusieurs *types de vue*.

— *Vue* : une *vue* est une représentation d'un système selon un *point de vue* donné. Dans un contexte de modélisation, il s'agit d'une instance d'un *type de vue* particulier et consiste en un ensemble d'éléments provenant d'un ou plusieurs *modèles sources*. Cette instance peut être complétée par de nouvelles interconnexions et des données supplémentaires, qui sont saisies manuellement et / ou calculées (généralement via une ou plusieurs transformations de modèles).

— *Métamodèle source* : un *métamodèle source* est un métamodèle référencé par un *type de vue*. Selon les approches, un *type de vue* donné peut avoir un ou plusieurs *métamodèles sources*.

— *Modèle source* : un *modèle source* est un modèle observé par une *vue*. Selon les approches et le *point de vue* correspondant (et les *types de vue* associés), une *vue* peut rassembler des éléments provenant d'un ou plusieurs *modèles sources*.

Un *système* est donc représenté par des *modèles sources* conformément à des *métamodèles sources*. Ces modèles sont observés par des *vues* conformes à des *types de vue*. Les *types de vues* référencent les *métamodèles sources* et sont associés à différents *points de vue*. Les *points de vue* sont construits d'après les *préoccupations* à partir desquelles les systèmes peuvent être observés.

Dans la suite de ce manuscrit nous nous référerons aux définitions de [27] pour qualifier les termes de vue, système, modèle source et métamodèle source. Dans la littérature, les notions de *point de vue* et *type de vue* sont généralement confondues sous l'appellation générique "point de vue". Dans ses travaux les plus récents [28] Bruneliere lui même ne fait plus la distinction entre type de vue et point de vue. Les notions de vue, type de vue et point de vue prêtent souvent à interprétation et à débat, c'est pourquoi nous souhaitons nous appuyer sur une formalisation précise de ces notions présentées dans la section 2.3.2.

2.3.2 Formalisation des notions de vue, type de vue et point de vue

La littérature autour des notions de vue et de points de vue contient plusieurs propositions de formalisation. Elles peuvent être formalisées mathématiquement [30], via une catégorisation par type [151] ou encore par des diagrammes de caractéristiques [63, 27]. Dans ce manuscrit, nous mettons en avant une approche de modélisation et d'analyse de points de vue, c'est pourquoi nous focalisons nos recherches sur les approches de formalisation mathématique et par caractéristiques.

2.3.2.1 Formalisation par caractéristiques

Bruneliere et al. [27] en s'inspirant des travaux de [63] proposent un diagramme, représenté en figure 2.4, regroupant l'ensemble des caractéristiques des approches de modélisation de points de vue. Ces caractéristiques sont divisées en trois catégories : structure, conception et exécution.

Caractéristiques relatives à la structure

Cette catégorie regroupe l'arité des modèles et métamodèles et les restrictions relatives à la construction des vues.

Une vue peut contenir des données provenant d'un ou de différents modèles sources, ce qui est décrit par les caractéristiques "Single Model" et "Multiple Models". De même, un type de vue peut être construit relativement à un ou plusieurs métamodèles sources, ce qui représenté par les caractéristiques "Single Metamodel" et "Multiple Metamodels".

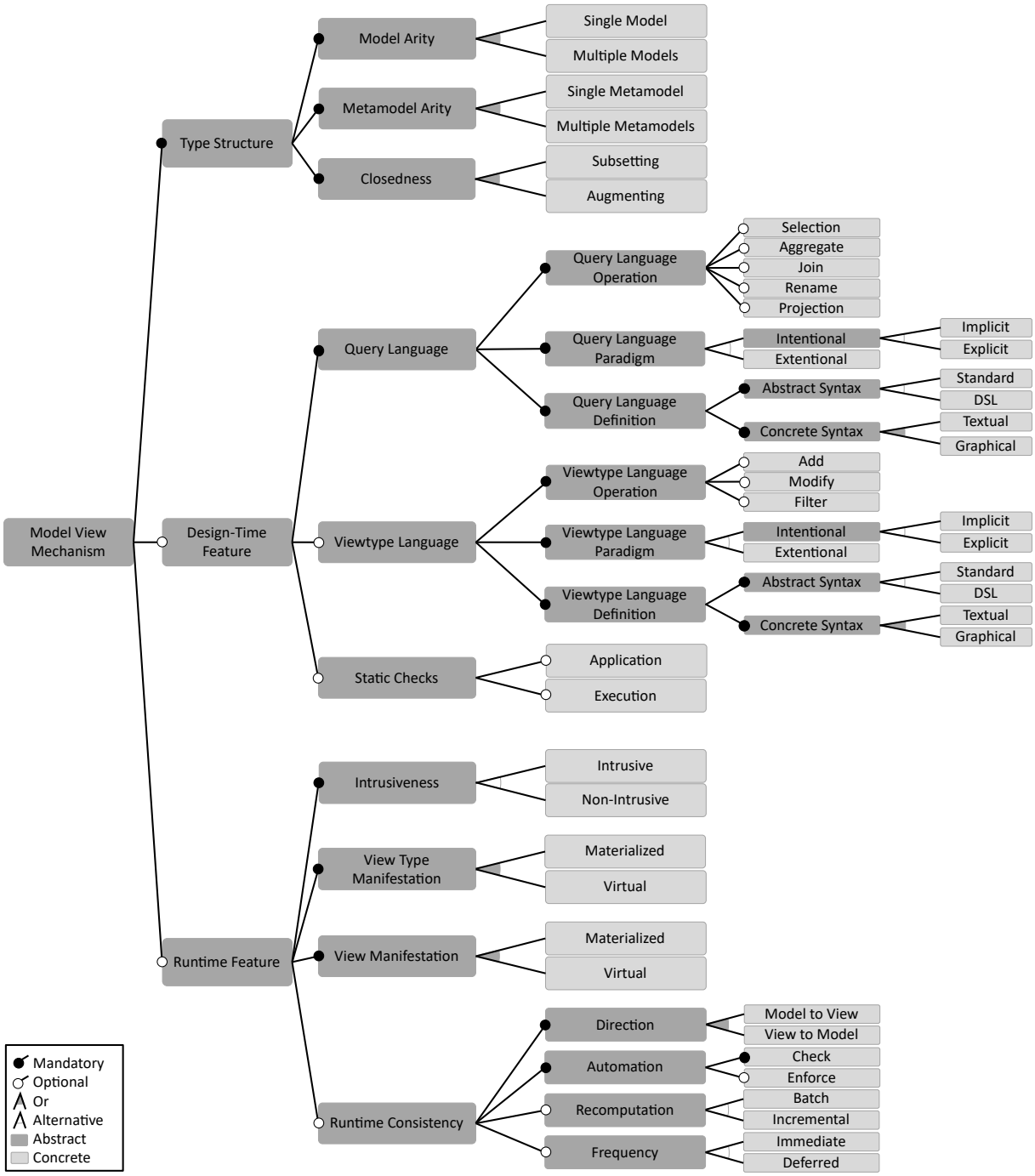


FIGURE 2.4 – Diagramme de caractéristiques des approches de modélisation de vue

Les restrictions de construction des vues sont exprimées par les caractéristiques "Closedness Subsetting" et "Closedness Augmenting". La première restreint la définition des vues en tant que sous-ensemble strict du ou des modèles sources. La seconde permet la construction de vues dans lesquelles le contenu provenant du ou des modèles de base peut être complété avec des données calculées ou entrées manuellement.

Caractéristiques relatives à la conception

La conception de vue et de type de vue est réalisée à l'aide de deux langages : un langage de requête ("Query Language") et de langage de type de vue ("Viewtype Language"). Le premier est utilisé pour remplir une vue avec les données attendues. Le second est utilisé

pour créer les types de vue correspondant à un point de vue. Chaque langage contient trois ensembles de caractéristiques relatives aux opérations, au paradigme et à la définition du langage.

Les opérations relatives au langage de requête se concentrent sur la façon dont les données des modèles sources sont sélectionnées et manipulées pour être intégrées aux vues. Ces opérations sont classifiées en deux groupes :

— Filtrage : le filtrage consiste à réduire la portée du ou des métamodèles sources en sélectionnant ou en projetant uniquement certains éléments des modèles sources dans les vues. Les caractéristiques relatives aux opérations de filtrage sont "Query Language Operation Selection" et "Query Language Operation Projection".

— Complétion : la complétion consiste à enrichir les données d'une vue en ajoutant des informations supplémentaires. Ces informations peuvent être dérivées d'éléments des modèles sources, via une agrégation ou un regroupement de données, ou être ajoutées manuellement. Les caractéristiques relatives aux opérations de complétion sont "Query Language Operation Aggregate", "Query Language Operation Join" et "Query Language Operation Rename".

Les opérations relatives au langage de type de vue permettent la création, la modification et le filtrage de concept. Ceux modifiés ou filtrés proviennent des métamodèles sources, ceux créés sont de nouveaux concepts définis manuellement. Ces opérations sont capturées par les caractéristiques "Viewtype Language Operation Add", "Viewtype Language Operation Modify" et "Viewtype Language Operation Filter".

La construction d'un type de vue ou d'une vue suit un paradigme pouvant être extensionnel, intentionnel explicite ou intentionnel implicite. Le paradigme est extensionnel si les éléments ou types d'éléments devant faire partie d'une vue sont répertoriés de manière exhaustive. Au contraire, il est intentionnel s'il existe un ensemble de propriétés et de conditions permettant de déduire si un élément (type d'élément) appartient ou non à une vue. La nuance entre explicite et implicite provient de l'expressivité ou non de ces propriétés.

Un langage possède une syntaxe abstraite et une syntaxe concrète. Une syntaxe abstraite peut être basée sur un standard (UML, SQL, OCL, etc.), ou être implémentée comme un langage de modélisation spécifique à un domaine (*DSL*). Une syntaxe concrète est textuelle et/ou graphique.

En plus des deux langages présentés ici, différents types de vérification statique peuvent être effectués au moment de la conception des types de vue et du peuplement de vues. Les auteurs de [27] identifient ici deux types de contrôle :

— Les contrôles d'applicabilité vérifiant que le calcul d'une vue ne retourne jamais un ensemble vide.

— Les contrôles d'exécutabilité vérifiant la cohérence d'une vue avec les contraintes définies au niveau du type de vue.

Caractéristiques relatives à l'exécution

Les caractéristiques d'exécution des vues et types de vue sont : intrusion, manifestation et cohérence d'exécution.

Un mécanisme de visualisation de modèles peut être intrusif ou non-intrusif. Il est intrusif si l'application d'une vue ou la définition d'un type de vue nécessite la modification de modèles ou de métamodèles de bases, dans le cas contraire il est non-intrusif.

La manifestation caractérise le mécanisme de mise en relation des modèles sources (métamodèles sources) et des vues (types de vue). Une vue peut faire référence à des données présentes dans un modèle source ou elle peut en contenir une copie. De même, un

Buckl et al. [30]	Bruneliere et al. [27]
Point de vue	Type de vue
Description d'une agrégation de préoccupation	Point de vue

Tableau 2.1 – Équivalences entre les appellations de Buckl et al. [30] et celles de Bruneliere et al. [27]

type de vue peut contenir des références ou des copies des concepts présents dans un métamodèle source. La manifestation est dite virtualisée en cas d'utilisation de références et matérialisée en cas de duplication des données.

La cohérence d'exécution regroupe les caractéristiques à prendre en compte lorsque les vues sont réellement calculées et gérées. Elles caractérisent la synchronisation entre les modèles sources et les vues. Cette dernière peut se faire des modèles vers les vues et/ou des vues vers les modèles.

Une synchronisation peut nécessiter de redéfinir l'ensemble d'une vue à chaque modification ou peut être incrémentale, dans ce cas, seule une partie de la vue est mise à jour. De plus, une synchronisation peut être déclenchée automatiquement après chaque modification d'une vue ou peut être déclenchée à des moments prédéfinis/ à la demande.

2.3.2.2 Formalisation mathématique

Dans cette section nous proposons une formalisation de l'approche de [27] en se basant sur les travaux de [30], les définitions de [27] permettant de dissocier clairement les aspects métamodèles (*type de vue*) des aspects préoccupations (*point de vue*). La relation entre les appellations de [30] et celle de [27] est présentée dans le tableau 2.1.

Dans cette partie nous présentons alors la formalisation mathématique des notions de point de vue, de type de vue et de vue.

Pour ce faire Buckl et al. [30] définit les notations suivantes :

— $\mathcal{A}$: l'ensemble des architectures possibles.

— $\mathcal{D}$: l'ensemble des entités et relations.

— $\mathcal{V}$: l'ensemble des types de vue.

— $\mathcal{C}$: l'ensemble des préoccupations.

— $\mathbb{P}(\mathcal{X})$: l'ensemble des parties de l'ensemble $\mathcal{X}$ [1]

Le terme architecture est défini comme "propriétés ou concepts fondamentaux d'un système dans son environnement incarné dans ses éléments, ses relations et les principes de sa conception et de son évolution" [86] [2].

Point de vue et préoccupation

Une préoccupation $c \in \mathcal{C}$ est définie formellement comme $c =< \mathcal{A}_c, \mathcal{D}_c, \mathcal{R}_c, f_c >$ avec :

— $\mathcal{A}_c \subseteq \mathcal{A}$ l'ensemble des architectures admissibles.

— $\mathcal{D}_c \subseteq \mathcal{D}$ l'ensemble des entités et relations admissibles.

— $\mathcal{R}_c \subseteq \bigcup_{n\in\mathbb{N}}(\mathcal{A}_c \to \mathbb{P}(\mathcal{D}_c^n))$ l'ensemble des concepts cohérents.

— $f_c : \mathcal{A}_c \to \mathbb{P}(\mathcal{D}_c)$ un filtre permettant d'identifier les concepts cohérents avec une architecture admissible.

1. Exemple : si $\mathcal{X} = \{1, 2\}$ alors $\mathbb{P}(\mathcal{X}) = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$
2. En version originale : fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.

La notation $[c]_{\equiv}$ représente une agrégation de préoccupations, formellement définie comme $[c]_{\equiv} := \{c' \in \mathcal{C} \mid c' \equiv c\}$. La relation d'équivalence $\equiv$ entre préoccupations est formalisée dans [30]. Bruneliere et al. [27] définissent un point de vue, entre autre, comme la description d'agrégation de préoccupations, un point de vue vp peut donc être formalisé comme une description syntaxiquement libre de $[c]_{\equiv}$.

Type de vue

Un type de vue $vt \in \mathcal{VT}$ est défini formellement comme $vt =< \mathcal{S}_{vt}, i_{vt}, n_{vt}, f_{vt} >$ avec :

— $\mathcal{S}_{vt}$ l'ensemble des syntaxes abstraites utilisées par le type de vue.

— $i_{vt} : \mathcal{S}_{vt} \rightarrow \mathcal{D}_{vt} \cup \mathcal{R}_{vt}$ une fonction de cartographie représentant une sémantique avec $\mathcal{D}_{vt} \subseteq \mathcal{D}$ l'ensemble des entités et relations utilisées par le type de vue.

— n_{vt} une fonction représentant la notation. n_{vt} n'est pas détaillée dans [30].

— $f_{vt} : \mathcal{A}_{vt} \rightarrow \mathbb{P}(\mathcal{D}_{vt})$ une fonction de filtrage déterminant les parties pertinentes d'une architecture.

Le lien entre type de vue et préoccupations est défini via une relation d'adressage $\sim \subseteq \mathcal{VT} \times [\mathcal{C}]_{\equiv}$ tel que :

$vt \sim [c]_{\equiv} \Leftrightarrow \mathcal{A}_c = dom(f_{vt}) \wedge \forall a \in \mathcal{A}_c : \{m(d) \mid d \in f_{vt}(a)\} = f_c(a)$.

avec $dom(f_{vt})$ le domaine de définition de la fonction f_{vt} et $m : \mathcal{D}_{vt} \cup \mathcal{R}_{vt} \rightarrow \mathcal{D}_c \cup \mathcal{R}_c$ la fonction d'adressage entre un type de vue et une préoccupation.

Vue

Une vue m_{vt} correspondant à un type de vue vt est définie comme l'application de vt sur une architecture $a \in \mathcal{A}$. Une vue est alors formalisée comme une fonction $m_{vt} : \mathcal{A}_{vt} \rightarrow \mathbb{P}(\mathcal{S}_{vt})$ telle que :

$\forall a \in \mathcal{A}_{vt} : \{i_{vt}(s) \mid s \in m_{vt}(a)\} \cap \mathcal{D}_{vt} = f_{vt}(a)$

Les formalisations présentées dans cette section servent de socle pour la formalisation de notre approche de modélisation de point de vue d'attaquant par les rôles (§4.1.2).

2.3.3 Utilisation de la modélisation de vues et de points de vue

Dans cette section nous présentons, dans une première partie, les domaines utilisant les points de vue, en nous focalisant plus particulièrement sur leur utilisation pour la modélisation de systèmes dynamiques. Dans une deuxième partie, nous comparons diverses solutions d'implantation de cette notion, et pour finir nous présentons un ensemble d'outils de modélisation de points de vue.

2.3.3.1 Domaines utilisant les notions de vue et de points de vue

La notion de vue apparaît dans le domaine de la gestion de bases de données avec les travaux de [42] et [17] qui utilisent le dynamisme des vues afin de modifier une sémantique ou des opérations en cours d'exécution. Dans le domaine de la modélisation les vues sont utilisées, d'après [145], pour définir plusieurs interfaces pour un même objet, contrôler la visibilité de variables ou encore permettre à plusieurs copies d'une même variable de coexister.

Une vue permet de faciliter l'interaction avec des systèmes complexes. Pour ce faire, dans certain domaines tels que la gestion d'architecture d'entreprise, les ingénieurs utilisent un ensemble fixe de points de vue prédéfini [53], l'ensemble des vues possibles est alors limité aux points de vue existants. Au contraire, dans d'autres domaines, tels que la transformation de modèles, il existe une infinité de vues possibles définies en fonction des modèles sources. Par exemple, [119] propose une approche de transformation bidirectionnelle basée sur la création de vues de complétion et de vues de vérification. Une vue

de complétion servant à interagir avec le système et une vue de vérification à observer le système.

Comme présenté dans 2.1 les vues et points de vue sont aussi utilisés dans les domaines de la cybersécurité. Par exemple [79] utilise le point de vue d'un attaquant et celui d'un défenseur pour estimer la cybersécurité des architectures d'entreprise. Les auteurs de [109] utilisent cette notion pour simuler le comportement d'un attaquant et identifier les points faibles de la défense d'un système.

Rolland et al. [140] proposent le concept de "*map*" et de "*guidelines*" pour gérer, via une modélisation multi-vues, la dynamicité d'un système . Une modélisation multi-vues représente un système grâce à des modèles hétérogènes sur lesquels peuvent s'appliquer différentes vues. Une *map* est une structure de navigation qui prend en charge la sélection dynamique d'intentions et de stratégies et une *guideline* aide à l'opérationnalisation des intentions. L'approche proposée permet de répondre à une situation en évolution en construisant des modèles de manière dynamique. Les modèles gèrent alors les situations au fur et à mesure qu'elles émergent et s'adaptent continuellement au changement du système. La notion de "*multimodel*" proche de celle de multi-vues est utilisée dans [46] pour décrire un ensemble de modèles définis dans différents métamodèles, chaque modèle capturant une vue spécifique du système. Les auteurs se focalisent sur la vérification de la cohérence globale des "*multimodels*" hétérogènes. Pour ce faire, ils recherchent des points de vues communs entre les métamodèles impliqués, projettent tous les métamodèles sur ces point de vues, fusionnent les projections et vérifient que le résultat respecte les contraintes spécifiées dans les points de vues. Les vues doivent alors s'adapter dynamiquement aux modifications des modèles sources afin de représenter le système en cours d'exécution. Les auteurs proposent également une solution de transformation bidirectionnelle basée sur l'utilisation des vues et définissent, pour chaque transformation, une transformation inverse.

Ces exemples illustrent deux types d'utilisation de la notion de points de vue pour capturer la dynamicité d'un système. Les vues générées doivent s'adapter à l'évolution du système modélisé par un ensemble de modèles hétérogènes. La prise en compte de cette évolution conduit à des problèmes d'interopérabilité, de traçabilité et de cohérence entre modèles [25]. La dynamicité d'une vue regroupe alors l'adaptation en temps réel au changement d'un système et la possibilité d'interagir sur ce dernier au travers des vues.

2.3.3.2 Implantation des notions de vue et de points de vue

Les notions de vue et de points de vue supportent diverses implantations, par exemple via la notion de sujet, de *"view programming"*, d'aspect, de rôle ou de profil UML.

La programmation orientée sujet [131] implémente les points de vue via une composition de modèles ou une fragmentation de modèles.

Les solutions de "*view programming*" [123] proposent de créer un ensemble de vues modélisant différents comportements qu'un objet peut adopter au cours du cycle de vue du système. Les vues sont alors associées en cours d'exécution à des éléments de modèles, cette association permet de modifier le comportement de l'objet.

Les solutions utilisant les aspects [138] offrent un mécanisme pour modulariser et composer des préoccupations transversales d'un système.

La modélisation par les rôles [13, 12] a pour objectif de séparer différentes préoccupations, les rôles permettent alors de considérer différentes vues sur un système.

Pour finir, *VUML* ("*View-based Unified Modelling Language*") [127] est un profil UML qui prend en charge la modélisation basée sur les vues pour une famille de langages de modélisation UML. La méthodologie proposée comprend la définition d'acteurs, qui possèdent

	P0S	**VP**	**Aspect**	**Rôle**	**VUML**
Analyse	-	-	-	xx	xxx
Conception	xxx	xxx	x	xx	xxx
Profilage et gestion des droits d'accès	xx	xxx	-	xxx	xxx
Dynamisme	-	xxx	x	x	xxx

Acronyme : POS : Programmation Orientée Sujet, VP : "View Programming"
Support : (-) Non supporté, (x) faiblement, (xx) modérément, (xxx) fortement

Tableau 2.2 – Tableau comparatif de différentes approches de modélisation de point de vue [126]

tous un unique point de vue. L'approche se compose d'une vue de base commune à tous les acteurs et de vues spécifiques à chaque acteur.

Ces différentes implantations sont comparées dans [51, 126] selon 9 critères :

- Utilisation durant le processus du développement. Ce critère se compose de trois phases :
  - Analyse : Description des différents phénomènes fonctionnels, technologiques et métiers d'un système. Cette phase sert à identifier l'ensemble des préoccupations d'un système.
  - Conception : Mise en œuvre d'un ensemble d'activités et de concepts qui permettent la mise au point d'un système.
  - Implantation : Traduction des concepts de conception dans un langage de programmation.
- Réutilisabilité : Ensemble des solutions permettant la réutilisation d'analyses, de concepts et/ou de codes sources.
- Intelligibilité du code : Capacité de compréhension de codes existants et nécessitant une séparation des préoccupations.
- Testabilité : Ensemble des tests unitaires, fonctionnels et structurels.
- Profilage et gestion des droits d'accès : Manière d'assurer la pertinence des informations et la gestion des droits d'accès d'utilisateurs d'un système.
- Dynamisme : Prise en compte en temps-réel de l'évolution du système.
- Multiplicité : Gestion des accès simultanés à un même objet.
- Identité et Intégrité : Différentiation entre une vue et l'objet observé.
- Maintenabilité : Prise en charge de l'évolution et de la maintenance du système.

Dans ce manuscrit nous nous concentrons sur la modélisation et l'analyse du point de vue d'un attaquant. Nous avons montré dans 2.1 que, dans un contexte de cybersécurité, il est nécessaire de prendre en compte les préoccupations d'un attaquant ainsi que l'évolution de ses accès sur le système attaqué. C'est pourquoi les critères les plus pertinents pour ce manuscrit sont "Analyse", "Conception" "Profilage et gestion des droits d'accès" et "Dynamisme". Le tableau 2.2 synthétise les résultats présentés dans [126] en se concentrant uniquement sur les critères les plus pertinents à notre contexte.

Ce tableau permet d'illustrer le fait que, parmi les approches comparées dans [126], seules VUML et les rôles supportent les quatre critères qualifiés de pertinents pour la modélisation et l'analyse du point de vue d'un attaquant. Cependant, bien que VUML couvre parfaitement chacun des critères, cette approche n'est pas pertinente pour la modélisation de la perception d'un attaquant. En effet, VUML est un profil UML et, à ce titre, est fortement dépendante du métamodèle d'UML. Comme présenté dans la section 2.1,

les perceptions d'un attaquant couvrent un ensemble hétérogène de domaines. L'approche adoptée doit donc être flexible et adaptable. C'est pourquoi, dans la suite de ce manuscrit, nous valoriserons les approches de modélisation par rôles.

2.3.3.3 Présentation d'outils de modélisation de vue et de points de vue

Il existe de nombreuses solutions permettant de modéliser des vues et des points de vue. Les auteurs de [27] ont réalisé une étude sur un ensemble d'articles présentant les termes "model" et "view" ; suite à différents filtres, ils ont sélectionné et analysé 16 solutions de modélisation de points de vue différentes. L'étude, résumée dans la figure 2.3 se base sur le diagramme de caractéristiques présenté en section 2.3.2.

Comme nous le verrons dans le chapitre 3, quatre de ces approches ont un interêt particulier pour la modélisation de point de vue d'attaquant : EMF Views, Kitalpha, VIATRA viewers et OpenFlexo.

EMF Views [26] fournit des capacités pour spécifier des vues sur des modèles potentiellement conformes à différents métamodèles. Les vues obtenues sont principalement en lecture seule, seul un sous-ensemble limité de modifications peut être propagé aux modèles. Dans *EMF View*, un point de vue est un métamodèle virtuel qui ne contient que des références vers des éléments provenant de métamodèles sources. De même une vue est un modèle virtuel référant des éléments de modèles sources. Les liens vers les éléments de différents métamodèles / modèles sont stockés dans un modèle de tissage séparé.

Kitalpha [108] est l'environnement pour développer et exécuter des vues sur des modèles afin de décrire l'architecture de systèmes. Il est basé sur la norme ISO/IEC 42010 [86]. La notion de vue couvre les points suivants : syntaxe abstraite, notations (telles que les icônes), syntaxe concrète (textuelle et graphique), règles (par exemple vérification et transformation), services et outils. L'environnement de développement se concentre sur la réutilisation des vues en fournissant une solution d'héritage et d'agrégation des points de vue.

VIATRA Viewers [43] est une extension de *EMF Inc- Query* [161] qui est un cadre pour effectuer des requêtes de modèles incrémentales ("incremental model querying"). *VIATRA Viewers* crée des règles de dérivation qui sont définies via des annotations sur les modèles de requêtes. Des types de vues sont définis via ces annotations, et les modèles sources sont utilisés pour remplir les vues grâce au support d'*EMF IncQuery*. En plus des vues, des modèles de trace sont calculés entre les vues et les modèles sources, ils sont utilisés comme support pour la synchronisation.

OpenFlexo [64] est une solution générique permettant d'assembler et de relier, sans duplication, des données provenant de différentes sources. Il contient de nombreux composants dont "Viewpoint Modeler" et "ViewEditor". Le premier sert à spécifier des points de vue et indique comment mélanger différents types de données. Le second fournit des capacités de visualisation et d'édition des vues, créées à partir des points de vue précédemment spécifiés. *OpenFlexo* permet de propager des modifications d'un modèle source vers un autre via une vue commune.

La caractérisation des solutions de modélisations de points de vue présentée dans cette section est utilisée dans le chapitre 3 afin d'identifier le ou les outils, s'ils existent, adaptés à la modélisation de points de vue d'attaquant.

2.3.4 Discussions sur la modélisation de vues et de points de vue

Bien qu'il existe un standard (ISO 40200 [86]) définissant les notions de vue et de points de vues, il n'existe pas de consensus quant à leur utilisation. Dans les sections 2.3.3, nous montrons que ces notions sont utilisées dans de nombreux domaines, cependant, chacun

				Cicchetti	EMF Facet	EMF Profiles	EMF Views	Epsilon Merge	Epsilon Decoration	FacadeMetamodel	Kitalpha	ModelJoin	OpenFlexo	OSM	Sirius	TGGmv	TGGmvv	VIATRA Viewers	VUML
Type Structure																			
Model Arity	Single Model			✓	✓	✓			✓	✓				✓		✓	✓		
	Multiple Models						✓	✓			✓	✓	✓		✓			✓	✓
MM Arity	Single MM			✓	✓	✓			✓	✓				✓		✓	✓		✓
	Multiple MMs						✓	✓			✓	✓	✓		✓			✓	
Closedness	Subsetting			✓			✓	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓
	Augmentings				✓	✓	✓		✓	✓	✓	✓			✓	✓	✓	✓	
Design Time Features														∅					
QueryLanguage	QL Operations	Selecting		✓	✓		✓				✓	✓	✓		✓	✓	✓	✓	✓
		Projecting		✓	✓		✓				✓	✓	✓		✓	✓	✓	✓	✓
		Joining					✓	✓			✓	✓	✓		✓	✓	✓	✓	✓
		Rename					✓	✓			✓	✓	✓		✓	✓	✓	✓	✓
		Aggregation			✓		✓	✓			✓	✓	✓		✓	✓	✓	✓	
	QL Paradigm	Extensional			✓														✓
		Intensional	Implicit							✓									
			Explicit	✓	✓		✓	✓			✓	✓	✓		✓	✓	✓	✓	
	QL Definition	Abs. Syntax	Standard		✓										✓				✓
			DSL	✓			✓	✓			✓	✓	✓		✓	✓	✓	✓	
		Con. Syntax	Textual		✓		✓	✓			✓	✓	✓		✓			✓	✓
			Graphical	w									✓			✓	✓		
VR Language	VTL Operations	Add			✓	✓	✓		✓		✓		✓			✓	✓	✓	
		Modify					✓				✓	✓	✓						
		Filter					✓				✓	✓	✓						
	VTL Paradigm	Extensional		✓	✓	✓	✓	✓	✓		✓		✓		✓	✓	✓	✓	✓
		Intensional	Implicit																
			Explicit							✓		✓							
	VTL Definition	Abs. Syntax	Standard	✓		✓						✓				✓	✓		✓
			DSL		✓		✓	✓	✓	✓	✓		✓		✓			✓	
		Con. Syntax	Textual				✓	✓			✓	✓	✓					✓	
			Graphical	w	✓	✓			✓	✓			✓		✓	✓	✓		✓
Static Checks	Application					✓			✓	✓		✓	✓						
	Execution			✓			✓	✓		✓	✓		✓		✓	✓	✓	✓	✓
Runtime Features																			
Consistency	Direction	Model –>View		✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
		View –>Model				✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Automation	Check		✓								✓		✓					✓
		Enforce				✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
	Recomputation	Batch			✓		✓	✓			✓	✓			✓	✓	✓		✓
		Incremental		✓		✓			✓	✓			✓	✓		✓		✓	
	Frequency	Immediate		✓		✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	
		Deferred			✓			✓				✓							✓
VT Manifest.	Materialized			✓		∅		✓	∅	✓	✓	✓	✓	✓		✓	✓	✓	✓
	Virtual				✓	∅	✓		∅						✓				
View Manifest.	Materialized			✓		✓		✓	✓		✓	✓			✓	✓			✓
	Virtual				✓		✓			✓			✓	✓			✓	✓	
Intrusiveness	Intrusive													✓			✓		✓
	Non-Intrusive			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓	

✓ : Caractéristique supportée, ∅ : Non applicable, w : Dépend d'un *wizard*.

Tableau 2.3 – Comparaison des approches de modélisation par vues [27]

de ces domaines utilise une définition différente de la notion (voir section 2.3.1). Ces différentes définitions restent cependant cohérentes entre elles et peuvent être représentées via un diagramme de caractéristiques comme illustré dans la section 2.3.2. De plus, bien que l'on compte de nombreux outils de modélisation de points de vue, il n'existe pas d'implantation standardisée des vues et points de vues.

Cependant ces limitations sont aussi à l'origine de la flexibilité des points de vue. Ils peuvent être utilisés dans des domaines très variés tel que l'industrie [53] ou la cybersécurité [109]. Ils est également possible de définir des vues permettant d'interagir avec les modèles sources, ou encore des vues s'adaptant dynamiquement à l'évolution d'un système.

Il nous semble donc que l'ensemble de ces qualités font de la modélisation par les vues une approche adaptée pour capturer la perception d'un attaquant sur un système modélisé par un ensemble de modèles hétérogènes.

Cependant, la majorité des solutions de modélisation de points de vue considère leur définition comme statique. Cette importante limitation est rédhibitoire à nos yeux, compte tenu de l'évolutivité importante de la perception d'un attaquant sur son système cible et ce, au niveau même de la définition des concepts ciblés.

C'est pourquoi il nous a semblé incontournable de nous intéresser aux approches de modélisation basées sur les rôles. En effet, une des grandes caractéristiques des rôles est de considérer la définition du points de vue comme flexible et adaptable dynamiquement. La modélisation par les rôles, qui représente la pierre angulaire de notre approche, est définie dans la section suivante (§2.4).

2.4 Modélisation par les rôles

La notion de rôle est utilisée pour spécialiser ou caractériser un individu. Le terme "rôle" apparaît dans de nombreux domaines tel que le contrôle d'accès [56, 65, 93], la représentation de connaissances [114], la modélisation conceptuelle [76], la modélisation des données [70, 112, 89] ou encore la conception orientée objet [16, 77].

La définition du terme rôle remonte à 1647, Lodwick [113] le présente comme "un nom par lequel une chose est nommée et distinguée, mais pas continuellement, seulement pour l'instant présent, en relation avec une action faite ou subie [1]. On retrouve dans cette définition les notions qui seront centrales au concept de rôle : la dynamicité et la dépendance à un contexte/une relation. Un rôle est alors associé à quelque chose de manière temporaire et permet de distinguer cette chose en fonction de son comportement et du contexte dans lequel elle évolue.

La notion de rôle est présente depuis les débuts de la conception orientée objet, Chen [36] présente le modèle Entité-Relation et définit la notion de rôle d'une entité dans une relation comme la fonction que l'entité joue pour cette relation.

La modélisation par les rôles (*role modeling*) est apparue en 1977 avec les travaux de Bachman and Daya [15]. La modélisation par les rôles y est définie comme un moyen de modéliser des domaines complexes et dynamiques. Les auteurs introduisent également la notion de segment de rôle comme un concept servant à regrouper et nommer les propriétés concernant l'existence d'un rôle. Ces propriétés font référence à la collaboration d'un rôle avec d'autres objets et à la dépendance à un contexte.

Entre les travaux de Chen et ceux de Bachman and Daya il n'y pas de sémantique commune du concept de rôles, le premier le définissant comme une fonction d'une relation et les seconds comme un concept regroupant des objets.

1. En version originale : "A name by which a thing is named and distinguished, but not continually, only for the present, in relation to some action done or suffered"

C'est pourquoi, une première section présente les différentes approches et formalisation de la notion de rôle nous permettant, par la suite, de positionner notre travail par rapport aux travaux existants. Les deux dernières sections se focalisent sur l'utilisation des rôles et l'identification de leurs limites.

2.4.1 Formalisation du concept de rôle

Il existe diverses définitions de la notion de rôle dans la littérature. Bachman and Daya [15] la définissent comme "un modèle de comportement défini qui peut être assumé par des entités de nature différente".

Kristensen and Østerbye [99] définissent eux la notion de rôle comme "un ensemble de propriétés qui sont importantes pour qu'un objet puisse se comporter d'une certaine manière attendue par un ensemble d'autres objets".

Loebe [114] spécialise la notion de rôle en *rôle abstrait* et *rôle social*. Un *rôle abstrait* est un moyen de visualiser quelque chose dans un contexte particulier et un rôle *social étant* est "un objet social complexe pour lequel la relation avec un objet est primordiale" [114].

Ces quelques définitions illustrent le problème relatif à la définition de la notion de rôle : l'absence de compréhension commune. Ce problème est connu depuis près de 20 ans et de nombreux travaux le solutionnent en proposant une formalisation de la notion [98, 95, 94, 154, 67, 33, 101].

Kniesel [95] propose de formaliser la notion de rôle via un tableau à quatre dimensions. Ces dimensions représentent le fait qu'un même type d'objet peut jouer différents types de rôles, le couplage simultané d'un même objet avec plusieurs rôles, l'association dynamique d'un objet avec un rôle et la visibilité par un objet des propriétés d'un rôle. Cette formalisation est reprise dans les travaux de Graversen [67].

Pour [98], un rôle spécifie et implémente toutes les propriétés qui relèvent de la même perspective qu'un objet. Les rôles peuvent être liés à des objets ou à d'autres rôles. Ils peuvent être transférés d'un objet à un autre, mais ils ne peuvent pas exister par eux-mêmes. Le propriétaire d'un rôle est un objet avec un attribut du type de rôle. Les propriétés d'un rôle sont acquises par un objet une fois qu'il assume le rôle et perdues une fois qu'il l'abandonne ; le rôle est alors vacant et prêt à être joué par un autre objet. L'auteur de [98] propose 6 caractéristiques propres à la notion de rôle : La visibilité, la dépendance à un objet, l'identité, la dynamicité, la multiplicité et l'abstractivité. Ces caractéristiques sont reprises dans les travaux de Steimann [155, 152, 154].

Dans [94] les auteurs comparent différents utilisations des rôle dans la modélisation orientée objet. Pour effectuer cette comparaison, ils proposent un ensemble de 12 critères d'évaluation. Ces critères se retrouvent dans les travaux de Steimann [155, 152, 154].

Steimann [154] étudie un grand nombre de formalisation de la notion de rôle dont [98] et [94]. Il conclut qu'un rôle peut être modélisé comme un nom de relation, une spécialisation, un super-type ou encore une instance complémentaire. Steimann démontre alors qu'il n'existe pas une unique définition de la notion de rôle et propose un ensemble de caractéristiques permettant de caractériser une approche de modélisation par les rôles [154, 152]. Les travaux de Steimann servent de référence pour [67] et [101] formalisant chacun la notion de rôle via un diagramme de caractéristiques.

Kühn soutient en 2017 sa thèse [100] dans laquelle il propose un diagramme regroupant l'ensemble des caractéristiques de Steimann enrichi de 13 nouvelles décrivant entre autres la notion de compartiment. Les travaux de Kühn reposent en partie sur les travaux de Graversen [67].

Cabot and Raventós [33] présentent 5 patrons de conception ainsi de 12 caractéristiques adaptées de celles de Steimann permettant de représenter l'ensemble des approches de modélisation par les rôles [33].

Toutes les approches de formalisation du concept de rôle exposées ici ont été reprises directement ou indirectement par Kühn mise à part les travaux de Cabot and Raventós [33]. Dans la suite de cette section, nous détaillons la formalisation par patrons de Cabot and Raventós [32, 33] et celle par caractéristiques de Kühn [101, 100, 106]. De plus, nous présentons, dans une troisième partie, une formalisation mathématique de la notion de rôle. Dans la suite de ce manuscrit, les entités de natures différentes qui peuvent assumer / jouer un rôle sont appelées des "*players*".

2.4.1.1 Formalisation par patrons

Dans [32, 33], les auteurs soulignent l'absence de définition générale et cohérente de la notion de rôle et rappellent que le concept de rôle est utilisé avec de nombreuses significations : classe dynamique, aspect, perspective, interface ou mode. Ils identifient alors une sémantique commune à différents modèles de rôles provenant de la littérature et présentent un ensemble de patrons de modélisation pour la notion de rôle qui incluent à la fois les aspects statiques et dynamiques des rôles. Cabot and Raventós proposent 5 patrons de modélisation :

— *Roles as Participant Names Pattern* : Un rôle est ici un nom d'une relation entre deux concepts.

— *Roles as Subtypes Pattern* : Un rôle est une spécification / un sous type du *player*.

— *Roles as Interfaces Pattern* : Un rôle est une interface du *player*.

— *Roles as Reified Entity Types Pattern* : Un rôle est représenté comme un type d'entité réifié d'une relation entre le *player* et un autre concept

— *Roles as Entity Types Pattern* : Un rôle est un concept indépendant et est relié au *player* par une relation de type *RoleOf*.

Il est possible d'identifier deux grandes approches dans les patrons proposés ici : l'une définissant la notion de rôle en tant que relation (*Roles as Participant Names Pattern*) et l'autre la définissant en tant qu'objet (*Roles as Subtypes Pattern*, *Roles as Interfaces Pattern*, *Roles as Reified Entity Types Pattern* et *Roles as Entity Types Pattern*).

Le patron *Roles as Entity Types Pattern* est celui couvrant le maximum de caractéristiques de Steimann [32]. La figure 2.5 présente un exemple de création d'un rôle d'après le patron *Roles as Entity Types Pattern*. Le concept d'*Employé* est ici un rôle joué par le

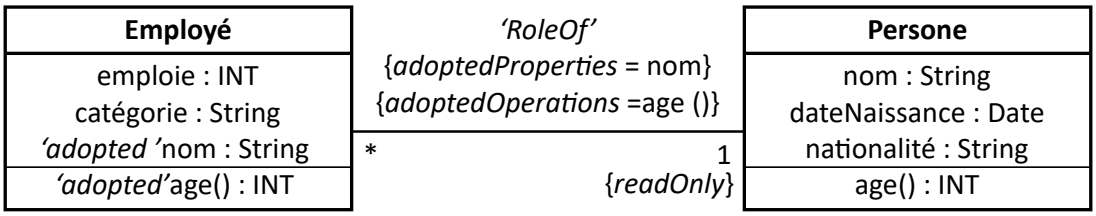


FIGURE 2.5 – Exemple implémentant le patron *Roles as Entity Types Pattern*

concept de *Personne* La relation reliant le rôle au concept se nomme *"RoleOf"*. Cette relation explicite la liste des propriétés et des opérations adaptées dans le rôle. Le rôle adapte ici une propriété et une opération du concept de *Personne*, la propriété *"nom"* et l'opération *"age()"*. La relation *RoleOf* est en lecture seule, ce qui signifie qu'un rôle peut référencer des propriétés ou des opérations d'un concept mais qu'il ne peut pas les modifier.

L'implantation du patron *Roles as Entity Types Pattern* proposée dans [33] offre des avantages :

— Implantation d'un grand nombre de caractéristiques de Steimann, à savoir les caractéristiques : 1, 2, 3, 4, 5, 6, 8, 11, 12, 13 et 14/15.

— Un rôle est considéré comme un concept à part entière, il est donc possible de créer tout type de relation entre les rôles : référence, contenance, héritage, etc.

— Possibilité de contraindre les relations entre les rôles ou les relations *"RoleOf"* avec des contraintes OCL (*Object Constraint Language*).

— L'implantation des rôles ne demande pas de nouveaux concepts mais se base sur des notions existantes à savoir les entités et les contraintes.

Cependant cette approche souffre de quelques limitations :

— Nécessite de modifier les *players* en ajoutant les opérations *addRole* et *removeRole*.

— N'implante pas les caractéristiques de Steimann numéro : 7, 9 et 10. Cela signifie que nativement la solution proposée ne permet pas de transférer un rôle entre 2 objets, de spécifier l'état d'un objet avec un rôle et d'associer le même rôle à différent objets.

— Les rôles sont définis indépendamment d'un contexte. La prise en compte du contexte dans la définition des rôles est discutée par Kühn et al. [101].

Les travaux de Cabot and Raventós [32, 33] mettent en avant l'efficacité de la modélisation des rôles en tant que concept distinct par rapport à sa modélisation en tant que relation, sous-type, interface, etc. Dans les chapitres suivants, le concept de rôle est modélisé d'après le patron *Roles as Entity Types Pattern*. Un rôle est alors un concept indépendant joué par un *player* via une relation *RoleOf*.

2.4.1.2 Formalisation par diagramme de caractéristiques

Le travail de Steimann [154] reste fondateur pour clarifier et définir formellement le concept de rôle. En effet pour la première fois, il définit un ensemble de 15 caractéristiques relatives à la modélisation par les rôles permettant de répondre au problème de compréhension commune des rôles. Cependant, en plus d'un problème de compréhension commune, Kühn [100] identifie trois autres facteurs responsables de la non utilisation des rôles :

— La notion de rôle est fragmentée. La plupart des approches de modélisation par les rôles ne prennent pas en compte les résultats précédents. Cela conduit à une stagnation des recherches autour de ce domaine [101].

— Peu d'approches fournissent une base formelle pour leur langage de modélisation basé sur les rôles. De plus la plupart d'entre elles couvrent seulement quelques unes des caractéristiques de Steimann.

— La plupart des langages de modélisation et de programmation basés sur les rôles ne sont pas facilement utilisables, en raison de leur complexité, de leur niveau d'abstraction et/ou de l'absence de prise en charge d'outils.

Pour répondre à ces problèmes Kühn étend les caractéristiques proposées par Steimann et propose un ensemble de 27 caractéristiques présentées dans [101, 100, 106].

Le tableau 2.4 énumère l'ensemble de ces 27 caractéristiques et précise pour chacune si elle s'applique aux niveau des types (M1), des instances (M0) ou aux deux (M0,M1). Les caractéristiques 1 à 15 capturent principalement la nature comportementale et relationnelle des rôles, celles 16 à 19 se focalisent sur les relations entre les rôles et celles de 20 à 27 sur la notion de compartiments.

La figure 2.6 présente le diagramme de caractéristiques pour les langages de modélisation basés sur les rôles proposés par Kühn [100]. Ce diagramme regroupe et structure

1. Roles have its own properties and behaviors	(M0, M1)
2. Roles depend on relationships	(M0, M1)
3. Objects may play different roles simultaneously	(M0 M1)
4. Objects may play the same role several times, simultaneously	(M0)
5. Objects may acquire and abandon roles dynamically	(M0)
6. The sequence of role acquisition and removal may be restricted	(M0, M1)
7. Objects of unrelated types can play the same role	(M1)
8. Roles can play roles	(M0, M1)
9. A role can be transferred from one object to another	(M0)
10. The state of an object can be role-specific	(M0)
11. Features of an object can be role-specific	(M1)
12. Roles restrict access	(M0)
13. Different roles may share structure and behavior	(M1)
14. An object and its roles share identity	(M0)
15. An object and its roles have different identities	(M0)
16. Relationships between roles can be constrained	(M1)
17. There may be constraints between relationships	(M1)
18. Roles can be grouped and constrained together	(M1)
19. Roles depend on compartments	(M0, M1)
20. Compartments have properties and behaviors	(M0, M1)
21. A role can be part of several compartments	(M0, M1)
22. Compartments may play roles like objects	(M0, M1)
23. Compartments may play roles which are part of themselves	(M0, M1)
24. Compartments can contain other compartments	(M0, M1)
25. Different compartments may share structure and behavior	(M1)
26. Compartments have their own identity	(M0)
27. The number of roles occurring in a compartment can be constrained	(M1)

Tableau 2.4 – Ensemble des caractéristiques des rôles [154, 100]

les caractéristiques proposées par Kühn. Elles sont divisées en 9 groupes : la structure des rôles, la notion de jouabilité d'un rôle, les dépendances des rôles, les contraintes des rôles, l'identité d'un rôle, les relations entre les rôles, la structure des compartiments, les éléments inclus dans un compartiment et l'identité d'un compartiment. Dans ce manuscrit, nous proposons de regrouper les caractéristiques conceptuellement proches et proposons une division en quatre catégories :

— Caractéristiques structurelles : cette catégorie regroupe les caractéristiques relatives à la structure des rôles et des compartiments. Ces derniers peuvent avoir des comportements et des propriétés (caractéristiques 1 et 20) et peuvent mettre en œuvre une notion d'héritage (caractéristiques 13 et 25).

— Caractéristiques restrictives : cette catégorie regroupe les caractéristiques relatives aux limitations, dépendances, restrictions et contraintes des rôles. Ces restrictions peuvent s'appliquer à l'acquisition ou à l'abandon d'un rôle (caractéristiques 6). Un rôle peut dépendre d'une relation ou/et d'un compartiment (caractéristiques 2 et 19), par exemple le rôle *Employé* peut dépendre d'un compartiment *Travail*, la notion d'employé n'ayant du sens que dans un contexte de travail. Les relations entre différents rôles peuvent être contraintes (caractéristique 16), par exemple un soldat ne peut appartenir qu'à un seul régiment à la fois. Le nombre de rôles se produisant dans un même compartiment peut être limité (caractéristique 27). La caractéristique 17 souligne la possibilité de définir des contraintes entre des relations telles

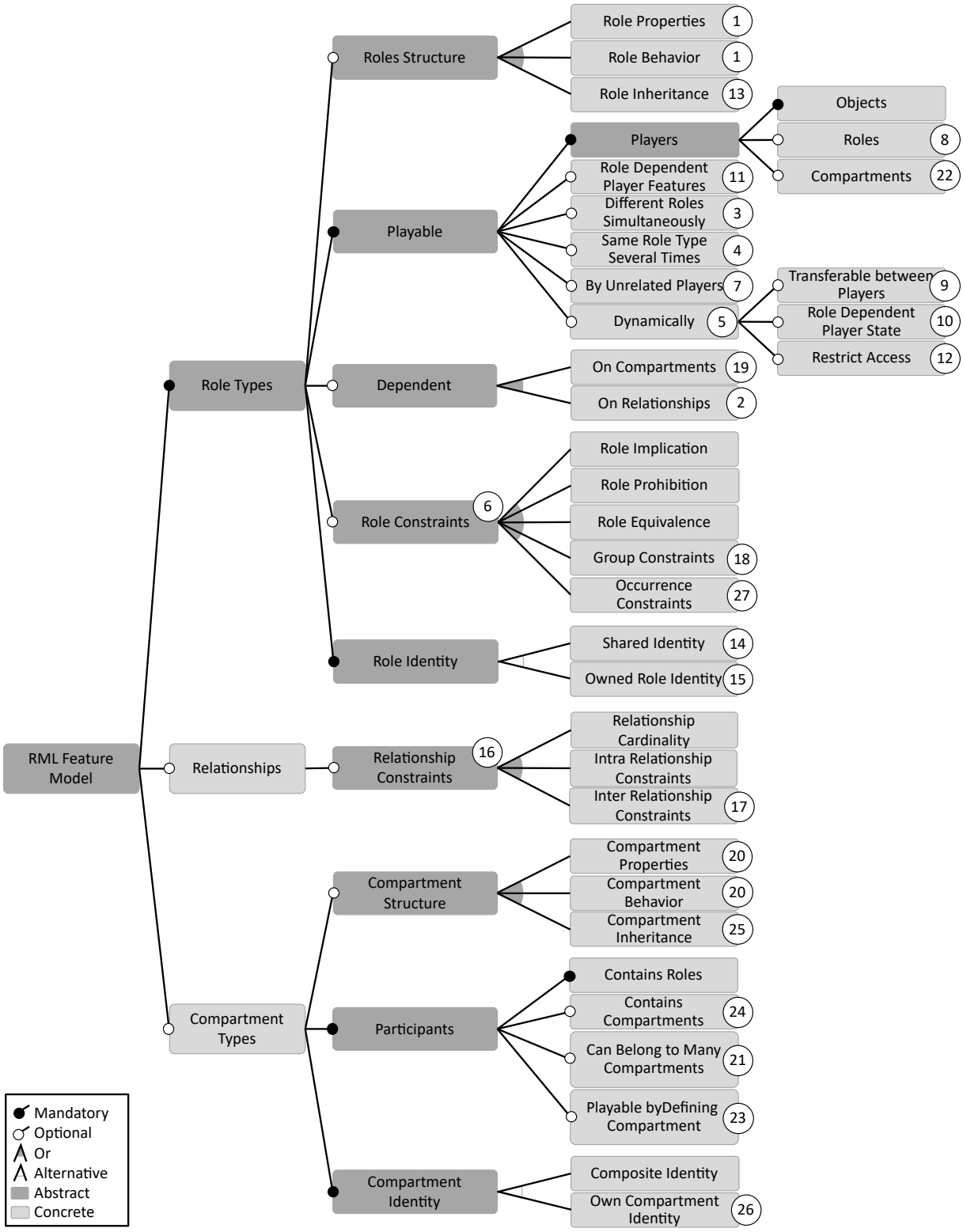


FIGURE 2.6 – Diagramme de caractéristiques de la modélisation par les rôles [100]

que des contraintes d'exclusion mutuelle. Pour finir il est possible de regrouper différents rôles et de définir des restrictions s'appliquant au groupe entier (caractéristique 18).

— Caractéristiques identitaires : cette catégorie regroupe les caractéristiques relatives à l'identité des rôles et des compartiments. Un rôle ou un compartiment peut avoir sa propre identité (caractéristiques 15 et 26) ou il peut partager l'identité d'autres objets (caractéristique 14 et 26).

— Caractéristiques relationnelles : cette catégorie regroupe les caractéristiques relatives au lien entre *player*, rôle et compartiment. Un *player* peut être un objet quelconque, un autre rôle (caractéristique 8) ou un compartiment (caractéristique 22). Il existe trois types de relation :

  — Rôle –> *player* : le fait d'être joué par un rôle est capturé par les caractéristiques 8, 10, 11 et 12. Un rôle peut permettre : d'accéder à une partie du contenu d'un *player* (caractéristique 12), de surcharger la méthode d'un *player* (caractéristique 11) ou encore de modifier l'état perçu d'un *player* (caractéristique 10). De plus, le *player* en question peut être lui même un rôle (caractéristique 8).

  — Player –> rôle : le fait de jouer un rôle est capturé par les caractéristiques 3, 4, 5, 7 et 9. La relation de jeu, à savoir un *player* joue un rôle, peut être dynamique (caractéristique 5) permettant alors à un *player* de jouer de nouveaux rôles ou d'en abandonner un en cours d'exécution. De plus, un *player* peut jouer plusieurs rôles simultanément (caractéristique 3). Des objets de types différents peuvent jouer le même rôle (caractéristique 7). Par exemple une entreprise et un particulier, deux objets de nature différente, peuvent jouer le rôle de client d'une banque. Par ailleurs, un *player* peut jouer plusieurs fois le même rôle simultanément (caractéristique 4), c'est le cas par exemple d'une personne qui a plusieurs emplois. De plus, un *player* peut transférer son/ses rôle(s) à un autre objet (caractéristiques 9). Par exemple lors une succession dans une entreprise, le directeur transférant alors le rôle de PDG à une autre personne.

  — Compartiment <–> contenus : la relation de dépendance à un compartiment est caractérisée par les caractéristiques 21 ,22, 23 et 24. Ce qui dépend d'un compartiment peut être un rôle ou un autre compartiment (caractéristique 24). Un compartiment peut être exclusif, un rôle ne peut alors dépendre d'un seul compartiment ou il peut être éclectique, un rôle peut dépendre de plusieurs compartiments à la fois (caractéristique 21). Pour finir, la caractéristique 22 permet à un compartiment à jouer un rôle, il peut même jouer l'un des rôles qu'il contient (caractéristique 23).

En se référant à l'état de l'art réalisé par Kühn [100] en 2017 comparant 14 approches de modélisation par les rôles, on peut observer qu'en grande majorité les approches couvrent les caractéristiques structurelles, les rôles étant alors définis comme des concepts et non pas de simples relations.

De même, on peut observer que l'ensemble des approches comparées couvrent, au moins partiellement, les caractéristiques restrictives. La notion de rôle est alors contrainte, que ce soit par une relation (caractéristique 2), un compartiment (caractéristiques 19 et 27), d'autres rôles (caractéristique 18) ou d'autres concepts (caractéristique 6).

Les caractéristiques identitaires ne sont pas supportées par un grand nombre d'approches, on peut alors en conclure qu'il n'existe pas, à l'heure actuelle, de consensus sur la notion d'identité des rôles et des compartiments.

Les caractéristiques relationnelles traduisent les notions de "jouer un rôle", "d'être joué par un rôle" et de "dépendre d'un compartiment". Dans la majorité des approches étudiées dans [100], un élément peut jouer un ou plusieurs rôles et un rôle peut interagir avec son *player* (limitation d'accès, spécification d'états ou de méthodes, etc.). Cependant seules peu d'approches permettent à un rôle d'interagir avec un compartiment.

Finalement, les travaux de Steimann et Kühn définissent un ensemble de caractéristiques permettant de décrire clairement une approche de modélisation par les rôles. L'étude comparative menée dans [100] permet de mettre en avant les caractéristiques adoptées par la majorité des approches et celles rarement couvertes. Les caractéristiques le plus souvent couvertes étant celles relatives à la structure des rôles et au lien entre un rôle et un player. Dans les chapitres suivants, un rôle est défini comme un concept distinct qui est joué par un *player* et dont l'implantation couvre un sous-ensemble des 27 caractéristiques de Kühn. Ces travaux seront également utilisés pour positionner notre travail afin de nous placer dans la continuité de cette description des rôles.

2.4.1.3 Formalisation mathématique

Dans cette section nous définissons formellement la notion de rôle au niveau types et au niveau instances. Pour ce faire, nous présentons une formalisation mathématique, proposée dans [102, 100], des notions de "*Compartment Role Object Model with Inheritance*" ($CROM_I$) et de *Compartment Role Object Instance with Inheritance* ($CROI_I$). Ces formalisations combinent les natures comportementales, relationnelles et de dépendance au contexte des rôles.

Modèle d'objet de rôle de compartiment avec héritage

Afin de formaliser cette notion, les auteurs de [102] et [100] définissent les notations suivantes :

- NT : un ensemble de type de naturel.
- RT : un ensemble de type de rôle.
- CT : un ensemble de type de compartiment.
- $T := NT \cup CT$: un ensemble de type rigide.
- RST : un ensemble de type de relation.
- F : un ensemble nom de corps. Par exemple $F = \{$Liquidité, intérêt, bénéfice, ...$\}$.
- M : un ensemble nom de méthode. Par exemple $M = \{$augmentation, ...$\}$.
- $MM(S)$: l'ensemble de tous les multiensembles [1] finis de S.
- $\prec$: la relation de sous-typage. $\mathcal{X} \prec \mathcal{Y}$ indique que $\mathcal{X}$ est un sous-type de $\mathcal{Y}$.

Un modèle d'objet de rôle de compartiment avec héritage $CROM_I$ est fini comme :
$CROM_I = (NT, RT, CT, RST, F, M, fills, rel, fields, methods, \prec_{NT}, \prec_{CT})$
avec :

- $fills \subseteq T \times CT \times RT$: une relation qui signifie que pour un type de compartiment donné, les types rigides ne peuvent jouer que des rôles d'un certain type.
- $rel : RST \times CT \to (RT \times RT)$: une fonction partielle [2] capturant les deux types de rôles aux extrémités respectives d'un type de relation défini dans un type de compartiment.
- $fields : T \cup (CT \times RT) \to MM(F \times T)$: une fonction totale [2] affectant un ensemble de corps à chaque type.
- $methods : T \cup (CT \times RT) \to MM(M \times (\bar{T} \to T))$: une fonction totale affectant un ensemble de définitions de méthodes à chaque type.
- $\prec_{NT} \subseteq NT \times NT$: représente la relation de sous type pour les types naturels.

1. Un multiensemble est une généralisation d'un ensemble, un objet peu appartenir plusieurs fois au même multiensemble. Par exemple si $\mathcal{X} = \{1, 2\}$ est un ensemble, $\mathcal{Y} = \{\{1, 2, 1\}\}$ est un multiensemble fini de $\mathcal{X}$

2. Une fonction partielle est une fonction qui n'est définie que sur une partie de son domaine, au contraire une fonction totale est définie sur l'ensemble de son domaine.

— $\prec_{CT} \subseteq CT \times CT$: représente la relation de sous type pour les types de compartiments.
— $\preceq_{\_}$ correspond à la limitation réflexive et transitive de $\prec_{\_}$.

En partant de cette formalisation Kühn [100] présente 11 axiomes, non détaillés ici, caractérisant un modèle d'objet de rôle de compartiment avec héritage bien formé.

Instance d'objet de rôle de compartiment avec héritage

Afin de formaliser cette notion les auteurs de [102] et [100] définissent les notations suivantes :

— N : un ensemble de naturels
— R : un ensemble de rôles
— C : un ensemble de compartiments
— $O := N \cup C$: un ensemble d'objets

Une instance d'objet de rôle de compartiment avec héritage $CROI_I$ est fini comme :
$CROI_I = (N, R, C, type, plays, links, attr)$
avec :

— $type : (N \rightarrow NT) \cup (R \rightarrow RT) \cup (C \rightarrow CT)$: Une fonction de labellisation attribuant un type distinct à chaque instance.
— $play \subseteq O \times C \times R$: Une relation identifiant les objets jouant un certain rôle dans un compartiment spécifique.
— $links : RST \times C \rightarrow 2^{R \times R}$: Une fonction partielle capturant les rôles actuellement liés par un type de relation dans un certain compartiment.
— $attr : (N \cup R \cup C) \rightarrow 2^{F \times O}$: Une fonction totale mappant chaque instance de naturelle, de compartiment et de rôle à son état. Par exemple, si $x := false$ alors $attr(x) = (Boolean, false)$.

En partant de cette formalisation Kühn [100] présente 6 axiomes, non détaillés ici, caractérisant une $CROI_I$ bien formée.

De plus, les auteurs de [102] formalisent les notions de contrainte sur les rôles, sur les relations entre les rôles et sur les relations entre *player* et rôles. Ces dernières ne concernant pas directement la formalisation des rôles, elles ne sont pas détaillées ici.

Dans le chapitre 4, nous mettons en évidence les parallèles existants entre la formalisation de la notion de point de vue présentée précédemment (§2.3.2) et celle de $CROM_I$ et $CROI_I$ présentées ici. De plus, les travaux présentés ici sont utilisés dans le chapitre 4 afin de décrire formellement, via un ensemble d'axiomes et de propriétés, notre approche basée sur la modélisation par les rôles.

2.4.2 Utilisation de la modélisation par les rôles

La modélisation par les rôles est utilisée par des approches variées couvrant des domaines hétérogènes. De nombreux travaux utilisent les rôles pour modéliser des systèmes complexes, par exemple Richardson and Schwarz [138] proposent en 1991 *OODBS*, un cadre de modélisation utilisant la notion de rôle pour surcharger les objets avec de nouveaux états et comportements.

Dans le domaine des bases de données, les rôles sont utilisés, par exemple, pour prendre en charge les oscillations comportementales imprévues d'objets pouvant atteindre de nombreux types tout en gardant une identité unique [133]. D'autres approches proposent d'étendre un langage existant, tel que *RSQL* [88, 89] qui est une extension de *SQL* implémentant les rôles pour capturer la dynamicité et la variabilité des données. Les rôles sont aussi utilisés pour modéliser directement des informations, comme avec *ORM* [70], une approche orientée-fait dédiée à la modélisation, la transformation et l'interrogation d'informations.

La notion de rôle est aussi utilisée pour la gestion d'utilisateur. Par exemple, Ferraiolo et al. [56] modélisent les accès d'un système grâce à des rôles. Cette même notion est également utilisée dans [136] pour capturer le concept d'utilisateur principal qui participe à une exigence d'un cas d'utilisation.

Différentes contributions implantent la notion de rôle dans des langages de programmation et de modélisation. *PowerJava* [16] ou *ObjectTeams/Java* [77] étendent le langage *Java* pour y inclure une notion de rôles. De même, *OntoUML* [68] proposent d'inclure les rôles dans des diagrammes *UML*.

Dans ce manuscrit, nous nous focalisons sur la modélisation et l'analyse du point de vue d'un attaquant, comme présenté dans la section 2.1, cela nécessite de créer des points de vue dynamiques et de gérer l'interopérabilité entre différents modèles sources. C'est pourquoi, parmi la grande variété des domaines utilisant la notion de rôle, nous nous focalisons dans les trois prochaines parties sur leur utilisation pour : la gestion de la dynamicité, la création de point de vue et l'interopérabilité de modèles. Une quatrième partie est consacrée à la présentation de *FRaMED*, un éditeur graphique de modèle basé sur les rôles.

2.4.2.1 Les rôles pour la dynamicité

Différentes études [153, 14, 69, 112] ont souligné les lacunes des langages de modélisation conceptuels classiques, tels que le modèle Entité-Relation, lorsqu'ils sont utilisés pour modéliser des systèmes dynamiques.

Le concept de rôle est depuis ses débuts [113] lié avec la notion de dynamicité, un élément peut, au cours de son existence, s'associer et se dissocier d'un rôle dynamiquement. Par exemple, dans [66], les auteurs utilisent les rôles pour modéliser des aspects et soulignent que le concept de rôle fournit de nouvelles propriétés aux aspects, en particulier en termes de dynamicité. La notion de dynamicité des rôles se retrouve dans l'ensemble de la recherche les classifiant [95, 98, 12, 94, 154, 33, 67, 103].

Tamai et al. [158] utilisent les rôles pour capturer la dépendance d'un objet à un environnement qui évolue. Un objet peut se déplacer dans son environnement, l'environnement lui-même peut être amené à changer au cours du temps et un objet peut appartenir à plusieurs environnements à la fois. Pour résoudre cette dépendance, les auteurs de[158] présentent *Epsilon*, une approche permettant de décrire finement la séparation des préoccupations qui supporte l'adaptation et la réutilisation de code. D'autres recherches traitent du changement dynamique d'objets en utilisant le concept de rôles, par exemple pour capturer le cycle de vie d'un objet dans une base de données [65] ou encore pour la construction de cadre permettant la documentation et la compréhension de code source [139]. Cette notion de dynamicité d'un rôle se retrouve dans [71], les auteurs comparent ici le concept de rôle et celui d'aspect. Une de leurs conclusions est que les rôles sont à privilégier pour représenter des vues dynamiques.

Nous avons trouvé dans la littérature seulement deux exemples d'utilisation des rôles de manière purement statique [38, 18]. Dans [38] les auteurs cherchent à exhiber les qualités des rôles en dehors de l'aspect dynamique en particulier pour matérialiser des dépenses. A notre connaissance, JavaStage [18] est la seule implantation utilisant uniquement l'aspect statique des rôles, l'objectif des auteurs de [38] et de [18] est de prouver que le concept de rôle a un intérêt autre que la prise en compte de la dynamicité d'un système.

Finalement la notion de dynamicité est centrale au concept de rôle, elle permet d'adapter un modèle de rôles aux évolutions du système et de son environnement. Un *player* peut alors, au cours de son cycle de vie, jouer différents rôles et s'en dissocier en fonction de l'évolution du système. L'aspect dynamique des rôles sera abordé dans les chapitres 4 et 5 afin d'adapter la perception d'un attaquant et de simuler son comportement lors d'une attaque.

2.4.2.2 Les rôles pour la description de point de vue

Andersen [13, 12] est l'un des premiers chercheurs à proposer d'utiliser les rôles pour modéliser des points de vue. Il avance que la modélisation par les rôles a pour objectif de séparer les préoccupations, permettant alors au concepteur d'application de considérer différentes vues. Un modèle de rôles est alors un point de vue distinct de la conception globale du système. Un rôle est considéré ici comme une vue sur un objet dans un contexte particulier.

Kristensen and Østerbye [99] définissent un rôle comme un point de vue temporaire sur un système. Un rôle est ici attaché à un objet via un lien nommé "*Rolification*". Dans cette approche, un objet vu par un rôle garde toutes ses propriétés, auxquelles s'ajoutent les propriétés propres au rôle. Au cours de sa vie, un objet peut changer dynamiquement de rôle et être lié à plusieurs rôles à la fois.

La création de plusieurs points de vue sur un même système, via un ensemble de rôles, est abordé par Bardou [19] et par Reenskaug et al. [137]. Le premier définit un rôle comme un point de vue particulier sur une entité : celui de sa participation à l'application d'une tâche. Les seconds avancent qu'un système est trop complexe pour être représenté en un seul modèle. Ils proposent alors une méthode permettant d'isoler un ensemble de préoccupations en créant un modèle de rôles. Un système est alors représenté via différents points de vue chacun reposant sur un modèle de rôle.

Zhang et al. [171] utilisent les rôles pour modéliser trois types de points de vue sur un système.

— Un point de vue application décrivant, via des modèles, la structure et le comportement d'un système.

— Un point de vue architecture décrivant physiquement chacun des éléments composant un système.

— Un point de vue cartographique décrivant l'allocation des applications sur les éléments physiques.

Ces trois types de points de vues, représentés par des rôles, permettent de prendre en compte l'ensemble des préoccupations du domaine de l'ingénierie, de la conception et de la mise en œuvre du système.

Finalement, depuis sa définition en 1992 [13, 12] jusqu'à nos jours [106] la modélisation par les rôles est utilisée afin de représenter des points de vue sur un système. Nous présenterons dans le chapitre 3 notre approche utilisant la modélisation par les rôles afin de décrire le point de vue d'un attaquant sur un système.

2.4.2.3 Les rôles pour l'interopérabilité et la synchronisation

Reenskaug et al. [137] abordent l'intérêt de la modélisation par les rôles pour représenter un système complexe via plusieurs modèles interconnectés.

On retrouve cette utilisation des rôles dans [93] où les auteurs gèrent l'interopérabilité des politiques de sécurité entre des domaines hétérogènes via un ensemble de rôles. Par exemple, les droits et accès d'une personne jouant le rôle de manager dans un domaine industriel peuvent être équivalents à ceux d'un professeur dans un domaine scolaire. Les rôles permettent également la gestion de l'interopérabilité entre diverses ontologies [74].

Parmi la littérature existante, les travaux de Seifert et al. [143], de Zhang and Møller-Pedersen [169] et de Werner and Aßmann [165] présentent trois utilisations différentes des rôles pour gérer l'interopérabilité de différents outils.

Interopérabilité d'outils modélisée par les rôles

Seifert et al. [143] utilisent la notion de rôles pour gérer l'interopérabilité et l'échange de données entre différents outils. Les modèles de rôles sont ici utilisés pour définir une abstraction implémentant un outil spécifique. Chaque rôle définit un type d'objet nécessaire pour atteindre une fonctionnalité d'un outil. Contrairement à l'utilisation d'interfaces ou de classes, les rôles ne font pas de distinction entre les données physiquement représentées et les données obtenues à partir d'autres objets. Dans leur étude, les auteurs de [143] présentent un ensemble d'exigences pour l'intégration d'outils :

— Abstraction appropriée : chaque outil doit fonctionner sur une abstraction de données (un métamodèle) qui lui est propre.

— Indépendance : un outil doit être autonome et utilisable indépendamment d'autres outils.

— Données partagées : plusieurs outils doivent être en mesure d'accéder à et de manipuler des données partagées.

— Interaction : divers outils doivent pouvoir interagir si nécessaire.

Les auteurs démontrent qu'une solution d'interopérabilité par les rôles permet de couvrir l'ensemble de ces exigences. Leur solution s'appuie sur deux langages distincts permettant de spécifier à la fois les abstractions de données utilisées par les outils (c'est-à-dire les modèles de rôles) et les interactions possibles entre eux (c'est-à-dire les liaisons entre modèles de rôles).

Le langage de création de modèles de rôle est présenté en figure 2.7. Un *Role Model* ré-

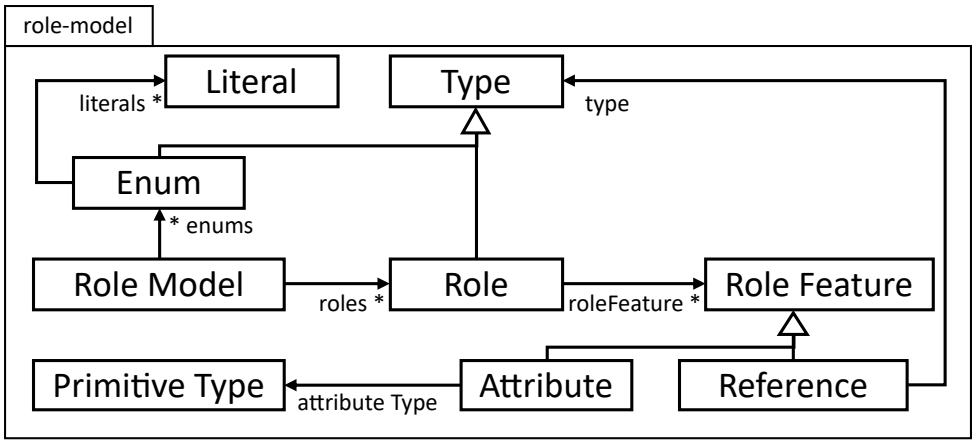


FIGURE 2.7 – Métamodèle du langage de modélisation par les rôles de Seifer et al. [143]

férence un ensemble de *Roles* et d'*Enums*. Un *Role* référence un ensemble de *RoleFeatures* qui peuvent être un attribut avec un type primitif ou une référence à un type complexe. Une *Enum* est un type complexe référençant des littéraux. La notion de rôle est utilisée en remplacement des structures de données concrètes basées sur des classes. Le métamodèle décrivant un outil est alors défini avec des types de rôles à la place de classes. Les données requises pour implémenter les fonctionnalités d'un outil sont modélisées via des attributs de rôle ou via des références connectant plusieurs types de rôle.

Le langage de composition de rôles est présenté en figure 2.8. Une *Composition* référence un ensemble de *RoleGrounding* et de *RolesBinding*. Le concept de *RolesBinding* permet la mise en relation des rôles. Chaque instance de *RoleBinding* relie un rôle d'un modèle de rôle à un *player*. Ici le *player* est un rôle provenant d'un autre modèle de rôle. Un *RoleBinding* référence des *RoleFeatureBindings*, un *player* peut alors manipuler certains des attributs et des références du rôle qu'il joue, permettant ainsi le partage de données entre différents rôles.

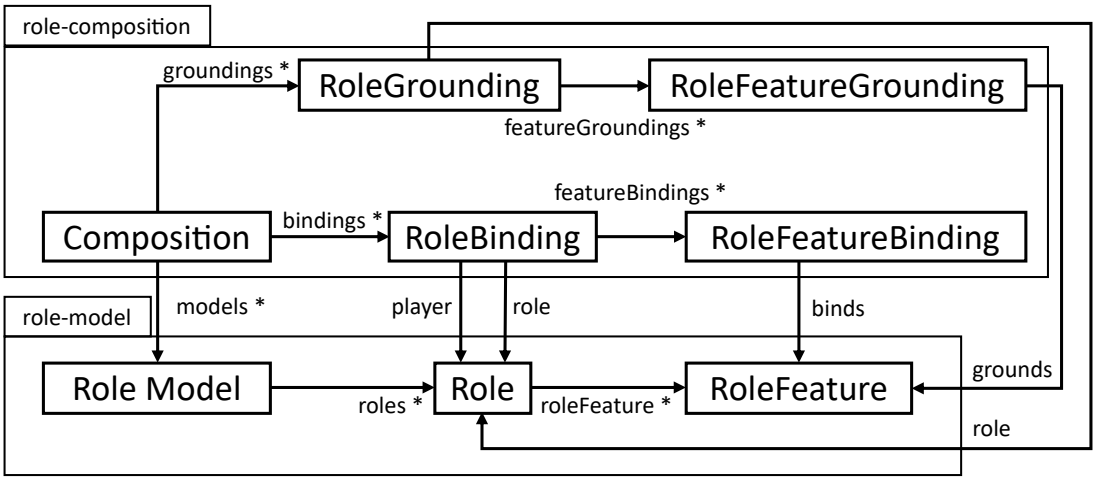


FIGURE 2.8 – Métamodèle du langage de composition de rôle de Seifer et al. [143]

Le concept de *RoleGrounding* est utilisé pour identifier les rôles et les fonctionnalités de rôle qui matérialisent physiquement les données. Les rôles ne faisant pas de distinction entre les données physiquement représentées et les données obtenues à partir d'un autre objet, l'utilisation de *RoleGrounding* permet de reporter la décision de matérialisation des données jusqu'au moment de l'intégration des outils. Les données partagées par plusieurs rôles sont matérialisées physiquement dans un seul d'entre eux, cela permet d'éviter les problèmes de réplication et de synchronisation des données.

Finalement l'approche de Seifert et al. se base sur trois étapes :

— Modélisation d'outils : chaque outil que l'on souhaite interconnecter doit être modélisé avec les notions de *Role* et d'*Enum*.

— Modélisation des relations : un ensemble de *RoleBinding* est créé afin de modéliser les relations entre les divers éléments des métamodèles décrivant les outils.

— Matérialisation des données : un ensemble de *RoleGrounding* est créé afin d'identifier les éléments de modèles qui matérialisent physiquement les données.

L'étape de modélisation des outils couvre les exigences d'abstraction et d'interaction présentées précédemment. Les étapes modélisation des relations et de matérialisation des données permettent de couvrir les exigences de partage de données et d'interactions entre outils. La principale limitation de l'approche proposée par Seifert et al. [143] est qu'elle n'est applicable que sur des outils modélisés via le langage de création de modèles présentés figure 2.7. Avec cette approche il n'est pas possible d'intéropérer des outils préexistants.

Intégration d'outils hétérogènes via des modèles de rôle

Dans leurs travaux Zhang and Møller-Pedersen [171, 169, 170] utilisent les notions de rôle et d'artefacts pour gérer l'intégration d'outils. Dans leur proposition initiale [171], ils introduisent les artefacts comme des représentants de modèles ou d'éléments de modèles. Un artefact possède les propriétés requises pour une intégration d'outils, telles que la traçabilité. Il permet aussi, via une spécialisation, de gérer les transformations de données entre différents outils. Un artefact, représentant un élément de modèle, peut jouer un ou plusieurs rôles, chaque rôles permettant une intégration différente. Les rôles sont utilisés ici pour représenter les concepts communs à plusieurs outils. Dans [169], les auteurs étendent leur définition des rôles pour inclure une prise en charge du contexte via des rôles sémantiques et des relations entre artefacts via des rôles d'association. Dans un certain contexte, deux éléments provenant d'outils distincts peuvent représenter la même

notion, cette notion est ici modélisée par un rôle sémantique liée à un contexte précis. Un rôle d'association permet de créer une relation entre deux artefacts, elle est utilisée afin de transmettre des informations d'un artefact à un autre, et donc d'un élément d'un modèle à un élément d'un autre modèle. De plus, les auteurs introduisent la notion d'adaptateur. Un adaptateur expose certaines fonctionnalités d'un outil sous forme de services qui sont interrogés par des artefacts. Dans la suite de leurs travaux, Zhang and Møller-Pedersen [170] se focalisent sur l'aspect dynamique de leurs notions de rôles. La flexibilité apportée permet alors d'adapter, en cours d'exécution, une sémantique commune. Si deux éléments ne partagent plus la même sémantique (à cause d'un changement de contexte par exemple) alors les rôles sémantiques associés peuvent être dynamiquement dissociés des artefacts.

Le langage d'intégration basé sur les rôles proposés dans [170] est modélisé figure 2.9. Un outil est manipulé au travers d'une API par un adaptateur. Cet adaptateur contient

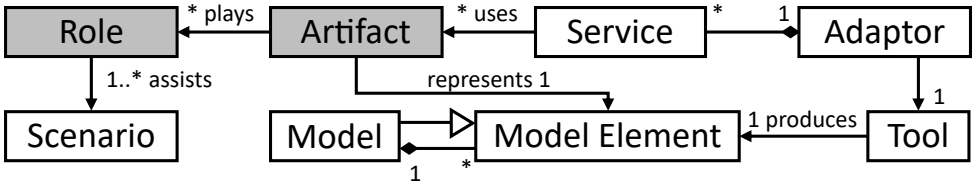


FIGURE 2.9 – Métamodèle du langage d'intégration par les rôles de [170]

des services qui peuvent être utilisés par des artefacts. Un artefact représente un élément de modèle produit par un outil. De plus, il peut jouer un ou plusieurs rôles capturant un ensemble de propriétés spécifiques à un scénario d'intégration. Zhang and Møller-Pedersen couvrent partiellement les exigences de Seifert et al. [143] :

— Abstraction appropriée : (couverte) Les outils de base ne sont pas impactés par la mise en place des relations d'interopérabilité, chacun peut alors fonctionner sur une abstraction de données qui lui est spécifique.

— Indépendance : (couverte) Les outils de base ne sont pas couplés, ils peuvent donc fonctionner indépendamment les uns des autres.

— Données partagées : (partiellement couvert) Les auteurs permettent le partage de donnée via l'utilisation de rôles mais les données partagées sont dupliquées dans les différents outils et les différents artefacts représentant ces outils. Une solution de synchronisation doit alors être mise en place entre les éléments de modèle et les artefacts et entre différents artefacts.

— Interaction : (partiellement couvert) Un artefact contient des opérations spécifiant les services autorisés à accéder à un élément de modèle d'un outil [169]. Cette caractéristique des artefacts permet la manipulation de données contenues dans un outil depuis l'extérieur de l'outil. En revanche, seules les interactions prévues lors de la création d'un artefact sont possibles, l'approche proposée ne permet pas d'ajouter dynamiquement des interactions.

Finalement, la solution d'interopérabilité par les rôles de Zhang and Møller-Pedersen peut se décliner en trois étapes :

— Représentation des éléments de modèle : Les artefacts sont définis avant l'intégration afin de représenter les éléments des modèles des outils à intégrer. Un outil est alors représenté par un ensemble d'artefacts connectés avec ce dernier grâce à des adaptateurs.

— Prise en compte d'un scénario : Un ensemble de rôles est défini durant le processus d'intégration afin de spécifier les informations relatives à un scénario d'intégra-

tion. Un rôle peut être dynamiquement ajouté ou retiré à un artefact en fonction du contexte d'intégration.

— Spécification du scénario d'intégration : Des modèles de processus sont définis afin de spécifier l'ordre et le séquencement des activités d'intégration. Un modèle de processus met en œuvre une collaboration entre des adaptateurs de différents outils. Pour ce faire, il définit toutes les étapes du processus d'intégration devant être exécutées via un ensemble d'artefacts et de rôles. Le scénario d'intégration est défini à l'aide de modèles BPMN [1] et de diagramme de séquence UML.

L'approche présentée ici fait une distinction claire entre les modèles sources (les outils) et les modèles d'intégration (Artefacts et rôles) et permet la réutilisation des modèles d'intégrations. Cependant, la solution de partage de données adoptée entraîne une recopie de l'information entre différent artefacts. En outre, l'interaction entre outils est limitée à un ensemble d'actions prédéfinies dans des artefacts.

Les deux solutions d'interopérabilité détaillées ici [143, 170] sont deux approches complémentaires utilisant la notion de rôle. La première permet de partager simplement des données entre divers outils mais ne prend pas en compte les outils pré-existants et la seconde peut être utilisée pour tous types d'outils mais nécessite une duplication des données partagées et ne permet pas l'ajout dynamique de nouvelles intégrations.

Synchronisation bidirectionnelle par les rôles

Werner and Aßmann [166, 165, 167] exploitent les rôles pour synchroniser des modèles hétérogènes. Dans [166] les auteurs présentent une solution de synchronisation bidirectionnelle et dynamique utilisant les rôles. L'utilisation des rôles permet d'ajouter et de personnaliser des règles de synchronisation au moment de l'exécution. Pour ce faire, il suffit de créer/supprimer dynamiquement des règles de synchronisation et d'associer/dissocier des rôles aux éléments du modèle désiré. Si deux types d'éléments appartenant à deux modèles distincts doivent être synchronisés, la synchronisation doit couvrir la création, la suppression et la modification de ces éléments et doit propager toutes les modifications à tous les modèles associés. L'approche proposée représente les règles de synchronisation sous forme de compartiments, un compartiment contenant un ensemble de rôles de synchronisation joués par des éléments des modèles sources. La figure 2.10 schématise cette approche. Chaque élément de modèle est lié à un *RoleManager* dans un *SynchronizationContext*. Chaque *RoleManager* joue plusieurs rôles de synchronisation lui permettant de gérer la construction, la destruction et la synchronisation du contenu des éléments du modèle. Pour chaque *SynchronizationContext*, il n'existe qu'un seul *ConstructionContext* et un *DestructionContext* mais il existe autant de compartiments de synchronisation (*SyncNames*), que d'éléments de modèle synchronisés. Cette conception permet d'ajouter ou de remplacer dynamiquement des règles de synchronisation sans affecter les éléments du modèle, car les nouveaux rôles de synchronisation ne sont liés qu'aux *RoleManagers* correspondants.

Se basant sur [101], Werner and Aßmann proposent dans [165] et [167] une approche orientée rôle fournissant un moyen de créer des vues à partir d'un modèle en cours d'exécution et introduisant un mécanisme d'adaptation flexible. La notion de compartiment est utilisée pour modéliser les relations entre différent éléments de modèles. La notion de rôle est ici utilisée pour gérer l'interaction entre différentes vues en réagissant aux opérations d'insertion, de suppression et de modification des éléments de modèles. Les rôles ne contiennent pas directement d'informations, ces dernières peuvent être lues et modifiées par les rôles directement dans les modèles sources.

1. BPMN *(Business Process Model and Notation)* est une méthode de modélisation graphique de processus métier.

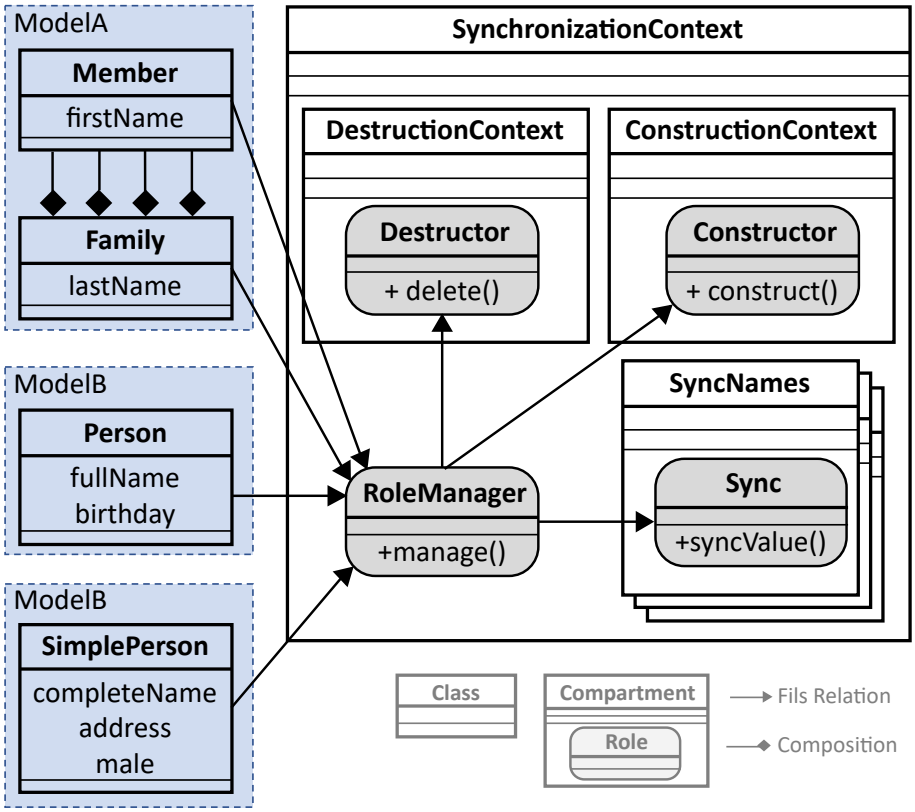


FIGURE 2.10 – Schématisation d'une synchronisation de modèle réalisé grâce aux rôles [166]

Dans le chapitre 4 nous présentons notre solution d'interopérabilité par les rôles réunissant la simplicité de l'approche de Seifert et al. [143], la généricité de l'approche de Zhang and Møller-Pedersen [170] et la flexibilité de la solution de synchronisation par les rôles présentée par Werner and Aßmann [166, 165, 167].

2.4.2.4 Présentation de FRaMED et CROM

Kühn [100], énumère différents facteurs responsables de la non utilisation des rôles par certains chercheurs et praticiens : la non-prise en compte des travaux précédents, l'absence de base formelle pour les langages de modélisation basés sur les rôles et le manque d'outils de conception, de validation et de génération de logiciels fondés sur les rôles.

Le cadre de modélisation *CROM* [103, 105] et l'éditeur *FRaMED* [104] permettent de pallier à ces différents manques. *CROM* (*Compartment Role Object Model*) est un cadre de modélisation conceptuelle qui incorpore la plupart des caractéristiques des rôles dans un langage de modélisation graphique cohérent. *FRaMED* (*Full-fledged Role Modeling Editor*) est un éditeur de modélisation graphique pour *CROM*, englobant toutes les natures de rôles et les contraintes de modélisation. Il fournit un moyen de générer des métamodèles pour des langages de modélisation et de programmation basés sur les rôles simplement en fournissant un ensemble de caractéristiques de rôles. Comme le schématise la figure 2.11, l'outil proposé dans [104] permet de générer un métamodèle décrivant un langage de modélisation basé sur les rôles à partir d'un sous-ensemble de 27 caractéristiques présenté dans la section 2.4.1. Les propriétés comportementales, relationnelles et contextuelles des rôles dans ce langage sont définies par le sous-ensemble sélectionné.

La figure 2.12 présente la notation graphique définie par *CROM*. Cette notion gra-

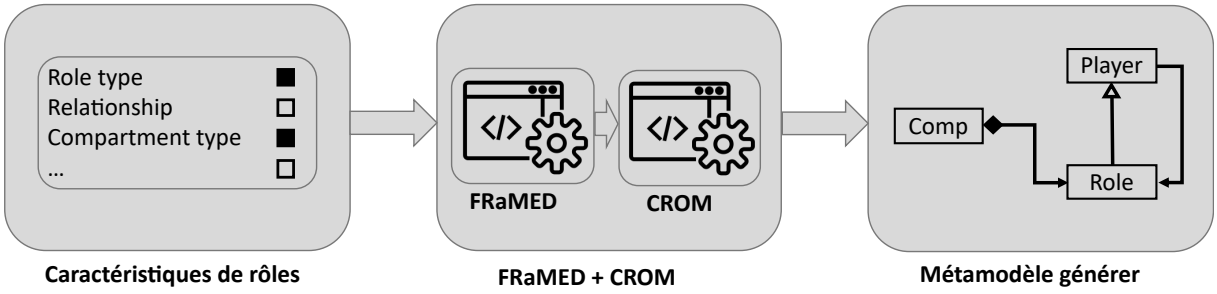


FIGURE 2.11 – Schématisation de la génération d'un métamodèle avec *FRaMED* [104] et *CROM* [105]

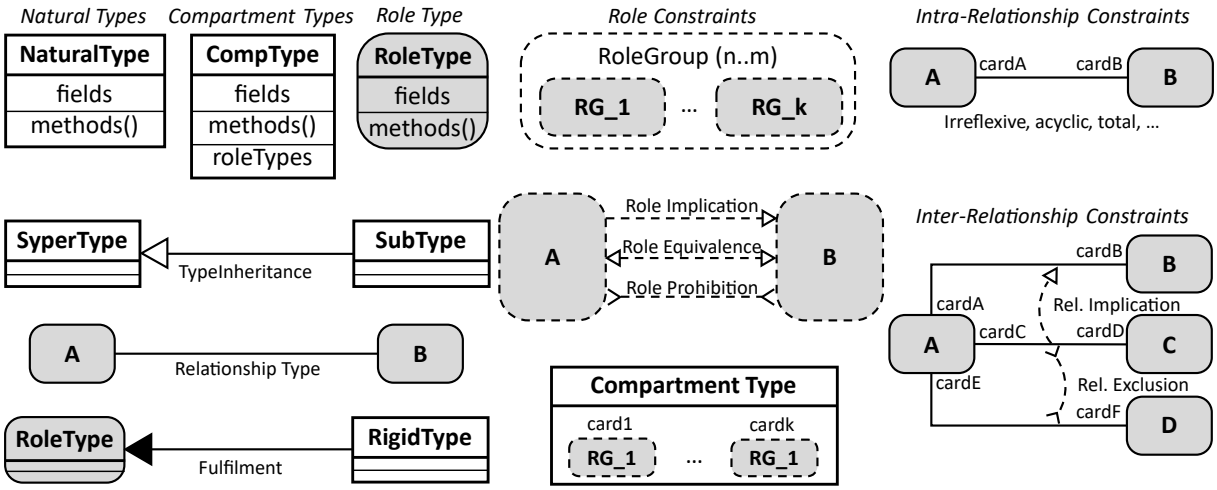


FIGURE 2.12 – Notation graphique de *CROM* [105]

phique est conçue pour distinguer visuellement les types naturels, les types de compartiments et les types de rôles ainsi que leurs relations. Tous les types sont représentés sous forme de rectangles contenant au moins trois parties incluant un nom, des attributs et des opérations. Seuls les types de compartiments contiennent un segment supplémentaire pour les types de rôles participants au compartiment. Les types de rôles sont représentés sous forme de rectangles arrondis. Les relations entre les types de rôles sont représentés sous la forme de lignes étiquetées, l'héritage entre les types est dessiné avec une flèche à tête blanche allant du sous-type vers le super-type et la relation *fulfillment* est indiquée par une flèche à tête noire entre un type rigide et un type de rôle.

La notation graphique présentée ci-dessus ainsi que l'outil de génération de métamodèle *FRaMED* seront utilisés dans le chapitre 4 afin de représenter différents modèles de rôles.

2.4.3 Limitations de la modélisation par les rôles

La modélisation par les rôles a été popularisée dans les années 90 [13] et a connu plusieurs étapes de formalisation [12, 154, 33, 101]. Dans cette section, nous discutons des limites de la modélisation par les rôles.

Comme le souligne Steimann [154] et Kühn [100], les travaux autour de ce concept ne prennent que rarement en compte les résultats précédents, ce qui a conduit à une stagnation des recherches autour de ce domaine. L'une des causes de ce problème est le manque de cohérence globale autour de la notion de rôle, liée à la pluri-naturalité des rôles. En

effet, un rôle possède trois natures, à savoir une nature comportementale, une nature relationnelle et une nature contextuelle [104].

La nature comportementale établit qu'un objet peut jouer un rôle pour adapter son comportement. La nature relationnelle établit qu'un rôle est lié à une relation et qu'il peut être joué par un objet ayant des propriétés spécifiques à cette relation. La nature contextuelle établit qu'un rôle et ses relations sont encapsulés dans un contexte délimitant une frontière de définition. Les approches de modélisation par les rôles peuvent couvrir une [41], deux [70] et parfois même les trois [76, 90] natures des rôles. Cependant, selon la ou les natures couvertes, les auteurs proposent des implantations différentes de la notion de rôles. Par exemple, un rôle peut être implémenté via un attribut, une relation, une spécification, une interface ou encore un type distinct.

De plus, l'abondance de solutions formalisant le concept de rôle dans les années 90 [98, 95, 94] était symptomatique du manque de stabilité de la notion. Mais nous pouvons constater que les travaux de Steimann [154] et Kühn et al. [101] constituent une étape importante pour la formalisation du concept de rôle. C'est pourquoi, dans la suite de nos travaux, nous nous baserons sur leurs travaux pour positionner et formaliser notre travail de l'utilisation des rôles dans le contexte de la cybersécurité.

2.5 Conclusion de l'état de l'art

L'analyse de l'état de l'art nous permet d'identifier un certain nombre de problématiques que nous cherchons à adresser dans la suite de ce document.

La cybersécurité foisonne d'actions de recherche et développement dans de nombreux domaines dont la modélisation et l'analyse de la menace. Dans ce cadre, il faut noter qu'il y a pas de consensus sur la définition d'un attaquant et/ou de la menace, ce qui nous semble justifié, compte tenu de la diversité à laquelle nous faisons face. Donc, pour nous, la problématique n'est pas de proposer une nouvelle définition, il est surtout question de proposer une modélisation reposant sur des définitions existantes tout en assurant flexibilité et adaptation à différents contextes. Nous apportons une contribution à cette problématique dans le chapitre 3.

Notre approche de modélisation, appliquée à l'attaquant d'un système, repose sur des points de vue construits à partir de modèles existants tout en offrant une capacité d'adaptation. Cette approche requiert alors une interopérabilité de modèles hétérogènes reposant sur une mise en relation dynamique. De plus, elle nécessite des point de vue adaptatifs et interagissant avec les modèles sources. L'état de l'art sur ces sujets montre que les approches de fédération de modèle et de modélisation par point de vue répondent en partie à ces problématiques.

Dans notre approche, la conceptualisation et l'implantation de ces notions reposent sur l'utilisation des rôles. Ce choix est motivé par une analyse bibliographique montrant que le concept de rôle fournit un cadre pour construire des points de vue flexibles et adaptables tout en servant de support à la fédération de modèles. De plus, le concept de rôle est riche et formalisé par différentes approches.

Cet aspect, le plus important de notre contribution, basé sur la formalisation de Kühn [100], est présenté dans le chapitre 4. Nous y présentons Role4All, un langage dédié à la modélisation de points de vue d'attaquants et aux premières phases d'analyse de menaces. Notre approche est mise en œuvre sur un cas d'études dans le chapitre 5.

Chapitre 3

Conceptualisation de l'analyse de la menace et modélisation d'un point de vue d'un attaquant

Sommaire

Les deux chapitres suivants sont consacrés à la présentation de notre contribution centrée sur la création du point de vue d'un attaquant pour la modélisation et l'analyse de menaces. Pour mener à bien ce travail, nous identifions dans un premier chapitre la méthodologie de construction d'un point de vue d'attaquant. Dans un second chapitre, nous modélisons ce point de vue en nous appuyant sur la fédération de modèles par les rôles, cette dernière représente la pierre angulaire de notre conceptualisation et de notre expérimentation basée sur le langage Role4All.

Dans ce chapitre, la première section est dédiée à la description de notre approche permettant de modéliser le point de vue d'un attaquant et d'effectuer des analyses à partir de ce dernier. La deuxième présente un cas d'étude utilisé par la suite pour illustrer les différent concepts exposés. La troisième section décrit les concepts utilisés pour construire et analyser le point de vue d'un attaquant sur un système. Pour finir, nous détaillons les étapes de construction et les contraintes propres au point de vue d'un attaquant. Pour rappel, nous utilisons dans ce chapitre, les termes de *vue*, de *point de vue*, de *système*, de *modèle source* et de *métamodèle source* défini dans la section 2.3.

3.1 Description de l'approche

Dans cette section, nous présentons notre approche permettant de modéliser le point de vue d'un attaquant. Dans une première section, nous présentons les concepts utilisés et dans une seconde, nous détaillons notre utilisation de ces concepts.

3.1.1 Description des concepts

Comme présenté dans les sections 2.1 et 2.3, nous utilisons les notions de point de vue et de vue afin de représenter un système. Un système est modélisé par un ensemble de modèles sources conformes à des métamodèles sources. Nous créons une représentation de ce système via un ensemble de vues conformes à des points de vue et couvrant des préoccupations données.

Les notions utilisées dans notre approche sont définies dans le modèle conceptuel en figure 3.1. Les relations représentées en gras sur cette figure (*adapte* entre *Point de Vue* et *Préoccupation*, et *décrit* entre *Point de Vue* et *Metamodèle source*) sont détaillées dans la section 3.1.2.

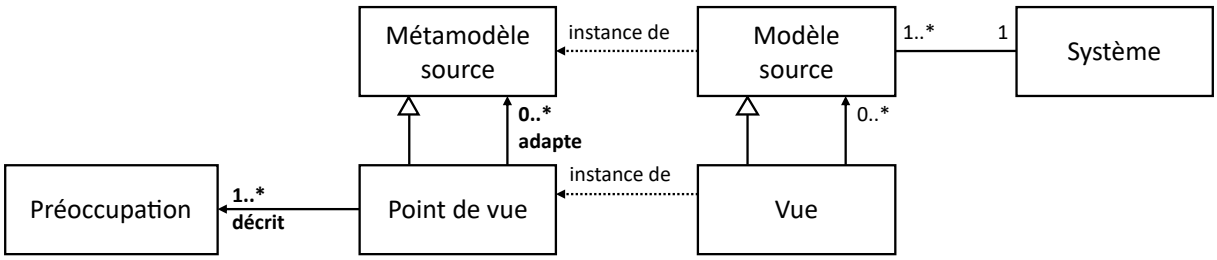


FIGURE 3.1 – Terminologie de la modélisation par les vues, extension de [27]

Notre approche se fonde sur les concepts de :

— Métamodèles sources : ils représentent les formalismes utilisés comme sources de données. Ces dernières peuvent être des bases de données structurées tel NVD ou des DSL tel que *Pimca* [1].

1. Pimca [157] est un DSL permettant de décrire la topologie d'un système et d'identifier ses frontières. Nous présentons et utilisons Pimca dans le chapitre 5 afin de représenter un système dans un contexte de cybersécurité.

— Modèle sources : ils contiennent les informations permettant de décrire un système. Ces dernières peuvent être des informations structurées provenant d'un métamodèle source ou des informations structurées se présentant sous forme textuelle.

— Préoccupation : une préoccupation est un point d'intérêt pertinent pour un individu. Par exemple, la topologie et le cycle de vie d'un système sont deux préoccupations pertinentes pour un attaquant.

— Point de vue : un point de vue décrit un ensemble de préoccupations et modélise les types d'éléments pouvant apparaître dans une vue. Les concepts constituant un point de vue sont définis ou adaptés de métamodèles.

— Vue : une vue est une représentation d'un système conformément à un point de vue. Les informations contenues dans les vues sont définies manuellement, adaptées de modèles sources ou adaptées d'autres vues.

Classiquement, notre approche se base sur : un système modélisé par un ensemble d'outils décrits dans des métamodèles sources et reposant sur des formalismes hétérogènes. Ces outils génèrent des informations contenues dans des modèles sources. Ces informations sont adaptées dans des vues, conformes à des points de vue construits selon des préoccupations spécifiques.

3.1.2 Caractérisation de l'approche

Dans la section 2.1, nous soulignions l'absence de solution globale couvrant le domaine de la modélisation et de l'analyse de la menace. Cette observation illustre un manque de consensus autour des solutions de cybersécurité dû à l'étendue et la variabilité du domaine. Pour permettre de capturer une fraction cohérente de ce domaine, tout en prenant en compte les préoccupations spécifiques à chaque attaquant, nous proposons de construire deux types de points de vue : les points de vue de références (PVR) et les points de vue spécifiques (PVS). Les PVR sont utilisés pour décrire des préoccupations communes à tous les attaquants. Ils sont construits indépendamment des métamodèles sources. Les PVS sont des spécialisations des PVR. Ils permettent de décrire la perception qu'un attaquant a d'un système, selon ses propres préoccupations et ses aptitudes.

Pour préciser notre approche, nous détaillons dans une première partie les relations entre préoccupation, point de vue et métamodèle source et dans une seconde partie les relations entre vue et modèle source.

3.1.2.1 Relations au niveau métamodèle

Un ensemble de préoccupations est représenté par un point de vue via la relation *décrit*. La relation *adapte* est utilisée afin de construire les deux types de points de vue : Point de Vue de Référence (PVR) et Point de Vue Spécifique (PVS). Notre utilisation de la relation *adapte* est particulière au regard de l'état de l'art (§2.3). En effet, nous utilisons cette relation pour adapter des concepts provenant des métamodèles sources dans des PVR mais également pour adapter des PVR dans des PVS. Cette utilisation est possible car, comme modélisé en figure 3.1, le concept de point de vue hérite du concept de métamodèle source.

La figure 3.2 présente, via un exemple, les relations entre PVS, PVR et préoccupations. Comme illustré sur la figure 3.2, l'approche proposée dans ce manuscrit peut se découper en différentes étapes :

① Identification des préoccupations : parmi les préoccupations propres à un domaine, celles jugées comme pertinentes, pour un contexte donné, sont regroupées dans différents ensembles appelés "ensembles préoccupations".

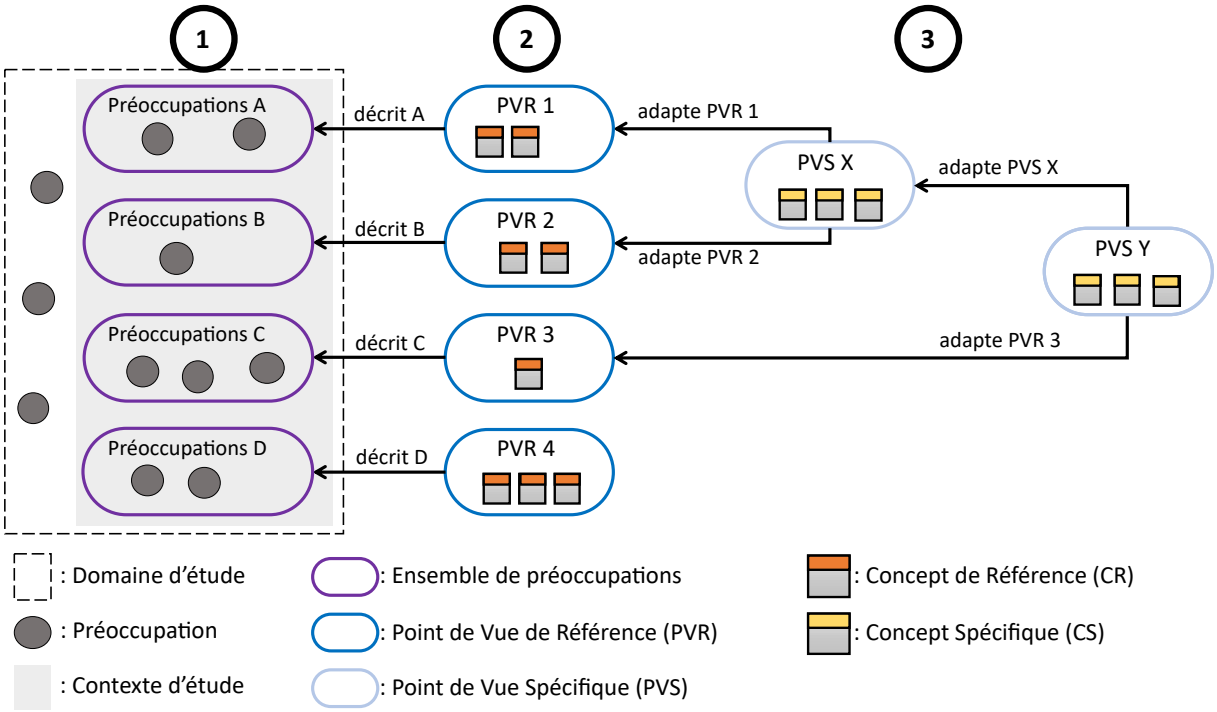


FIGURE 3.2 – Construction des points de vue de référence (PVR) et des points de vue spécialisés (PVS)

② Création de Point de Vue de Référence (PVR) : chaque ensemble de préoccupations identifié est décrit via un PVR. Dans notre contexte de modélisation, un PVR est un métamodèle composé de concepts appelés Concepts de Référence (CR).

③ Création de Point de Vue Spécifique (PVS) : un PVS est une adaptation des PVR couvrant différentes préoccupations. Dans notre contexte de modélisation, un PVS est constitué de concepts spécifiques (CS). Un CS peut être une adaptation de CR ou de CS, une combinaison des deux ou éventuellement un concept additionnel.

Finalement, les PVR permettent de modéliser des préoccupations d'un domaine pour un contexte donné. Les PVS permettent de créer une représentation particulière de ce contexte.

Dans ce manuscrit le domaine d'étude est la modélisation et l'analyse de menaces et le contexte est le point de vue d'un attaquant. Les PVR permettent alors de modéliser l'ensemble des préoccupations propres au point de vue d'un attaquant et les PVS permettent de construire une représentation spécifique à un type d'attaquant. La création des PVR est présentée dans la section 3.3 et celle des PVS dans la section 3.4.

3.1.2.2 Relations au niveau modèle

Comme présenté dans la section 2.3.1 une vue est une représentation d'un système répondant à un ensemble de préoccupations décrites dans un point de vue. Dans un contexte de modélisation, une vue est conforme à un point de vue et un système est modélisé par des modèles sources. Les informations contenues dans une vue peuvent provenir de modèles sources, provenir d'autres vues ou être définies manuellement.

L'existence de deux types de points de vue (PVR et PVS) induit celle de deux types de vue : les Vues de Référence (VR) et les Vues Spécifiques (VS). Par construction, une VR représente un ou plusieurs modèles sources et une VS représente des VR et des modèles

sources. En somme, une VR est une représentation d'un système et une VS est une représentation d'une représentation.

La figure 3.3 schématise les relations vues –> modèles sources et points de vue –> métamodèles sources. Dans notre approche, comme le modélise la relation d'adaptation entre

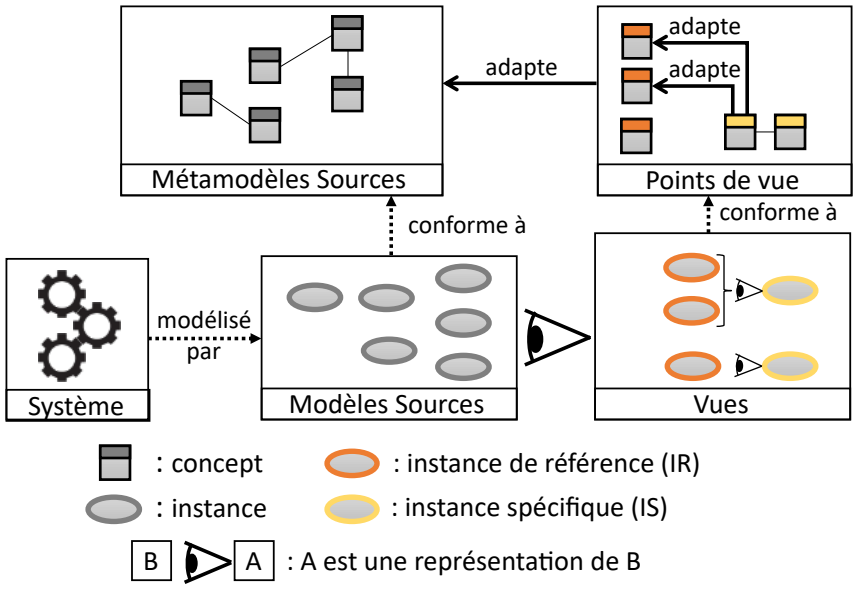


FIGURE 3.3 – Relation de représentation de modèle et d'adaptation de métamodèle.

les points de vue et les métamodèles sources, les CR adaptent les concepts des métamodèles sources. Au niveau modèle, cette relation est représentée par le fait que les Instances de Références (IR) représentent des modèles sources.

Pour résumer, une vue est un ensemble d'instances conformes aux concepts composant un point de vue. Ces instances contiennent des données qui sont une adaptation des informations contenues dans les modèles sources, éventuellement complétées par de nouvelles données. L'adaptation entre une vue et un système est définie au niveau conceptuel, à savoir entre un point de vue et des métamodèles sources. La création de VR est illustrée en section 3.3 et celle de VS en section 3.4.

3.2 Un système fil rouge : Alice et Bob

Dans cette section, nous présentons un système , nommé "système fil rouge", servant d'exemple dans les sections 3.3 et 3.4. Il est constitué d'un réseau de trois machines, d'un attaquant et d'internet. Les machines sont des ordinateurs utilisant un système d'exploitation et des applications dont la description est la suivante :

— Systèmes d'exploitation :

  — Ubuntu Xenial Xerus : Ubuntu est un système d'exploitation libre pour ordinateur et serveur utilisant un noyau Linux. Xenial Xerus de Ubuntu est le nom donné à la version Ubuntu 16.04 LTS sortie le 21 avril 2016. Dans cette exemple, Ubuntu 16.04 LTS utilise la version 4.4.8 du noyau Linux, sortie le 20 avril 2016.

  — Windows 10 Redstone 1 : Windows 10 est un système d'exploitation développé par Microsoft. La version Redstone 1 est le nom donné à la version Windows 10 1607 sortie le 02 août 2016.

- Applications :
  - WordPress Coleman Hawkins V7 : WordPress est un système de gestion de contenu gratuit, libre et open-source permettant de créer et gérer différents types de sites Web. La version Coleman Hawkins V7 de WordPress est le nom donné à la version numérotée WordPress 4.5.7 sortie le 6 Mars 2017.
  - Symposium 14.10.3 : Symposium est une extension de WordPress permettant la création d'un réseau social. La version 14.10.3 de Symposium a été publiée le 17 octobre 2014.
  - Contact Form 7 to Database 2.10.32 : Contact Form 7 to Database est une extension de WordPress permettant la sauvegarde de contact dans une base de données. La version 2.10.32 de Contact Form 7 to Database a été publiée le 23 février 2017.

Les systèmes d'exploitation et les applications présentés ici contiennent des vulnérabilités qui seront modélisées dans la section 3.3.2. Les trois machines constituant le système sont :

- Alice : Alice est un serveur web sur lequel figurent des informations sensibles. Alice utilise le système d'exploitation Windows 10 Redstone 1 et une version de WordPress Coleman Hawkins V7 utilisant l'extension Contact Form 7 to Database 2.10.32. Alice est connectée avec Bob.
- Bob : Bob est un serveur web contenant le système d'exploitation Ubuntu Xenial Xerus. Bob utilise les mêmes applications qu'Alice, plus l'extension Symposium 14.10.3. Bob est connecté avec Alice et Carol.
- Carol : Carol est un serveur web et possède le même système d'exploitation et les mêmes applications que Bob. Dans notre exemple, cette machine sera considérée comme éteinte tout au long de l'attaque. Carol est connectée avec Bob.

La figure 3.4 schématise le système fils rouge avant l'attaque. Dans cet exemple l'attaquant sera appelé Mallory. Mallory veut accéder aux informations sensibles que détient Alice. Pour ce faire, il déroule un scénario d'attaque en quatre étapes numérotées de e0 à e3. Ces étapes sont :

e0 | Situation initiale : Mallory sait qu'Alice est dans le système fil rouge mais n'a aucune autre information sur cette dernière.

e1 | Connexion au système : Mallory communique, via internet, avec le système et découvre les machines d'Alice et de Bob. Cette étape peut être découpée en sous-étapes :

- e1.1 | Découverte du système : Mallory établit une connexion avec les machines d'Alice et de Bob.
- e1.2 | Exploration du système : Mallory tente d'obtenir des informations sur Alice et Bob. Il découvre alors une partie des applications présentes sur ces machines.

e2 | Prise de contrôle : Mallory tente de prendre le contrôle d'Alice et de Bob. Il réussit à prendre le contrôle de Bob. Cette étape peut être découpée en sous-étapes :

- e2.1 | Entrée dans le système : Mallory obtient, via l'exploitation d'une vulnérabilité, un accès limité à Bob. Cela lui permet d'en apprendre plus sur la configuration de ce dernier.
- e2.2 | Élévation de privilège : Mallory, tirant partie de ses nouvelles connaissances et accès, exploite une vulnérabilité lui permettant de prendre le contrôle de Bob.

e3 | Récupération d'informations : Mallory, via Bob et grâce à l'exploitation d'une vulnérabilité, subtilise les informations détenues par Alice. Cette étape peut être découpée en sous-étapes :

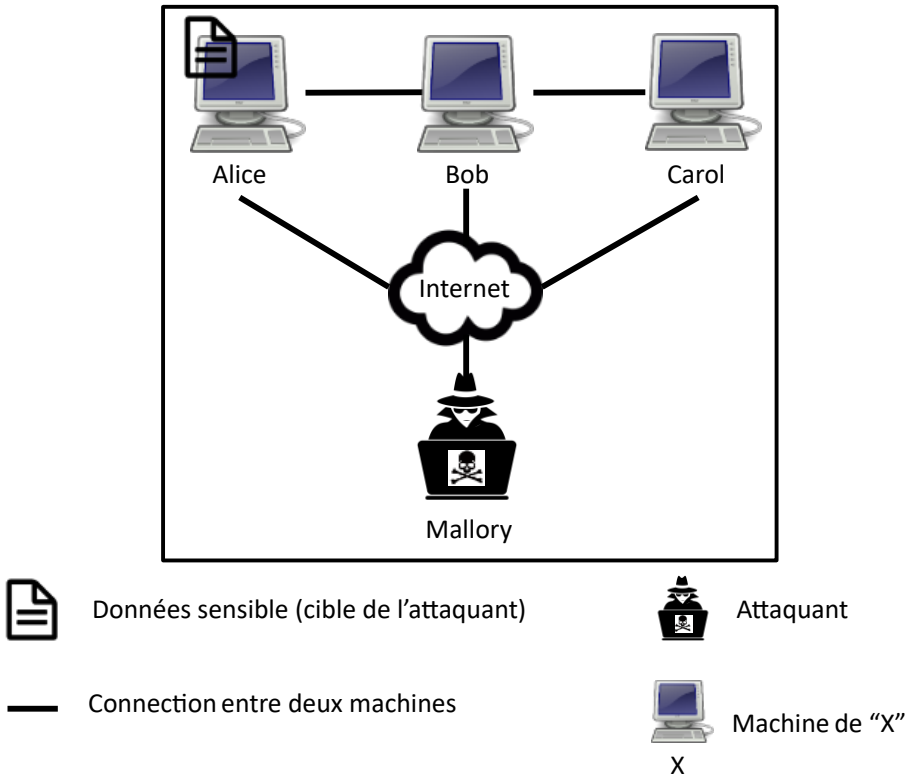


FIGURE 3.4 – Représentation schématique du système "fil rouge"

e3.1 | Pivotage et découverte du réseau : Mallory utilise la machine Bob afin de scanner le réseau du système fil rouge. Il découvre alors que Bob est connecté à Alice. Lors de cette étape, l'attaquant ne découvre pas la connection entre Bob et Carol car Carol n'est pas allumée.

e3.2 | Exploration du réseau : Mallory, via la machine Bob, apprend quel est le système d'exploitation utilisé par Alice.

e3.3 | Collecte de données : Mallory utilise une vulnérabilité afin de récupérer les informations détenues par Alice.

Le système fil rouge consiste donc en un réseau de trois machines sur lesquelles s'exécutent divers systèmes d'exploitation et applications vulnérables. Ce réseau est attaqué par un attaquant ayant pour objectif de récupérer des informations sur la machine nommé Alice. Des descriptions plus approfondies du système et de l'attaquant sont données en section 3.3.2.

3.3 Définition du concept d'attaquant et construction de points de vue de références

Dans la section 3.1, nous présentons notre approche de modélisation du point de vue d'un attaquant. Nous introduisons également la notion de Point de Vue de Référence (PVR) composé de Concepts de Référence (CR) décrivant les types d'éléments pouvant appartenir à une Vue de Référence (VR).

Dans cette section, nous définissons tout d'abord la notion d'attaquant dans le domaine de la modélisation et l'analyse de la menace et nous identifions les préoccupations pertinentes pour la construction d'un point de vue d'attaquant. Par la suite, nous définissons un ensemble de CR décrivant ses préoccupations et nous construisons différents PVR.

3.3.1 Notion d'attaquant et identification de ses préoccupations

Nous définissons tout d'abord la notion d'attaquant dans notre contexte puis nous identifions les préoccupations communes à tous types d'attaquants.

3.3.1.1 Définition de la notion d'attaquant dans le domaine de la modélisation et l'analyse de la menace

Dans la section 2.1.2, nous mettons en avant l'absence de définition standardisée du concept d'attaquant. Dans cette sous-section nous proposons une définition du terme "attaquant" dans le domaine de l'analyse et de la modélisation de menaces. Les termes techniques, écrit en italique lors de leur première occurrence dans cette section, sont définis dans le glossaire (§6.3).

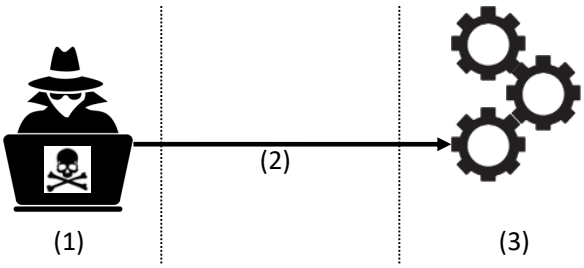


FIGURE 3.5 – Schémas d'un attaquant réalisant une attaque sur un système

Le dictionnaire définit le terme attaquant comme "une personne qui *attaque*", dans ce manuscrit, nous nous focalisons sur les attaques sur des *systèmes d'information*. Un attaquant est donc une personne qui attaque un système d'information. La figure 3.5 schématise cette définition, on peut y observer trois parties distinctes : une personne (1), une attaque (2) et un systèmes d'information (3).

Heckman [75] identifie neuf types d'attaquants que l'on peut regrouper en trois catégories comme illustrées dans le tableau 3.1. Dans notre définition "une personne" peut donc être un individu, une organisation ou encore un état.

Notre catégorisation	**Type d'attaquant de Heckman [75]**
Individu	Unstructured Hacker
	Structured Hacker
	Script Kiddie
	Hobbyist Hacker
	Insider (user/supervisor/admin)
Groupe / Organisation	Organized Crime / Industrial Espionage
	Unfunded Terrorist Group / Hacktivist
	Funded Terrorist Group
État / Gouvernement	Nation State

Tableau 3.1 – Catégorisation des types d'attaquant

Comme présenté dans la section 2.1.2, un attaquant se différencie d'un utilisateur normal par le fait qu'il possède des *compétences* qu'il met en œuvre lors d'une attaque. Pour aller plus loin, nous nous référerons à la norme ISO-27000 :2018 [9] afin de définir certains termes :

— Attaque [9] : tentative de détruire, de rendre public, de modifier, d'invalider, de voler ou d'utiliser sans autorisation un actif, ou de faire un usage non autorisé de celui-ci

— Compétences [9] : capacités à appliquer des connaissances et des aptitudes pour obtenir les résultats escomptés.

— Système d'information [9] : ensemble d'applications, services, actifs informationnels ou autres composants permettant de gérer l'information.

Ici un "actif" est la traduction du terme anglais "asset" signifiant un composant ou une ressource d'un système d'information. Un actif peut donc être un objet physique tel qu'un serveur ou un câble mais aussi un logiciel ou une donnée.

Dans la section 2.1 nous identifions la confidentialité, l'intégrité et la disponibilité comme étant les propriétés de sécurité les plus couramment utilisées. La définition de la notion d'attaque [9] peut être exprimée en terme de violation de propriétés de sécurité conformément aux associations présentées dans le tableau 3.2. Ce rapprochement nous

	Confidentialité	**Intégrité**	**Disponibilité**
Détruire		x	x
Rendre public	x		
Modifier	x	x	
Invalider		x	
Voler	x	x	x
Utiliser sans autorisation	x		x
Faire un usage non autorisé			x

Tableau 3.2 – Interprétation de la définition d'une attaque [9] en terme de violation de confidentialité, d'intégrité et de disponibilité.

permet de définir une attaque comme une tentative de nuire à la disponibilité, la confidentialité ou l'intégrité d'un actif.

Les compétences d'un attaquant regroupent un ensemble de connaissances et d'aptitudes. Les connaissances d'un attaquant sont composées de tout ce qu'il connaît d'un système, en particulier les composants du système, les relations entre ces composants et les acteurs interagissant avec le système. Comme présenté dans la section 2.1.2, les aptitudes d'un attaquant regroupent tout ce qu'il sait faire, à savoir sa capacité à exploiter différent types de *vulnérabilités* et à interagir avec le système.

Pour aller plus loin nous nous référerons, une fois encore, à la norme ISO-27000 :2018 [9] afin de définir les termes suivants :

— Vulnérabilité [9] : faille dans un actif ou dans une mesure de sécurité qui peut être exploitée par une ou plusieurs *menaces*.

— Menace [9] : cause potentielle d'un incident indésirable, qui peut nuire à un système ou à un organisme.

L'ensemble des définitions et des rapprochements présentés dans cette section nous permettent de proposer une définition d'un attaquant dans le domaine de la modélisation et de l'analyse de la menace.

Definition 1 - Attaquant, pour la modélisation et l'analyse de la menace.

Un individu, un groupe ou un état doté d'aptitudes à exploiter des vulnérabilités et qui met en œuvre ses connaissances sur les composants, les relations et les acteurs d'un système d'information, afin de nuire à la disponibilité, la confidentialité et/ou l'intégrité d'actifs de ce système.

Cette définition est utilisée dans la suite de ce manuscrit afin de définir un ensemble de préoccupations, de construire un point de vue et d'interpréter un comportement malveillant.

3.3.1.2 Identification des préoccupations d'un attaquant

La figure 3.6 présente une version schématisée de la définition donnée dans la section précédente. A partir de cette représentation, il est possible d'identifier trois types de

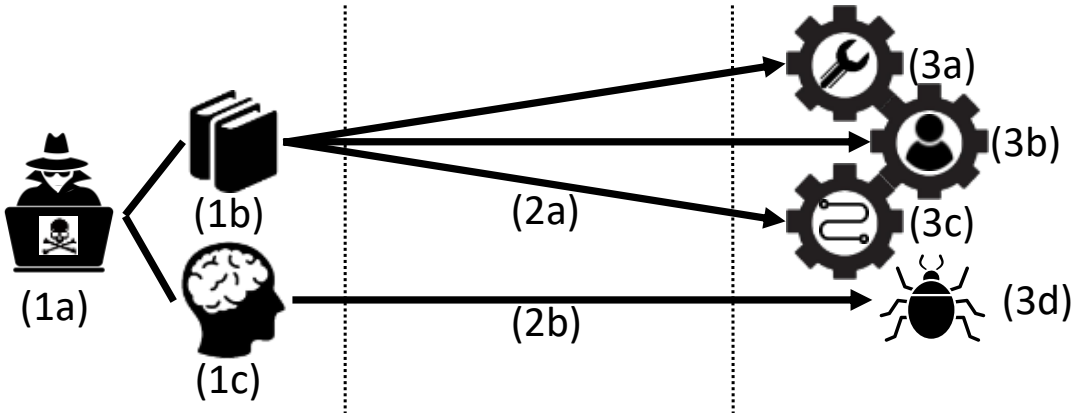


FIGURE 3.6 – Schéma raffiné d'un attaquant réalisant une attaque sur un système

préoccupations : la description d'un attaquant (1), la description d'interactions (2) et la description d'un système (3),. Ces types de préoccupation peuvent être détaillés à savoir :

(1.a) Une personne (individu, organisation ou état).

(1.b) Un ensemble de connaissances sur les composants, relations et acteurs d'un système.

(1.c) Un ensemble d'aptitudes à exploiter les vulnérabilités.

(2.a) Usage malveillant de composants, relations ou acteurs d'un système.

(2.b) Exploitation de vulnérabilités présentes sur un système.

(3.a) Les composants d'un système.

(3.b) Les relations d'un système.

(3.c) Les acteurs d'un système.

(3.d) Les vulnérabilités d'un système.

Un attaquant possède une identité (1.a), des connaissances portant sur les différents constituants d'un système (1.b) et des aptitudes lui permettant d'exploiter certains types de vulnérabilités (1.c).

Nous identifions deux types d'interactions, une utilisation malveillante des constituants d'un système (2.a) et l'exploitation de vulnérabilité (2.b). Un attaquant interagit avec le système avec un objectif. Ce dernier peut être précis, tel que le vol d'informations ou imprécis, l'attaquant est alors opportuniste.

Un système contient des composants (3.a), des relations (3.b), des acteurs (3.c) et des vulnérabilités (3.d). Un composant possède une configuration, des relations et éventuellement des vulnérabilités. Les relations modélisent les échanges entre des composants mais aussi les interactions avec des acteurs. Un acteur agit sur le système, pour ce faire, il bénéficie de droits et d'accès particuliers. Une vulnérabilité est une faille dans un composant du système qui peut être exploitée par un attaquant.

On identifie deux types de comportements, ce que l'attaquant cherche à faire, que l'on appellera *comportement malveillant* et ce que le système cherche à faire, appelé *comportement nominal*. Le comportement nominal est composé d'un ensemble d'actions définies comme normales pour un système, on parle alors d'*action nominale*. Elles peuvent avoir un impact sur la configuration, la topologie, les droits et les accès d'un élément du système. Par exemple, la suite d'actions suivantes est un comportement nominal pour le système fil rouge : extinction de Carol, envoi d'un message d'Alice à Bob et mise à jour de la version de WordPress utilisée par Bob et Carol. Le comportement malveillant est composé d'actions nominales et d'*actions malveillantes*. Les actions malveillantes sont le résultat d'exploitation de vulnérabilités. Elles permettent de modifier les connaissances, les accès et les droits d'un attaquant. Par exemple, dans le système fil rouge, la connexion de Mallory avec Alice et Bob (étape e1.1) est une action nominale effectuée par l'attaquant et l'élévation de privilège sur la machine de Bob (étape e2.2) est une action malveillante effectuée par l'attaquant.

La carte mentale schématisée en figure 3.7 représente l'ensemble des préoccupations d'un attaquant dans un contexte de modélisation et d'analyse de menaces.

Dans cette carte on peut identifier plusieurs occurrences du concept de vulnérabilité, l'attaquant s'intéresse d'une part aux éléments vulnérables mais aussi aux types de vulnérabilités et à leurs exploitations.

Tout comme l'approche ADVISE [109] présentée en section 2.1.3, nous dissocions les préoccupations décrivant l'attaquant de celles décrivant le système. Cette distinction permet, pour un même système, de comparer différents types d'attaquants et de réutiliser une description d'attaquant pour différents systèmes. Dans notre approche, contrairement à [109], nous distinguons également les aspects structurels et des aspects comportementaux. Cela permet d'une part d'effectuer des analyses indépendantes des comportements telles que la détection d'une surface d'attaque et d'autre part, d'étudier les effets d'un changement de comportement ou de stratégie d'attaque.

3.3.2 Définition des concepts de référence

Dans la section précédente (§3.3.1), nous avons identifié les préoccupations d'un attaquant regroupées en trois groupes : description d'un système, description d'un attaquant et description de comportements.

Dans cette section, à travers des modèles conceptuels, nous formalisons un ensemble de Concepts de Références (CR) permettant de construire des Points de Vue de Référence (PVR). La description des vulnérabilités étant primordiale dans notre domaine d'étude, nous lui consacrons une sous-section spécifique. Par la suite, nous définissons, au cours de trois sous-sections, les CR permettant de modéliser les préoccupations identifiées. Pour finir, nous présentons quatre PVR permettant la modélisation et l'analyse de la menace du point de vue d'un attaquant.

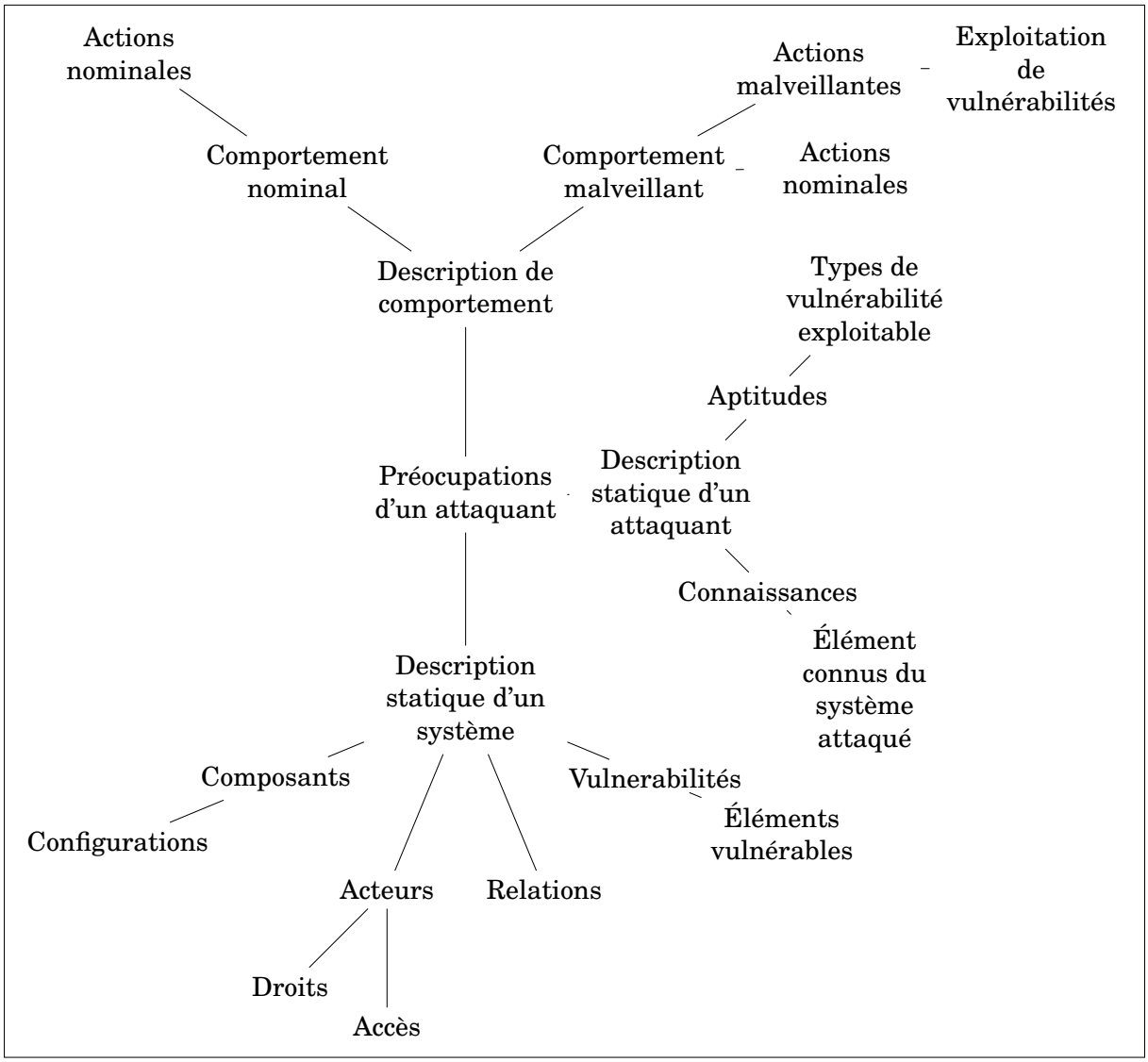


FIGURE 3.7 – Carte mentale (*mind map*) des préoccupations d'un attaquant dans un contexte de la modélisation et de l'analyse de menaces

3.3.2.1 Concepts de Référence liés à la description de vulnérabilité

Une vulnérabilité est une faille dans un actif qui peut être exploitée afin de nuire à un système (§ 3.3.1). Elle est alors relative à "un actif", c'est-à-dire à un service, une application ou plus largement un élément d'un système. De plus, elle "peut être exploitée" dans le but de "nuire à un système", ce qui signifie qu'une vulnérabilité a des prérequis d'exploitation et des impacts. Le CR modélisant une vulnérabilité doit alors prendre en compte ces trois facteurs :

— L'actif vulnérable, c'est-à-dire l'application ou le service qui contient la vulnérabilité.

— Les prérequis d'exploitabilité, c'est-à-dire les ressources nécessaires à un attaquant pour exploiter la vulnérabilité.

— Les impacts de l'exploitation, c'est-à-dire les modifications apportées au système suite à l'exploitation de la vulnérabilité.

Une vulnérabilité possède également un identifiant CVE, comme présenté dans la section 2.1.2 CVE [1] est le standard utilisé pour identifier les vulnérabilités. L'identifiant est donné sous la forme "CVE-année-numéro", par exemple CVE-2014-10021 est une vulnérabilité découverte en 2014 portant le numéro 10021.

Dans notre approche, une vulnérabilité est alors modélisée comme un élément contenant un identifiant, des actifs vulnérables, des prérequis et des impacts. La figure 3.8 présente les CR permettant de modéliser les préoccupations d'un attaquant liées aux vulnérabilités.

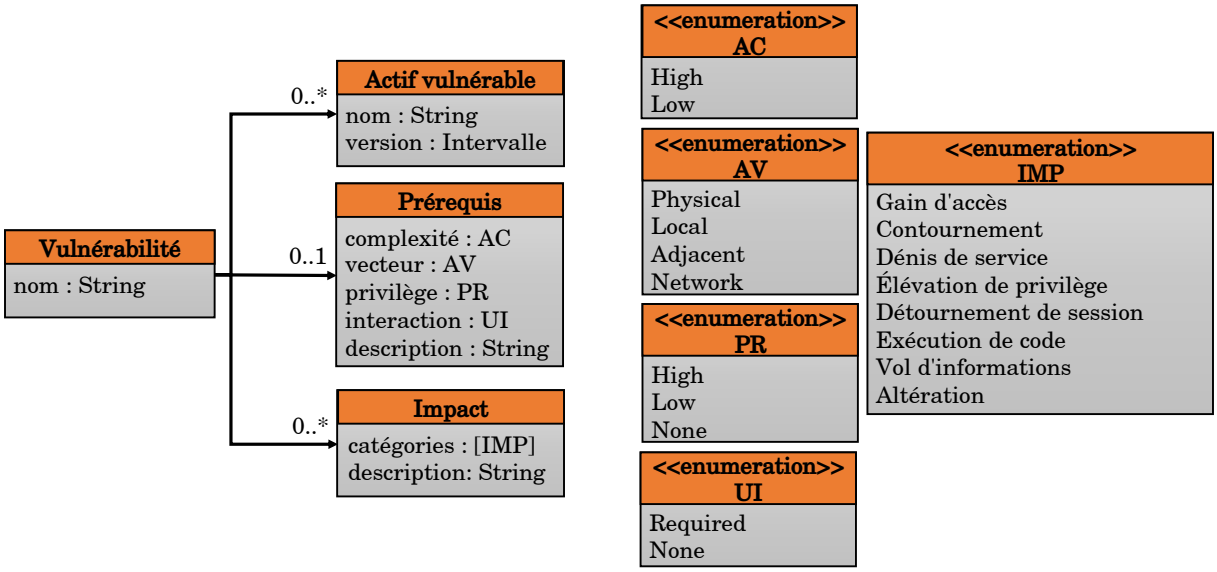


FIGURE 3.8 – Métamodèle du concept de vulnérabilité

Notre solution de modélisation permet à une vulnérabilité d'être présente sur plusieurs actifs et de déclencher différents impacts. Cependant, elle ne permet pas à une même vulnérabilité de posséder plusieurs identifiants ou prérequis. Nous effectuons ce choix de modélisation car les standards de cybersécurité (§2.1.2) se basent sur un identifiant unique et ne définissent qu'un seul ensemble de prérequis par vulnérabilité. Une faille, admettant deux ensembles de prérequis distincts, sera alors modélisée par deux vulnérabilités possédant des identifiants différents.

Nous modélisons un actif vulnérable comme un tuple composé d'un nom et d'une version.

Nous modélisons les prérequis d'exploitation d'une vulnérabilité comme un ensemble de besoins devant être satisfaits afin qu'un attaquant puisse exploiter une vulnérabilité. Un prérequis est composé de quatre attributs, extraits des métriques d'exploitabilité définies par CVSS [3], définis comme :

— La complexité d'attaque (Attack Complexity), noté AC. Elle représente les compétences techniques nécessaires à un attaquant permettant d'exploiter une vulnérabilité. La complexité d'attaque peut être de deux niveaux : faible (Low), noté AC:L, ou forte (High), noté AC:H. Seuls les attaquants possédant des connaissances techniques avancées peuvent exploiter des vulnérabilités complexes.

— Le vecteur d'attaque (Attack Vector), noté AV. Ce vecteur représente la distance entre un attaquant et sa cible. La cible est ici l'élément du système contenant l'actif vulnérable. Un vecteur d'attaque peut prendre quatre valeurs distinctes :

  — Physique, notée AV:P. Un attaquant doit accéder physiquement à sa cible, c'est à dire la toucher et/ou la manipuler pour pouvoir exploiter la vulnérabilité.

  - Local, noté AV:Loc. Un attaquant doit se connecter localement à sa cible (par exemple via SSH) pour pouvoir exploiter la vulnérabilité.
  - Adjacent, noté AV:A. Un attaquant doit avoir un accès local à un élément adjacent de sa cible (par exemple une imprimante connectée en bluetooth à la cible) pour pouvoir exploiter la vulnérabilité.
  - Réseau (Network), noté AV:Net. Un attaquant doit se connecter au réseau contenant sa cible pour pouvoir exploiter la vulnérabilité.
- Les privilèges nécessaires (Privileges Required), noté PR. Ces privilèges représentent les droits qu'un attaquant doit avoir sur la cible avant de pouvoir exploiter la vulnérabilité. Ils sont classés en trois types : haut niveau (High) noté PR:H, bas niveau (Low) noté PR:L et aucun privilège requis (None) noté PR:N. Par exemple, dans un réseau d'entreprise, un haut niveau de privilège peut être un compte administrateur, un bas niveau peut être un compte utilisateur et aucun privilège requis peut être un compte invité.
- Le besoin d'interactions avec un utilisateur (User Interaction), noté UI. Ces interactions représentent le fait qu'un utilisateur légitime du système doit effectuer une action particulière, comme ouvrir un email ou exécuter une application. Ces interactions peuvent être nécessaires (Required), noté UI:R, ou non nécessaires (None), noté UI:N.

Les impacts d'une vulnérabilité sont un ensemble de modifications apportées au système. Nous classons ici les impacts selon 8 catégories, à savoir :

- Gain d'accès : permet à un attaquant d'accéder à de nouveaux composants d'un système par exemple via l'exploitation d'une faille de type *XSS*.
- Contournement : permet à un attaquant d'éviter certaines contraintes d'un système, en particulier les éléments de sécurité.
- Dénis de service : rend indisponible un service, empêchant son utilisation par des utilisateurs légitimes.
- Élévation de privilège : permet à un attaquant d'obtenir de nouveaux droits sur un élément d'un système.
- Détournement de session : permet à un attaquant d'usurper l'identité, les droits et les accès d'un utilisateur légitime.
- Exécution de code : l'attaquant peut exécuter du code arbitraire sur une cible, cela peut lui permettre d'élever ses privilèges, de subtiliser des informations ou encore de nuire au système. L'exécution de code peut résulter d'un dépassement de mémoire, de l'installation d'un logiciel malveillant ou encore d'une erreur de configuration.
- Vol d'informations : l'attaquant subtilise, obtient ou divulgue de nouvelles informations portant sur les composants, les acteurs ou des données d'un système tel que des mots de passes ou des fichiers sensibles.
- Altération : permet à un attaquant de créer, supprimer, consommer, bloquer, modifier ou corrompre des données ou des ressources d'un système. Cela lui permet par exemple de modifier la configuration d'un composant.

Il existe de nombreuses méthodes permettant de typer des vulnérabilités, via leurs impacts, leurs causes, leurs sévérités, etc. Nous présentons dans l'annexe A une correspondance entre notre catégorisation par impacts et celle par types définie par CVE dans [2]. Cette correspondance permet pour chaque type de vulnérabilité d'identifier les impacts les plus courants.

Comme présenté en section 3.2, le système fil rouge compte quatre vulnérabilités : CVE-2014-10021, CVE-2018-9035, CVE-2016-5195 et CVE-2017-8584. Le tableau 3.3 modélise ces vulnérabilités conformément au CR présenté précédemment. Les données du

Identifiant	Actif vulnérable	Prérequis	Impacts
CVE-2014-10021	Symposium Version <= 14.11	AV:Net AC:L PR:N UI:N	Gain d'accès Exécution de code Vol d'informations
CVE-2018-9035	Contact Form 7 to Database Version = 2.10.32	AV:Net AC:L PR:N UI:R	Exécution de code
CVE-2016-5195	Linux kernel Version $\in [2.0; 4.8.3]$ Ubuntu Version <= 16.10	AV:Loc AC:L PR:L UI:N	Élévation de privilège
CVE-2017-8584	Windows 10 Version = 1607	AV:A AC:H PR:N UI:N	Gain d'accès Exécution de code Vol d'informations

Tableau 3.3 – Modélisation des vulnérabilités du système fil rouge conformément aux CR associés

tableau 3.3 proviennent des standards de cybersécurité présentés dans 2.1.2. Les identifiants proviennent de CVE [1], les actifs vulnérables et les impacts sont déduits des informations contenues dans la base de données NVD [6], et les prérequis utilisés sont extraits de CVSS [3]. Ces données permettent de modéliser une vulnérabilité selon les préoccupations d'un attaquant. Par exemple, la vulnérabilité *Dirty COW* possède l'identifiant CVE-2016-5195. Lors de l'étape e2.2 de l'exemple fil rouge, Mallory utilise cette vulnérabilité afin d'obtenir des droits administrateurs sur la machine Bob. Elle est présente sur les systèmes d'exploitation utilisant un noyau Linux dont la version est comprise entre 2.0 et 4.8.3, ce qui inclut toutes les versions Ubuntu postérieur à 16.10. Elle permet à un attaquant d'élever ses privilèges et requiert un accès local (AV:Loc) et des privilèges limités (PR:L). De plus, elle est exploitable par des attaquants peu qualifiés (AC:L) et ne demande pas d'interactions avec un utilisateur (UI:N).

3.3.2.2 Concepts de Référence liés à la description d'un système

Dans la section précédente (§3.3.1) nous précisons que les préoccupations d'un attaquant concernant le système qu'il attaque portent sur les composants, les acteurs et les relations entre ces derniers. Dans cette section, nous proposons des modèles conceptuels définissant ces différents éléments.

Composants

Un système contient des composants tel que des routeurs, des serveurs, des postes de travail, etc. Pour mener à bien son attaque, un attaquant s'intéresse aux configurations de ces éléments et à leurs interactions. Les interactions sont décrites dans la partie relative aux relations.

Nous définissons une configuration comme un ensemble de produits possédant un nom et un numéro de version. La figure 3.9 présente les CR utilisés. Un composant contient une unique configuration regroupant différents produits. Un produit possède un type, un nom, un numéro de version et éventuellement un opérant. L'opérant représente l'environnement informatique dans lequel le produit fonctionne. Par exemple Symposium est une extension

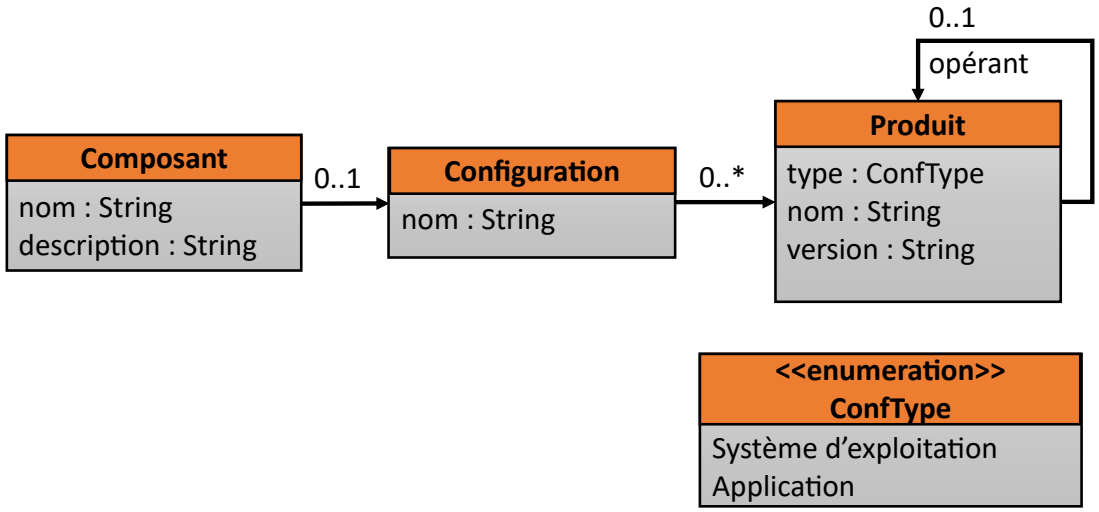


FIGURE 3.9 – CR permettant de modéliser les préoccupations d'un attaquant liées aux configurations des composants d'un système.

de WordPress, dans ce cas, l'opérant de Symposium est WordPress. Le type permet d'identifier si le produit est une application ou un système d'exploitation. Le nom et le numéro de version permettent de référencer clairement le produit.

Comme présenté en section 3.2 le système fil rouge est constitué d'Alice, Bob et Carol, utilisant deux systèmes d'exploitation et trois applications distinctes. Ce système contient donc trois composants et cinq produits. Le tableau 3.4 représente les configurations des éléments du système fil rouge conformément au CR de la figure 3.4.

(a) Configuration d'Alice

Type	Nom	Version	Opérant
Système d'exploitation	Windows 10	1607	-
Application	WordPress	4.5.7	Windows 10
Application	CF7tD	2.10.32	WordPress

(b) Configuration de Bob et de Carole

Type	Nom	Version	Opérant
Système d'exploitation	Ubuntu	16.04 LTS	-
Application	WordPress	4.5.7	Ubuntu
Application	CF7tD	2.10.32	WordPress
Application	Symposium	14.10.3	WordPress

Tableau 3.4 – Configuration des composants du système fil rouge

Ce tableau permet, par exemple, d'identifier que Bob utilise l'application Symposium en version 14.10.3, une extension de WordPress. Cette information est découverte par Mallory lors de l'étape e1.2 du scénario de l'exemple fil rouge et elle permet à l'attaquant d'apprendre que Bob contient la vulnérabilité CVE-2014-10021.

Il est à noter qu'au cours d'une attaque la configuration d'un système peut changer. Par exemple, lors de l'étape e3.1 Mallory utilise la machine Bob afin de scanner le réseau. Dans les faits, Mallory installe sur la machine de Bob un logiciel nommé *Nmap* lui permettant de réaliser un scan du réseau. Au cours de cette étape, la configuration de Bob est donc modifiée. Le tableau 3.4 représente donc l'état de la configuration à un instant donné (ici avant l'étape e3.1).

Acteurs

Un acteur interagit avec un système, pour ce faire il possède des accès et des droits particuliers. L'objectif de cette partie est de définir les CR permettant de modéliser les préoccupations d'un attaquant sur les accès et les droits des acteurs d'un système.

Dans la littérature, il existe un grand nombre d'approches permettant de définir des accès et des droits sur un système. Par exemple, Yongzheng and Xiaochun [168] utilisent la notion de privilège regroupant, en un seul objet, les notions d'accès et de droits, alors que le standard CVSS [3] définit deux notions distinctes : un vecteur d'attaque et des privilèges requis.

Dans ce manuscrit, nous dissocions les notions de droits et d'accès et décomposons ces dernières en accès physique et accès réseau. La figure 3.10 présente les CR permettant de modéliser les préoccupations d'un attaquant liées aux droits et accès d'un acteur.

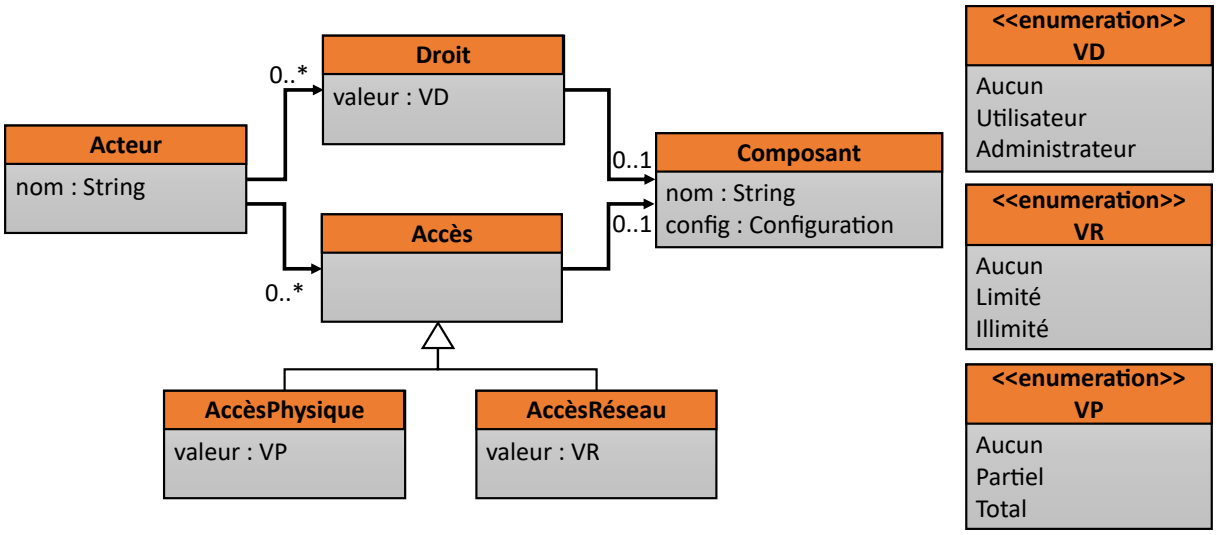


FIGURE 3.10 – CR permettant de modéliser les préoccupations d'un attaquant liées aux droits et accès d'un acteur.

Un acteur possède donc un ensemble de droits et d'accès relatifs à un composant d'un système. Les droits d'utilisation modélisent le niveau de privilège d'un acteur sur un composant. Nous prenons en compte trois types de droits d'accès :

— Aucun : l'acteur n'a aucun droit.

— Utilisateur : l'acteur a des droits limités lui permettant d'effectuer des tâches simples.

— Administrateur : l'acteur a tous les droits et peut effectuer tous types d'action.

Les accès réseaux modélisent la possibilité pour un acteur de se connecter à un système, à un sous-réseau ou à un composant spécifique. Nous prenons en compte trois types d'accès réseau :

— Aucun : l'acteur ne peut pas communiquer avec l'élément.

— Limité : l'acteur peut communiquer avec l'élément via des protocoles adaptés, par exemple un client se connectant en HTTP sur un serveur.

— Illimité : l'acteur se connecte directement à l'élément, par exemple, via un bureau distant ou une connexion SSH.

Les accès physiques modélisent la possibilité pour un acteur d'accéder à un bâtiment, un bureau ou une machine. Nous prenons en compte trois types d'accès physiques :

— Aucun : l'acteur ne peut ni manipuler ni interagir avec l'élément.

— Partiel : l'acteur peut interagir avec l'élément, par exemple via Bluetooth, mais pas le manipuler physiquement.

— Total : l'acteur peut manipuler physiquement l'élément.

Les CR définis ici permettent de modéliser finement chaque acteur d'un système. La distinction entre les concepts d'accès et de droits correspond aux notions de vecteur d'attaque et de privilège requis utilisés par le standard CVSS. Le raffinement de la notion d'accès permet de distinguer les attaques via le réseau des attaque physiques. Les trois exemples suivants permettent d'illustrer les différences entre les notions présentées ici :

— L'exploitation d'une vulnérabilité ayant comme impact "gain d'accès" permet à un attaquant d'obtenir de nouveaux accès réseau sur sa cible mais cela ne change ni ses droits ni ses accès physiques.

— L'exploitation d'une vulnérabilité ayant comme impact une "élévation de privilèges" permet à un attaquant d'obtenir de nouveaux droits sur sa cible mais cela ne change pas ses accès.

— L'exploitation d'une vulnérabilité ayant comme prérequis "AV :P" n'est possible que si l'attaquant peut accéder physiquement à sa cible, c'est le cas par exemple lors d'une attaque provenant de l'intérieur d'un système (*insider attack*). Quelques soient les droits et les accès réseau de l'attaquant, si il ne peut pas toucher sa cible, alors, il ne pourra pas exploiter cette vulnérabilité.

Comme présenté dans la section 3.2, le système fil rouge compte quatre acteurs, trois utilisateurs (Alice, Bob et Carole) et un attaquant (Mallory). Comme l'illustre le tableau 3.5, chaque acteur possède des droits et des accès sur les différents composants du système.

	Alice			Bob			Carole			Mallory		
	D	AR	AP	D	AR	AP	D	AR	AP	D	AR	AP
Alice	◐	●	●	◐	◐	●	◐	◐	●	○	○	○
Bob	◐	◐	●	◐	●	●	◐	◐	●	○	○	○
Carole	◐	◐	●	◐	◐	●	◐	●	●	○	○	○
Mallory	○	○	○	○	○	○	○	○	○	●	●	●

D : Droits AR : Accès réseau AC : Accès physique
○ : Aucun ◐ : Utilisateur / Limité / Partiel ● : Administrateur / Illimité / Total

Tableau 3.5 – Droits et accès des différents acteurs du système fil rouge

Dans notre exemple, chaque utilisateur légitime possède des droits utilisateur, un accès réseau illimité et un accès physique total sur sa propre machine. On peut noter que, dans le tableau 3.5, l'attaquant ne possède aucun droit ni aucun accès aux éléments du système, hormis à sa propre machine (Mallory). Cependant, au cours de son attaque, il peut obtenir de nouveaux accès et droits, par exemple, au cours l'étape e2.2, en exploitant la vulnérabilité CVE-2016-5195, il obtient des droits administrateur sur Bob. Le tableau 3.5 représente

donc l'état des droits et accès des acteurs du système fil rouge à un instant donné (ici avant l'attaque).

Relations

La topologie d'un système est la définition des liens entre les éléments du système. Modéliser les relations entre les composants d'un système revient donc à définir la topologie de ce dernier. L'objectif de cette partie est de définir les CR permettant de modéliser les préoccupations d'un attaquant sur la topologie d'un système.

Il existe de nombreux types de relations entre deux éléments, tels que le contrôle, la dépendance énergétique, l'échange d'information, etc. Notre domaine d'étude étant la modélisation et l'analyse de menaces sur des systèmes d'information, nous nous focalisons sur la modélisation de l'architecture réseau d'un système.

La figure 3.11 présente les CR permettant de modéliser les préoccupations d'un attaquant liés à la topologie d'un système.

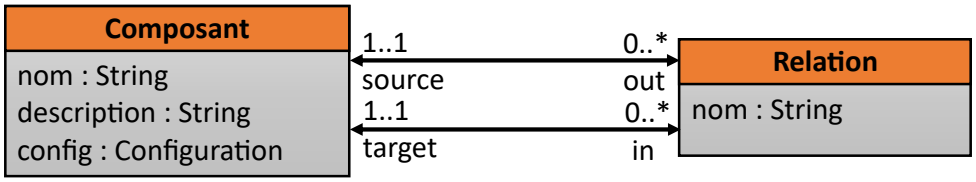


FIGURE 3.11 – CR permettant de modéliser les préoccupations d'un attaquant liées à la topologie d'un système.

Une relation lie donc deux composants, une "source" et une "cible". Comme présenté dans la section 3.2, la topologie du système fil rouge est composé d'un réseau de trois machines relié via internet à un attaquant. La figure 3.12 représente cette topologie.

Au cours d'une attaque, les connaissances d'un attaquant sur la topologie d'un système évoluent, il découvre de nouveaux éléments et de nouvelles relations. En général, la topologie globale d'un système n'évolue pas au cours d'une attaque, seul ce que l'attaquant connaît de cette dernière évolue. Cependant durant certains types d'attaques, en particulier les attaques physiques, un attaquant peut être amené à modifier la topologie d'un système, par exemple en détruisant un câble ou un équipement réseau.

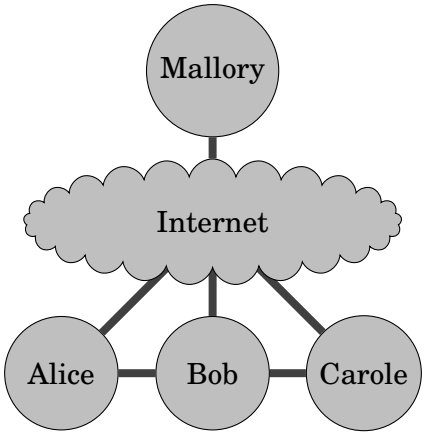


FIGURE 3.12 – Topologie du système fil rouge

3.3.2.3 Concepts de Référence liés à la description statique d'un attaquant

Les préoccupations relatives à la description d'un attaquant dans le domaine de la modélisation et l'analyse de menaces portent sur les aptitudes et les connaissances du système que possède un attaquant.

Aptitudes

Dans la section 2.1.2, nous regroupons différentes approches de description d'un attaquant en cinq catégories : utilisation d'un niveau global, création de types, création de profils, création d'un persona et utilisation d'attributs. Le standard de cybersécurité CVSS utilise un niveau global pour décrire les aptitudes d'un attaquant. Dans ce manuscrit, nous raffinons cette approche en utilisant la notion d'attribut. Ce choix de modélisation permet de spécifier les capacités d'un attaquant en fonction d'un ensemble de critères, tout en restant compatible avec les standards. En effet, il est toujours possible de calculer un niveau moyen à partir d'un ensemble d'attributs et de définir alors le niveau global en fonction de la moyenne calculée.

Parmi les approches présentées dans l'état de l'art (§2.1), en particulier dans les travaux de [59], les attributs d'un attaquant les plus souvent référencés sont : compétence, ressource, intention et motivation. Dans cette partie, nous nous focalisons sur l'aspect statique d'un attaquant, à savoir ses compétences et ses ressources.

La figure 3.13 présente les CR permettant de modéliser les compétences et les ressources d'un attaquant.

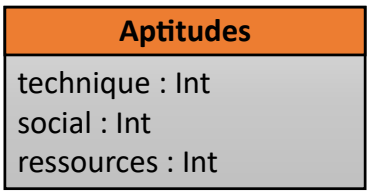


FIGURE 3.13 – CR permettant de modéliser les préoccupations d'un attaquant liées à la représentation de ses aptitudes.

La distinction entre compétences techniques et sociales fait référence aux prérequis "AC" et "UI" présentés dans la partie relative à la description de vulnérabilités. Les aptitudes d'un attaquant sont finalement représentées par trois attributs auxquels nous assignons une valeur entre 1 et 10. Une valeur élevée en technique reflète une capacité à exploiter des vulnérabilités complexes, une valeur proche de 10 en social reflète une capacité à utiliser des techniques d'ingénierie sociale avancées et une valeur élevée en ressources reflète une capacité à utiliser des outils complexes ou nécessitant de grandes puissances de calcul.

Dans l'exemple illustré en figure 3.14, nous définissons Mallory comme un attaquant possédant un haut niveau technique (8/10), un faible niveau de compétences sociales (3/10) et peu de ressources (2/10). Suivant cette description, Mallory a la capacité d'exploiter des

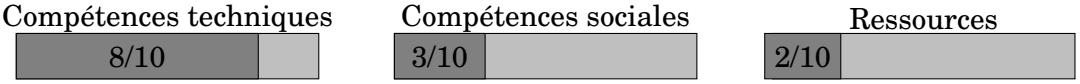


FIGURE 3.14 – Aptitudes de l'attaquant de l'exemple fil rouge

vulnérabilités demandant des connaissances techniques mais ne peut pas exploiter de vulnérabilités demandant de l'ingénierie sociale (*social engineering*) ou des ressources importantes tels que des supercalculateurs. Les vulnérabilités présentes sur le système fil rouge

sont présentées sur le tableau 3.3, dans notre exemple Mallory a les aptitudes nécessaires pour l'exploitation des vulnérabilités CVE-2014-10021, CVE-2016-5195 et CVE-2017-8584. La vulnérabilité CVE-2018-9035 demandant des interactions avec un utilisateur (UI:R), Mallory n'a pas, dans cet exemple, les aptitudes suffisantes pour l'exploiter.

On peut noter que les aptitudes d'un attaquant ne dépendent ni du déroulement d'une attaque ni du système attaqué, elles sont fixes pour un attaquant donné.

Connaissances

De nombreuses études utilisent les connaissances d'un attaquant, que ce soit pour la préservation de la confidentialité d'informations contenues dans des données anonymisées [118, 47, 23], pour le masquage de la position réelle d'un utilisateur [44] ou encore pour prévoir les prochaines actions d'un attaquant [109].

Les connaissances d'un attaquant sur un système comprennent tout ce qu'un attaquant connaît de ce système. L'objectif de cette partie est de définir les CR permettant de modéliser les préoccupations d'un attaquant liées à ses connaissances d'un système. La figure 3.15 présente ces CR.

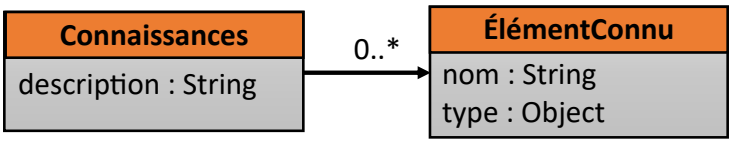


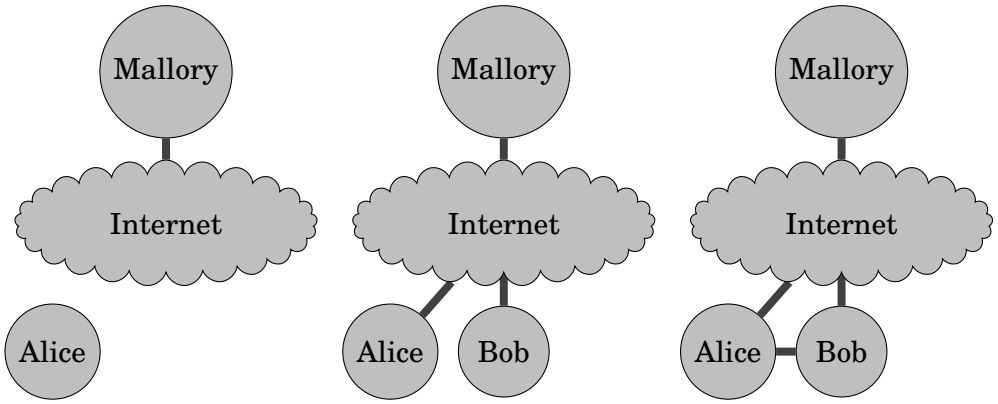
FIGURE 3.15 – CR permettant de modéliser les préoccupations d'un attaquant liées à ses connaissances d'un système.

Les connaissances d'un attaquant sont ici modélisées par un ensemble d'éléments connus. Dans notre approche, un système est modélisé par une topologie, des configurations, des droits et des accès. Un élément connu peut donc être une relation, un composant, une configuration ou un acteur. Ces derniers peuvent être connus totalement ou partiellement, par exemple un attaquant peut avoir connaissance de l'existence d'un composant mais ne connaître qu'une partie de ses relations ou de sa configuration.

Comme présenté dans la section 3.2, les connaissances de Mallory évoluent au cours de son attaque du système fil rouge. Par exemple, la figure 3.16 représente les connaissances de Mallory sur la topologie du système fil rouge lors des étapes e0, e1.1 et e3.1 de son attaque. Cette figure ce focalise sur les connaissances topologiques de Mallory mais il en va de même pour la configuration des composants ainsi que pour les droits et accès des acteurs du système. Sur cette exemple, Mallory a connaissance de l'existence d'Alice mais ne connaît ni Bob, ni Carol, ni aucune des relations du système. Au cours de son attaque, Mallory découvre Bob (étape e1.1) et la relation entre Alice et Bob (étape e3.1). On peut noter que Carol n'apparaît pas dans la figure 3.16, cela est dû au fait que Mallory ne découvre pas l'existence de Carol au cours de son attaque. Finalement, même une fois son attaque terminée (étapes e3.3), l'attaquant n'a qu'une connaissance partielle du système.

3.3.2.4 Concepts de Référence liée au comportement d'un système et d'un attaquant

Dans la section 3.3.1 nous identifions deux types de comportement, le comportement nominal d'un système, composé d'actions nominales et le comportement malveillant d'un attaquant, composé d'actions nominales et malveillantes. L'objectif de cette section est de définir les CR permettant de modéliser ces différents comportements. La figure 3.17 présente ces CR.



(a) Étape e0 (b) Étape e1.1 (c) Étape e3.1

FIGURE 3.16 – Evolution des connaissances de Mallory sur la topologie du système fil rouge au cours de son attaque

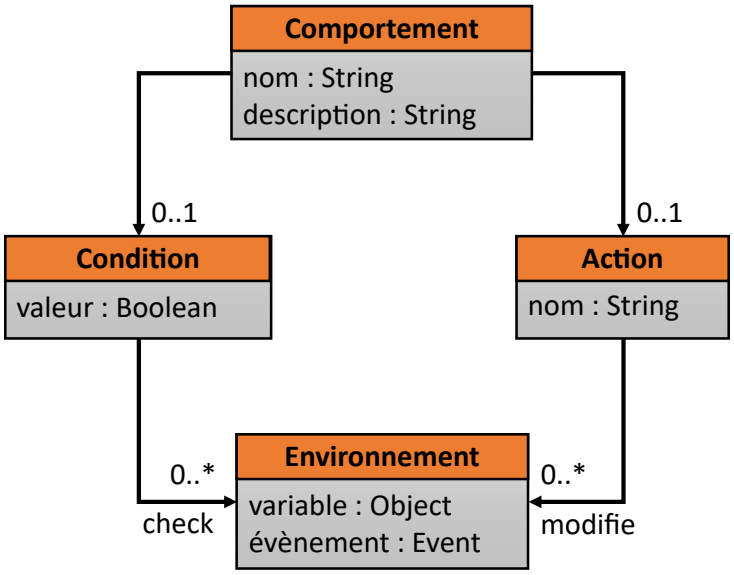


FIGURE 3.17 – CR permettant de modéliser les préoccupations d'un attaquant liées aux comportements.

Un comportement référence une condition et une action. Une condition est une assertion sur l'environnement et une action est une modification de cet environnement. Nous présentons en détail dans la section 4.3 en quoi ces conditions et actions sont liées au PVS et comment elles permettent d'interpréter le comportement d'un attaquant au cours d'une attaque.

Dans le système fil rouge, la connexion de Mallory avec Alice et Bob (étape e1.1) est une action nominale effectuée par l'attaquant et l'élévation de privilège sur la machine de Bob (étape e2.2) est une action malveillante. On peut ici modéliser deux comportements "mise en place d'une connexion" et "prise de contrôle". Le premier requiert que l'attaquant ait accès à internet et permette de créer de nouvelles relations. Le second est rendu possible via l'exécution d'une vulnérabilité, la condition correspond alors aux prérequis et l'action aux impacts de cette vulnérabilité (CVE-2016-5195).

Nous construisons en section 4.3 un interpréteur basé sur les notions de condition et d'action.

3.3.2.5 Construction des points de vue de référence

Dans la section 3.3.1, nous définissons un attaquant comme un individu, un groupe ou un état, doté d'aptitudes à exploiter des vulnérabilités et qui met en œuvre ses connaissances sur les composants, les relations et les acteurs d'un système d'information, afin de nuire à la disponibilité, la confidentialité et/ou l'intégrité d'actifs de ce système.

Au cours de cette section nous avons présenté les CR permettant de modéliser les préoccupations d'un attaquant relatives aux descriptions de vulnérabilités, de systèmes, d'attaquants et de comportements. Ces différents CR nous permettent de construire quatre PVR, à savoir "PVR vulnérabilités", "PVR système", "PVR attaquant" et "PVR comportement". Le premier décrit les préoccupations d'un attaquant relatives aux vulnérabilités (actif vulnérable, prérequis, impacts, etc.). Par construction ce PVR est indépendant du système modélisé et du type d'attaquant sélectionné. Le deuxième décrit les préoccupations d'un attaquant sur la topologie, les composants et les acteurs d'un système. Un système est alors modélisé comme un ensemble de composants interconnectés avec lesquels interagissent des acteurs. Le troisième, décrit un ensemble d'informations permettant de spécifier un attaquant à savoir ses aptitudes et ses connaissances. Le dernier PVR décrit les préoccupations d'un attaquant liées à son comportement et celui du système. La figure 3.18 présente les métamodèles de chacun de ces PVR.

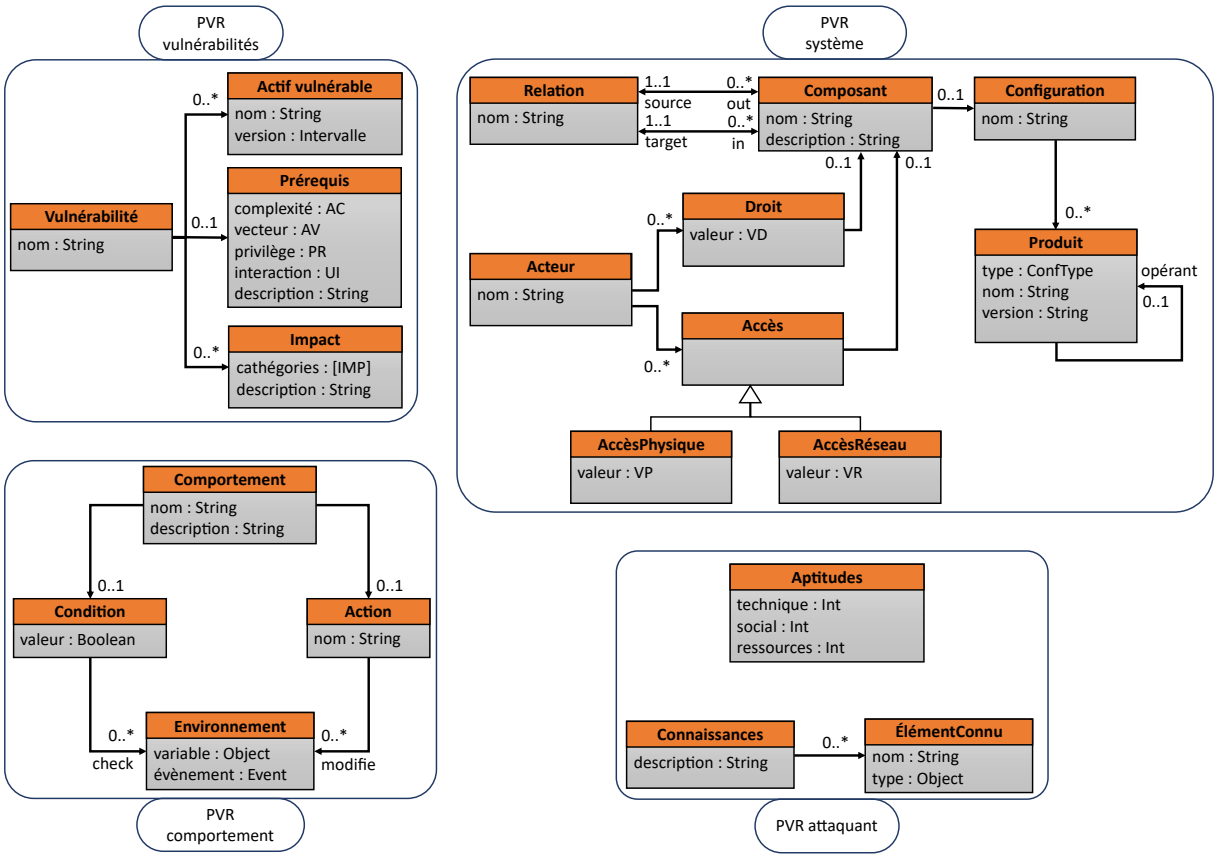


FIGURE 3.18 – Modélisation des différents points de vue de référence (PVR)

La création de quatre PVR distincts permet de décrire les vulnérabilités, le système, l'attaquant et les comportements indépendamment les uns des autres. Ces PVR peuvent être croisés et/ou spécifiés afin de décrire un point de vue spécifique. Il est alors possible de créer plusieurs vues sur un même système représentant ce que perçoivent différents types

d'attaquant, ou encore, d'étudier les effets de différents types de comportements lors d'une attaque.

Ces quatre PVR permettent, dans ce manuscrit, de couvrir l'ensemble des préoccupations propres à notre définition d'un attaquant dans le domaine de la modélisation et l'analyse de la menace. Cependant, la démarche proposée est adaptable à d'autres définitions, il suffit pour ce faire de modifier et/ou d'ajouter des PVR.

3.4 Spécification des points de vue pour un attaquant

Dans la section précédente (§3.3), nous avons défini la notion d'attaquant dans le contexte de la modélisation et l'analyse de la menace, nous avons également identifié ses préoccupations, modélisé un ensemble de CR et présenté les PVR utilisés dans ce manuscrit. Cette section se concentre sur la création de Points de Vue Spécifiques (PVS) et de Vues Spécifiques (VS). Une première partie expose les mécanismes de construction des PVS, une deuxième se focalise sur la modélisation de VS et la dernière énumère les fonctionnalités qu'une solution de description de points de vue doit couvrir pour permettre la modélisation de points de vue d'attaquants.

3.4.1 Construction de points de vue spécifiques

Dans la section 3.1.2, nous spécifions que les PVR permettent de modéliser les préoccupations communes à tout type d'attaquant et que les PVS permettent de les spécialiser pour un contexte particulier. Dans notre approche, les PVR sont utilisés pour mettre en forme les informations des modèles sources et les PVS permettent de représenter la perception qu'un attaquant se fait d'un système.

Dans cette section, nous décrivons les méthodes de construction des PVS, puis, nous présentons un exemple de modélisation de PVS, avec le point de vue de Mallory, sur le système fil rouge.

3.4.1.1 Méthode de construction des PVS

Notre approche se fonde sur la représentation d'un système selon les préoccupations d'un attaquant. Dans la section 3.3.2, nous définissons quatre PVR à savoir "PVR vulnérabilités", "PVR système", "PVR attaquant" et "PVR comportement".

Nous utiliserons dans la suite de ce manuscrit 3 PVS, représenté en figure 3.19, construits incrémentalement afin de représenter la perception d'un attaquant, à savoir :

— PVS système vulnérable : construit à partir des PVR décrivant les préoccupations d'un attaquant concernant le système et ses vulnérabilités. Ce PVS associe à chaque composant d'un système un ensemble de vulnérabilités.

— PVS système attaquable : construit à partir du PVS "système vulnérable" et du PVR "attaquant", il permet de représenter ce que l'attaquant perçoit du système.

— PVS système attaqué : construit à partir du PVS "système attaquable" et du PVR décrivant les préoccupations d'un attaquant sur les comportements nominaux et malveillants, il permet d'interpréter le comportement de l'attaquant.

Comme illustré en figure 3.19, nous définissons nos PVS de manière incrémentale, chaque point de vue étant plus spécifique que le précédent. Dans la section 4.3, nous utilisons le PVS système attaqué afin d'identifier des scénarios d'attaque.

Un PVS est constitué d'un ensemble de concepts spécifiques (CS). De part la conceptualisation de l'approche (§3.1), la grande majorité des CS sont définis via l'adaptation et

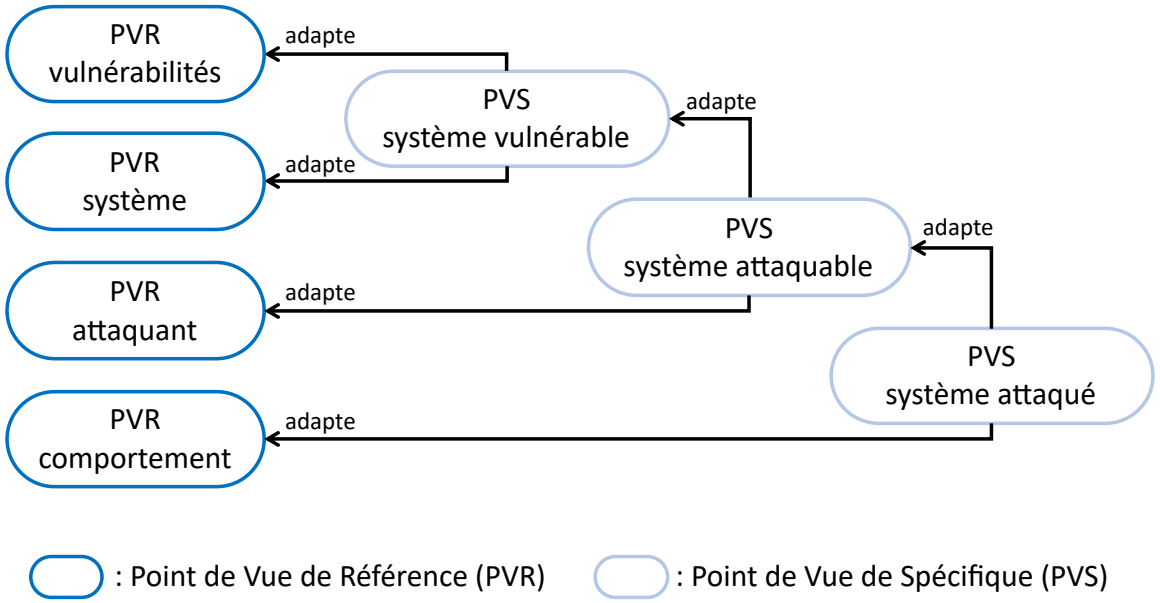


FIGURE 3.19 – Construction de points de vue spécifiques à partir des points de vue de références

l'interconnexion de CR. Cependant, nous permettons également la réutilisation de concepts définis pour d'autres PVS et la définition de nouveau concepts.

3.4.1.2 Construction d'un PVS représentant la perception de Mallory

Dans cette section, nous illustrons la construction de PVS via l'exemple du PVS de Mallory, l'attaquant du système fil rouge 3.2.

Nous considérons que Mallory décrit son point de vue selon ses intentions par rapport au système : *"Tous les composants du système fil rouge sont des machines interconnectées sur lesquelles sont installés des logiciels. Je porte un intérêt tout particulier aux logiciels comportant des vulnérabilités que je peux exploiter et aux machines sur lesquelles sont installées ces logiciels vulnérables. De plus, je souhaite faire la distinction entre les machines sous mon contrôle, les vulnérables et les autres. Pour finir, je souhaite connaître les dépendances entre les machines que je contrôle (j'ai le contrôle sur la machine B car je contrôle de la machine A)"*.

Nous proposons en figure 3.20 un métamodèle décrivant le PVS répondant aux exigences de Mallory. Ce point de vue est constitué de six concepts : *Machine*, *MachineVulné-*

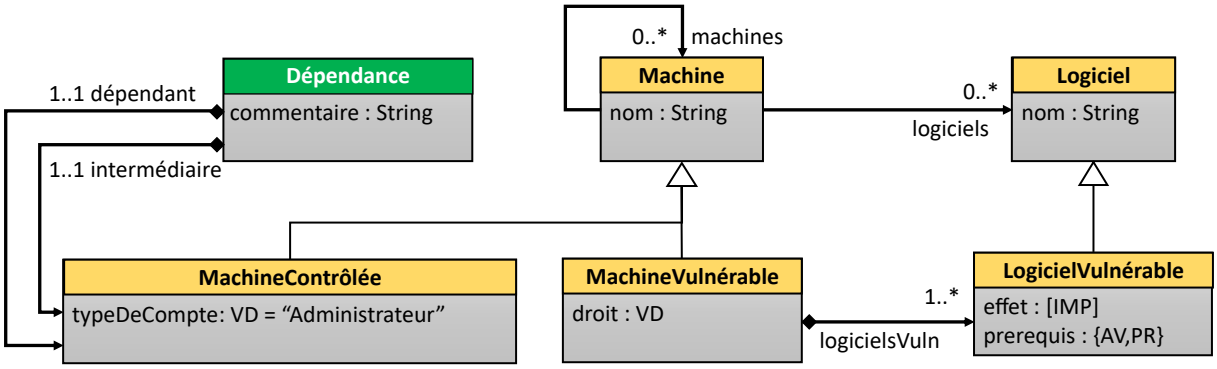


FIGURE 3.20 – PVS de Mallory

rable, *MachineContrôlée*, *Dépendance*, *Logiciel* et *LogicielVulnérable*. La référence entre deux *machines* permet de modéliser l'interconnexion entre les éléments du système fil rouge. La relation de contenance entre *Machine* et *Logiciel* modélise le fait qu'un *logiciel* est installé sur une *machine*. Les spécifications *MachineContrôlée*, *MachineVulnérable* et *LogicielVulnérable*, et le concept de *Dépendance*, permettent de respecter les exigences de Mallory. Pour finir, la relation de contenance du type 1..* entre *MachineVulnérable* et *LogicielVulnérable* modélise le fait qu'une *machine vulnérable* contient au moins un *logiciel vulnérable*.

Nous avons expliqué que les CS sont définis par une combinaison des CR auxquels peuvent s'ajouter de nouveaux concepts ou des concepts provenant d'autres PVS. Ici, comme le présente la figure 3.21, les CS constituant le point de vue de Mallory sont construits grâce à des combinaisons de CR et via la création d'un nouveau concept, celui de dépendance.

Les différent CS sont construits comme suit :

① Concept de *Logiciel* : un logiciel est un produit qui est connu de l'attaquant, il est défini à partir des concepts de produit provenant du PVR système et d'éléments connus provenant du PVR attaquant. Un logiciel contient l'attribut "nom" construit à partir du nom et de la version du produit ($logiciel.nom := produit.nom + "Version : " + produit.version$).

② Concept de *Machine* : ce CS est construit à partir de CR provenant des PVR système et attaquant ainsi que du CS *Logiciel*. Une machine représente un composant du système connu par l'attaquant et sur lequel il possède différents types de droits et d'accès. Une machine contient également un logiciel et des relations vers d'autres machines, ces dernières sont déduites du concept de relation entre deux composants.

③ Concept de *Logiciel vulnérable* : ce CS est construit via l'association des concepts de *Logiciel*, d'*Actif vulnérable*, de *Prérequis*, d'*Impact* et d'*Aptitude*. Un logiciel vulnérable est un logiciel sur lequel figure au moins une vulnérabilité exploitable par l'attaquant. Une vulnérabilité est dite exploitable si l'attaquant possède les aptitudes nécessaires à son exploitation.

④ Concept de *Machine contrôlée* : une machine contrôlée est une machine sur laquelle l'attaquant possède des droits administrateur.

⑤ Concept de *Machine vulnérable* : une machine vulnérable est une machine qui possède au moins un logiciel vulnérable et sur laquelle l'attaquant a des droits et des accès. En fonction de ces derniers, il pourra, ou non, exploiter l'une des vulnérabilités de la machine.

⑥ Concept de *Dépendance* : nous choisissons ici de construire ce concept indépendamment des CR ou d'autres CS. Une dépendance est utilisée afin de définir un type de relation particulier entre deux machines contrôlées. Cette relation permet de modéliser le fait que le contrôle d'un attaquant sur une machine est conditionné par son contrôle d'une autre machine. Ce concept est spécifique au point de vue de Mallory, c'est pourquoi il est défini manuellement pour un PVS précis et non pas comme un CR.

Le point de vue de Mallory modélisé en figure 3.20 représente de quelle manière Mallory perçoit le système fil rouge. L'objectif de Mallory étant de voler des informations à Alice via une attaque distante, il accorde une importance particulière aux logiciels présents sur le système et aux connexions entre les éléments du système. On peut noter que la vue construite à partir du PVS de Mallory ne modélise pas les éléments inconnus ou inaccessibles pour l'attaquant. Par ailleurs, seules les vulnérabilités exploitables par Mallory sont prises en compte. De plus, la notion de vulnérabilité n'apparaît pas directement

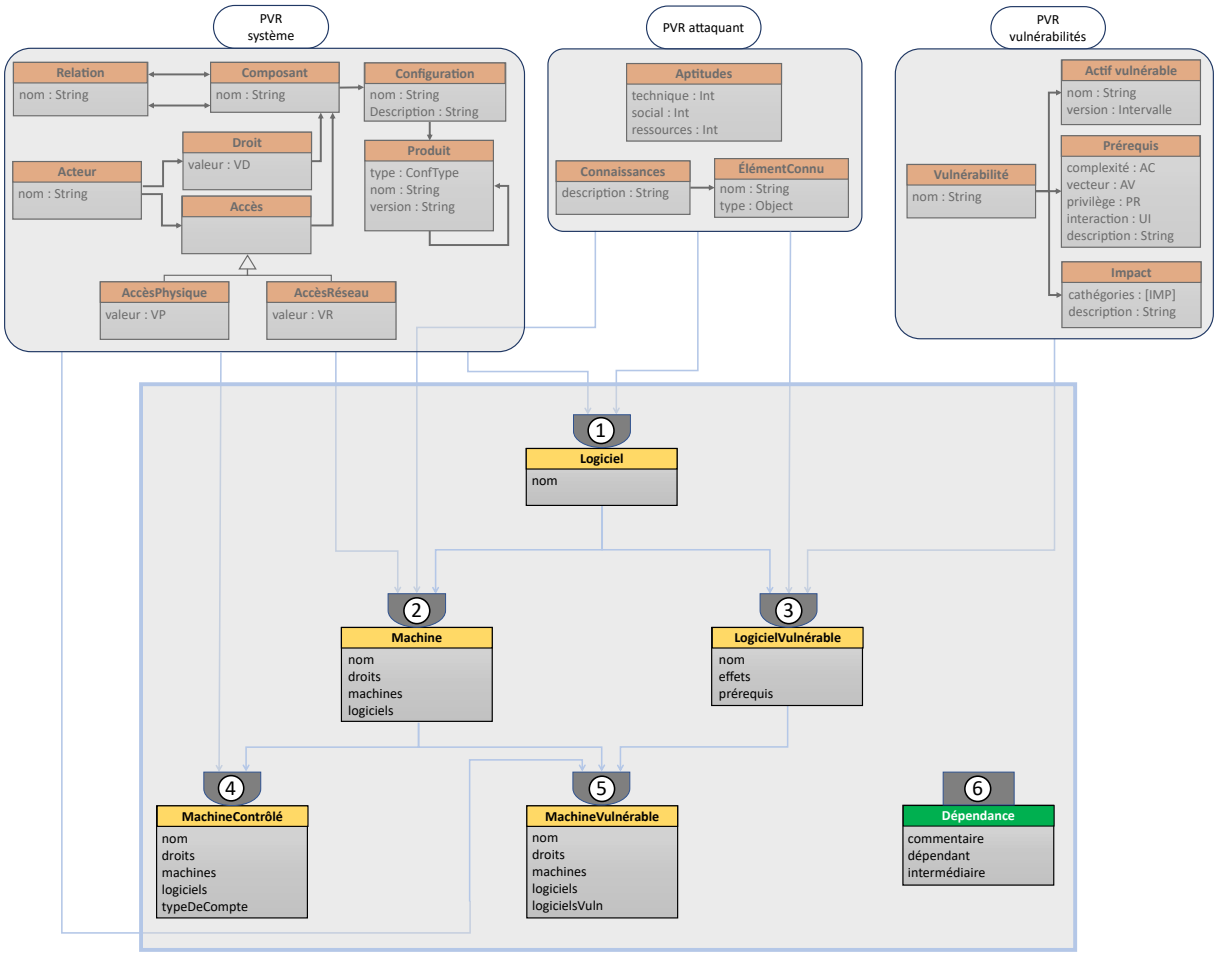


FIGURE 3.21 – Construction des concepts constituant le point de vue de Mallory

dans le PVS de Mallory, l'attaquant s'intéressant, dans cet exemple, uniquement aux effets de l'exploitation d'un logiciel vulnérable. Le point de vue modélisé dans cette section est alors spécifique à Mallory, un autre attaquant, avec d'autres préoccupations, aurait un PVS différent de celui exposé ici.

3.4.2 Construction de la vue d'un attaquant

Dans cette section nous décrivons les méthodes de construction de vue attaquant puis nous présentons un exemple de modélisation de la vue de Mallory sur le système fil rouge.

3.4.2.1 Méthode de construction d'une vue d'attaquant

Comme présenté dans la section 2.3.1, une vue est une représentation d'un système répondant à un ensemble de préoccupations décrites dans un point de vue. Dans un contexte de modélisation, une vue est conforme à un point de vue. Par extension, une Vue de Référence (VR) est conforme à un PVR, composé d'Instance de Référence (IR) et une Vue Spécifique (VS) est conforme à un PVS, composé d'Instance Spécifique (IS). Les informations contenues dans une vue peuvent être définies spécifiquement ou provenir de l'adaptation des informations contenues dans d'autre instances (modèle source, IR ou IS). Dans notre approche nous privilégions l'adaptation d'informations.

L'objectif est alors de mettre en forme les données de différentes sources pour une vue. L'adaptation est définie au niveau conceptuel et s'applique au niveau des vues. Dans notre approche, l'adaptation prend en compte les spécificités propres à la vue d'un attaquant, à savoir :

— L'hétérogénéité des sources d'informations : les données peuplant une vue proviennent de sources hétérogènes, l'adaptation doit permettre, au niveau instance, d'associer et de comparer des données de diverses sources.

— La variabilité des données : comme présenté dans la section 3.3, les modèles sources peuvent être modifiés au cours d'une attaque. Une vue étant une représentation d'un système, les données qu'elles contient doivent être cohérentes avec celles contenues dans les modèles sources. L'adaptation doit alors prendre en compte la variabilité des données.

— Les interactions d'un attaquant avec le système : de par son comportement malveillant, un attaquant apporte des modifications au système attaqué. Dans un contexte de modélisation, cela implique une interaction entre une vue et des modèles sources. Cette interaction requiert, en plus d'une adaptation allant des modèles aux vues, une adaptation allant des vues vers les modèles. La solution d'adaptation doit alors être bidirectionnelle.

Finalement, l'adaptation des données doit être bidirectionnelle et prendre en compte l'hétérogénéité des sources et la variabilité des données. Les données peuvent provenir des modèles sources, d'IR ou d'IS.

Nous avons présenté dans le chapitre 2 les approches de fédération de modèles et en particulier celles se basant sur la notion de rôle. Nous utiliserons cette approche en section 4.2 afin de permettre une adaptation pour la prise en compte de l'hétérogénéité des sources d'informations, la variabilité des données et les interactions de l'attaquant sur un système.

3.4.2.2 Construction d'une vue d'un attaquant

Dans cette section, nous présentons un exemple de construction de vue appliqué à Mallory sur le système fil rouge 3.2. Cet exemple illustre l'évolution de la vue d'un attaquant sur un système au cours d'une attaque. De plus, il permet également d'expliquer comment une vue est peuplée à partir de l'adaptation des informations contenues dans des VR.

Mallory voit le système fil rouge comme un réseau de machines contenant des logiciels. Comme présenté dans la section 3.4.1.2, le point de vue de Mallory est défini à l'aide d'adaptation des CR, seul le concept de Dépendance est défini manuellement. Les données peuplant la vue de Mallory proviennent alors d'IR. La figure 3.22 schématise la construction de la vue de Mallory sur le système fil rouge.

Nous avons montré précédemment que la vue d'un attaquant évolue au cours d'une attaque, dans notre exemple la vue de Mallory évolue durant chaque étape de son scénario d'attaque. La figure 3.23 présente l'évolution de cette vue au cours des huit étapes et sous-étapes de l'attaque du système fil rouge.

Dans cette représentation, on retrouve des instances des concepts de *Machine*, *Machine vulnérable*, *Machine Contrôlée*, *Logiciel*, *Logiciel Vulnérable* et *Dépendance*. Chaque concept est représenté par un pictogramme dont la signification est indiquée dans une légende. Lors de l'étape e0, Mallory n'a pas encore accédé au système mais sait qu'Alice en fait partie, c'est pourquoi sa vue est composée de deux éléments : Alice, modélisée ici par une machine, et Mallory, modélisée par une machine contrôlée. Cette modélisation reflète le fait que Mallory contrôle sa propre machine et qu'Alice n'est ni vulnérable ni contrôlée. A cette étape, il n'existe pas de lien entre Alice et Mallory car l'attaquant ne s'est pas

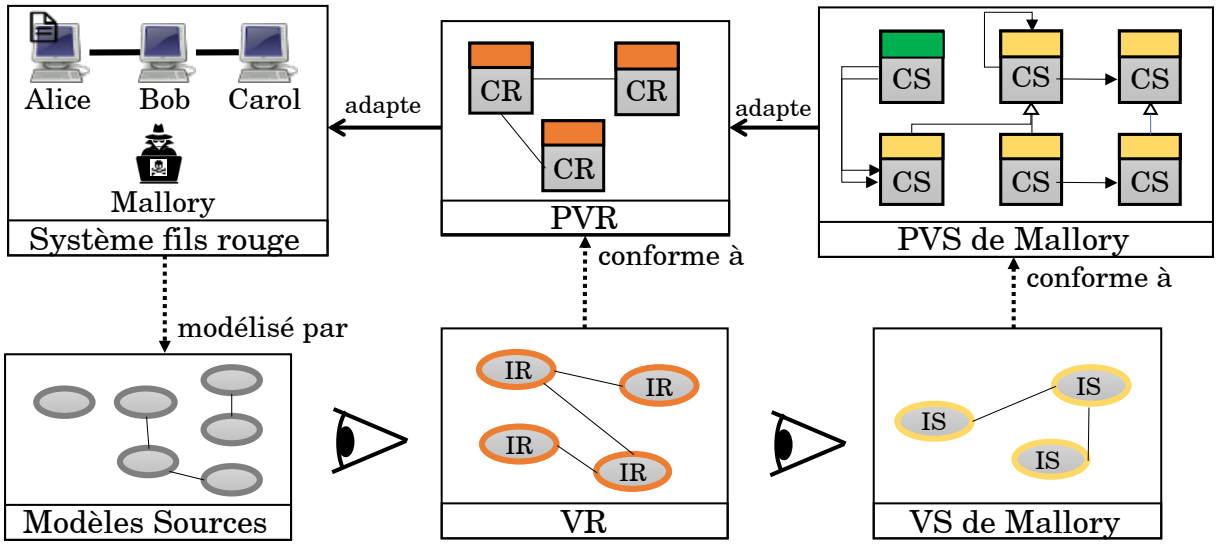


FIGURE 3.22 – Construction de la vue de Mallory sur le système fil rouge

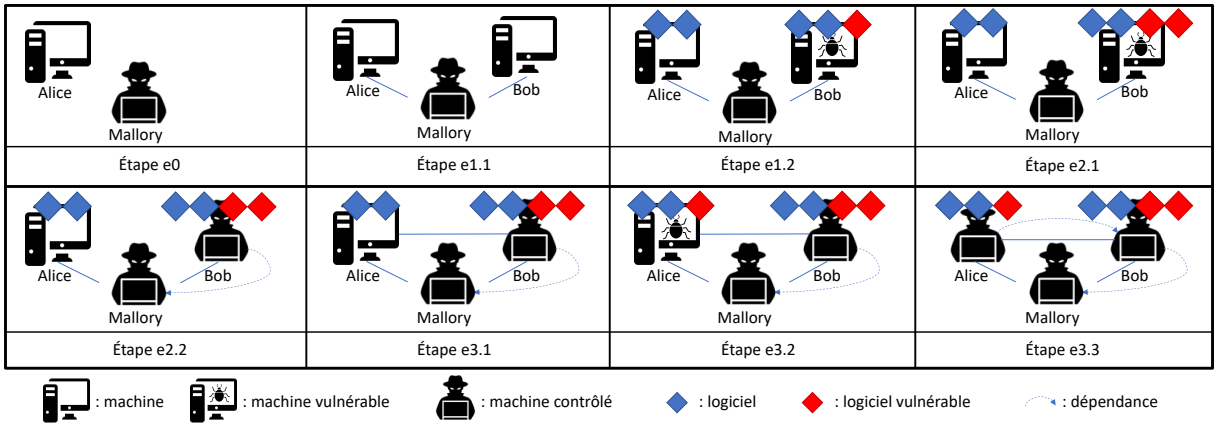


FIGURE 3.23 – Vue de Mallory sur le système fil rouge

encore connecté au système. Au cours de son attaque Mallory découvre de nouvelles machines et de nouveaux liens (étapes e1.1 et e3.1), découvre quels logiciels sont présents sur les diverses machines du système (e1.2, e2.1 et e3.2) et prend le contrôle de machines vulnérables du système (e2.2 et e3.3).

Les évolutions que connaît la vue de Mallory sur le système fil rouge sont les suivantes :

— Des éléments changent de type. Par exemple, la machine modélisant Bob devient une machine vulnérable à l'étape e1.2 puis une machine contrôlée à l'étape e2.2.

— De nouveaux éléments sont ajoutés à la vue. Bob n'est découvert que lors de l'étape e1.1, la machine le modélisant n'apparaît donc qu'à partir de cette étape. De même, les logiciels vulnérables d'Alice et de Bob ne sont découverts qu'au fur et à mesure de l'attaque.

— De nouvelles relations sont créées ou découvertes. Lors de l'étape e1.1, Mallory se connecte à Alice et à Bob, ce qui est modélisé par la création de nouvelles relations. De plus lors de l'étape e3.1, Mallory découvre l'existence d'un lien entre Alice et Bob, ce qui se traduit par la création d'une nouvelle relation.

Dans cet exemple, nous présentons une vue spécifique qui adapte les données provenant des VR. L'adaptation des données se fait à l'aide d'adaptateurs, par exemple à l'étape

e1.2, le logiciel vulnérable de la machine "Bob" est défini grâce à l'adaptation des données provenant de cinq IR. La figure 3.24 schématise cette adaptation.

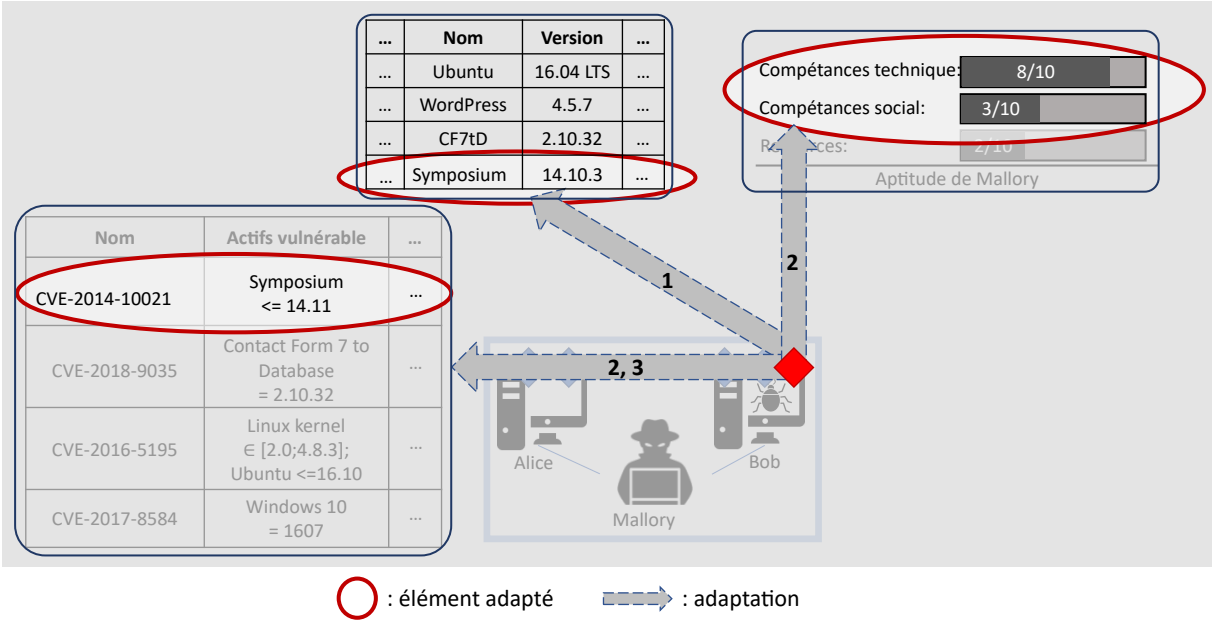


FIGURE 3.24 – Exemple d'adaptation

L'adaptation permettant de définir un logiciel vulnérable se fait en trois étapes :

1. Nous construisons un logiciel à partir du nom et du numéro de version d'un produit provenant de la configuration de Bob. Ici le logiciel a comme nom : "Symposium 14.11".
2. Le nom et la version du produit sélectionnés à l'étape précédente sont comparés aux actifs vulnérables présents sur le système. Dans cet exemple, il y a une correspondance, "Symposium 14.11" contient la vulnérabilité CVE-2014-10021. Nous comparons ensuite les prérequis de la vulnérabilité et les compétences techniques et sociales de Mallory. Dans cet exemple Mallory possède des aptitudes suffisantes afin d'exploiter la vulnérabilité CVE-2014-10021. Le logiciel "Symposium 14.11" est alors un logiciel vulnérable.
3. Nous définissons les prérequis du logiciel vulnérable à partir des attributs *vecteur* et *privilège* de la vulnérabilité CVE-2014-10021. De même, nous définissons les effets grâce aux impacts de cette vulnérabilité.

Finalement, l'utilisation d'adaptateur permet à la fois de recouper des informations mais aussi de construire une vue complète à partir d'informations partielles. Par ailleurs, au cours d'une attaque, la vue d'un attaquant évolue grâce à la découverte de nouveaux éléments ou la modification d'éléments existants. Ces deux spécificités sont discutées dans la section suivante (§3.4.3) afin d'identifier les caractéristiques de description de points de vue propres à l'attaquant.

3.4.3 Mise en évidence des caractéristiques propres à la description de points de vue d'attaquants

Dans la section 3.1, nous présentons une méthode permettant, à partir d'un ensemble de concepts de référence, de créer le point de vue d'un attaquant. Par la suite (§ 3.3, § 3.4.1, § 3.4.2) nous décrivons le processus de création de PVR, de PVS et des vues associées, nous

détaillons en particulier comment adapter les informations contenues dans des instances de référence afin de créer une VS. Dans chacune de ces sections, nous énumérons un ensemble de spécificités propres à la description du point de vue et de la vue d'un attaquant. Dans la section 2.3, nous présentons les travaux de Bruneliere et al. [27] proposant un ensemble de caractéristiques relatives à la description de points de vue. Ces dernières sont résumées dans un arbre de caractéristiques présenté en figure 3.25. La figure 3.25 pré-

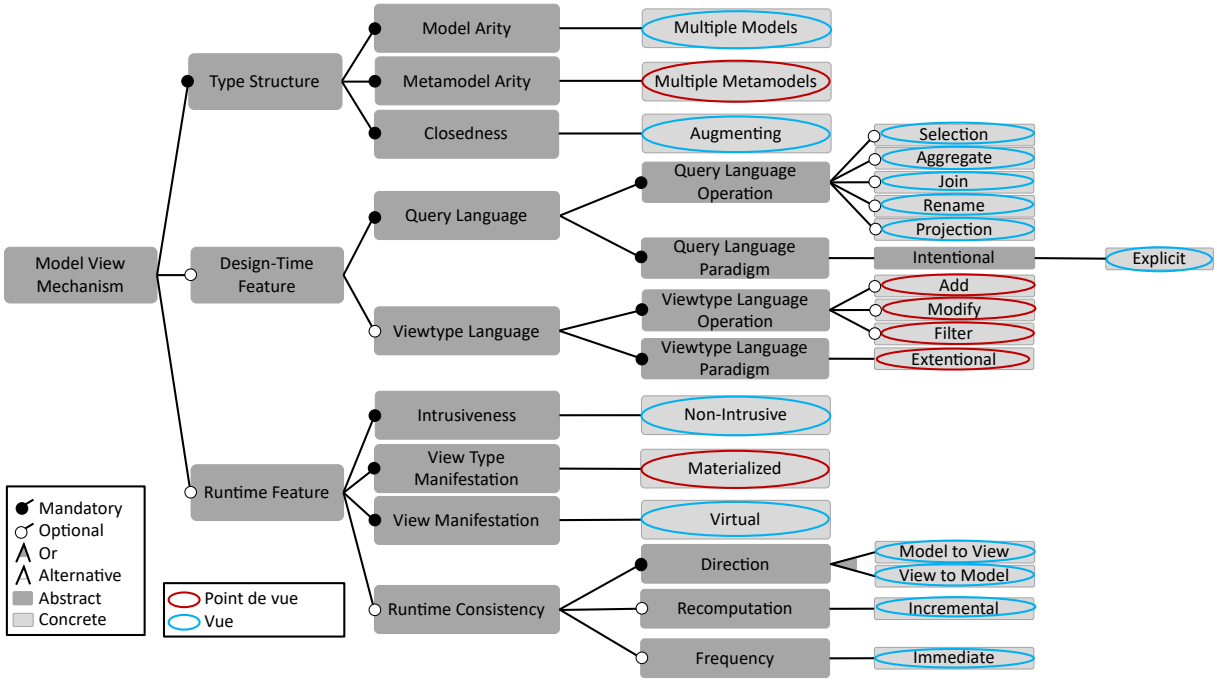


FIGURE 3.25 – Arbre de caractéristiques propres à la description de points de vue d'attaquants

sente uniquement les caractéristiques nécessaires à la description du point de vue d'un attaquant. Une version détaillée de cet arbre est présentée en section 2.3.2.

Dans cette sous-section, nous caractérisons le point de vue d'un attaquant. Pour ce faire, nous traduisons les spécificités propres à la modélisation de points de vue d'attaquants en caractéristiques de Bruneliere et al.. Dans la première partie, nous présentons les caractéristiques nécessaires à la modélisation du point de vue d'un attaquant. Dans la seconde, nous nous focalisons sur celles nécessaires à la modélisation de la vue d'un attaquant sur un système.

3.4.3.1 Traduction des spécificités propres à la modélisation du point de vue d'un attaquant en caractéristiques

Dans les sections 3.3 et 3.4.1, nous présentons deux types de points de vue utilisés afin de modéliser la perception d'un attaquant : les PVR et les PVS. Les PVR sont construits afin de couvrir l'ensemble des préoccupations énumérées en section 3.3.1. Les PVS permettent d'adapter les PVR afin de modéliser le point de vue d'un attaquant spécifique. Dans les sections précédentes, nous soulignons que la création de ces points de vue demande les spécificités suivantes :

— Un même PVR peut combiner différents concepts.

— Un même attaquant peut avoir différents PVS.

— Les concepts composant un point de vue peuvent être :

— Une combinaison de concepts provenant de plusieurs sources distinctes.

— Une combinaison / adaptation de CR.

— Un concept provenant d'un autre point de vue.

— Un nouveau concept définit spécifiquement.

Un concept constitutif d'un point de vue d'attaquant peut être construit via l'association de différents autres concepts provenant de sources hétérogènes. Un point de vue combine donc des concepts provenant de différents métamodèles. Cette spécificité est traduite par la caractéristique "Multiple Metamodel Arity". De plus, un concept constitutif d'un point de vue peut être défini spécifiquement, être une réutilisation d'un concept précédemment défini dans un autre point de vue ou être une combinaison de CR. Quelque soit la méthode utilisée pour le définir, ce concept sera explicitement modélisé dans le point de vue de l'attaquant. Les notions provenant de concepts préexistants sont alors dupliquées ou modifiées. Ce type de création de point de vue se caractérise par la caractéristique "Viewtype Manifestation Materialized".

Le choix de ces deux caractéristiques implique que le langage permettant de modéliser un point de vue doit permettre à la fois la création de nouveaux concepts et la réutilisation/adaptation de concepts existants. Ces différentes méthodes de création de concepts correspondent à trois opérations, traduites en caractéristiques cela correspond à :"Viewtype language Operation Add", "Viewtype language Operation Modify" et "Viewtype language Operation Filter".

Un point de vue d'un attaquant décrit explicitement le type d'élément pouvant appartenir à une vue. Ce qui correspond à la caractéristique "Viewtype language Paradigm Extentional".

Finalement, les spécificités de la création de points de vue d'attaquant correspondent à 6 caractéristiques de Bruneliere et al. identifiées par un cercle rouge sur la figure 3.25, à savoir :

— Multiple Metamodel Arity

— Viewtype Manifestation Materialized

— Viewtype language Operation Add

— Viewtype language Operation Modify

— Viewtype language Operation Filter

— Viewtype language Paradigm Extentional

3.4.3.2 Mise en évidence des caractéristiques spécifiques à la modélisation d'une vue d'un attaquant

Dans la sous-section 3.4.2, nous soulignons que la création d'une vue d'un attaquant sur un système demande les spécificités suivantes :

— L'application d'une vue ne demande aucune modification de l'élément observé.

— L'adaptation entre la vue d'un attaquant et les sources d'information est bidirectionnelle.

— Les informations contenues dans une VS sont une adaptation des informations provenant d'IR ou d'autres VS auxquelles peuvent s'ajouter des données définies spécifiquement.

— Les informations provenant d'IR ou d'autres vues sont mises en forme mais ne sont pas dupliquées.

Dans notre approche, l'application d'une vue ne requiert aucune modification de l'élément observé, cette spécificité correspond à la caractéristique "Intrisiveness Non-Intrusive".

La mise en place d'une adaptation bidirectionnelle permettant de modifier une source à travers une vue et de mettre à jour une vue lors de la modification d'une source, correspond à deux caractéristiques : "Runtime Consistency Direction Model to View" et "Runtime Consistency Direction View to Model".

Les informations contenues dans une vue sont en majeure partie des références vers des sources hétérogènes. Traduite en terme de caractéristiques, cette spécificité devient "Multiple Model Arity". Les informations provenant des VR peuvent être complétées afin de peupler une VS. Le langage permettant de modéliser une vue doit alors permettre de référencer des informations contenues dans des VR mais aussi permettre de modifier ces informations afin de les adapter à la VS. Il doit également permettre de définir spécifiquement de nouvelles données. Les spécificités du langage de modélisation de vue se traduisent par les caractéristiques "Closedness Augmenting" et "Query Language Opération Selection, Projection, Join, Aggregate and Rename". La création d'une vue se fait via l'utilisation des opérations "Selection", "Projection", "Join", "Aggregate" et "Rename". Le paradigme du langage de création d'une vue est alors, par définition, intentionnel. L'utilisation d'adaptateur, conceptualisée au niveau métamodèle, permet d'affirmer que ce paradigme est également explicite, ce qui correspond à la caractéristique "Query language Paradigm Intensional Explicit".

Mises à part les données définies spécifiquement, une vue ne contient pas de données propres mais uniquement des références. Les informations contenues dans les VR sont alors une référence aux données des modèles sources. Cette spécificité implique que les données contenues dans les VR ne sont pas dupliquées. L'absence de recopie de données correspond à la caractéristique "View Manifestation Virtual", de plus, comme nous le verrons dans le chapitre 4, cette spécificité implique les caractéristiques "Runetime Consistency Recomputation Incremental" et "Runetime Consistency Frequency Immediate".

Finalement les spécificités propres à la création d'une vue d'un attaquant sur un système correspondent à 14 caractéristiques identifiées par un cercle bleu sur la figure 3.25, à savoir :

— Intrisiveness Non-Intrusive
— Runetime Consistency Direction Model to View
— Runetime Consistency Direction View to Model
— Multiple Model Arity
— Closedness Augmenting
— Query Language Opération Selection
— Query Language Opération Projection
— Query Language Opération Join
— Query Language Opération Aggregate
— Query Language Opération Rename
— Query language Paradigm Intensional Explicit
— View Manifestation Virtual
— Runetime Consistency Recomputation Incremental
— Runetime Consistency Frequency Immediate

3.5 Bilan de l'approche

Dans les deux parties précédentes, nous avons énuméré 20 caractéristiques nécessaires à la description de points de vue d'un attaquant pour le domaine de la modélisation et l'analyse de menaces. Comme présenté dans la section 2.3, Bruneliere et al. [27] comparent 16 outils permettant de modéliser des points de vue en présentant les caractéristiques couvertes par chacun. Aucune des solutions présentées dans cette étude ne couvre pleinement l'ensemble des 20 caractéristiques nécessaires à notre approche. Cependant, certains des outils présentés en couvrent un grand nombres tel que *EMF Views* (18), *Kitalpha* (18), *VIATRA viewers* (18) et *OpenFlexo* (19) mais, compte tenu des informations disponibles, aucun ne prend en charge la totalité de nos caractéristiques énumérées.

Les caractéristiques retenues ont été sélectionnées pour satisfaire la création de PVR. Les PVR sont constitués de concept que nous considérons comme des références par rapport au domaine d'étude. Ils décrivent des vulnérabilités, des systèmes, les aptitudes et les connaissances d'un attaquant ainsi que le comportement d'un système et d'un attaquant. Ces concepts sont suffisamment génériques et représentatifs pour être réutilisés dans différentes modélisations. Cependant, cette généricité nuit à la spécialisation nécessaire pour les différentes parties prenantes (les différents attaquants avec leurs différentes aptitudes).

Ce compromis entre généricité et spécialisation existe toujours, dès lors que nous cherchons à créer des vocabulaires de référence au sein d'un domaine. C'est pourquoi, nous avons choisi de proposer une méthodologie intégrant nativement l'adaptation aux parties prenantes avec la création de PVS, permettant de spécialiser les PVR aux différents profils d'attaquant. Ainsi, dans notre approche, nous utilisons la relation *adapte* comme le standard de la modélisation des points de vue mais aussi pour adapter des PVR avec des PVS. Dans ce dernier cas, les PVR jouent le rôle des metamodèles sources. Ces différentes adaptations successives préservent les modèles sources. De plus, la création de concepts de point de vue (PVR et PVS) possédant leur propre comportement permet une adaptation dynamique.

Pour résumer notre approche, les métamodèles sources contiennent des informations décrivant un système, les PVR mettent en forme ces informations pour le domaine de la modélisation et l'analyse de la menace et les PVS composent ces adaptations afin de modéliser la perception qu'un attaquant spécifique à d'un système.

La contribution que nous allons exposer dans le chapitre suivant (§4) cherche donc à garantir ces propriétés de préservation des modèles sources et d'adaptation dynamique tout en s'appuyant sur les caractéristiques identifiées dans la formalisation de Bruneliere et al. [27].

Chapitre 4

Role4All : Une modélisation de points de vue par les rôles

Sommaire

Dans le chapitre 2, nous présentons, entre autres, la notion de rôle. Pour rappel, dans ce manuscrit, nous nous fondons sur la définition des rôles donnée par Steimann [154] et étendue par Kühn [100].

Dans le chapitre 3, nous présentons notre approche de modélisation de la perception d'un attaquant sur un système en nous basant sur les notions de métamodèle source, de point de vue de référence (PVR) et de point de vue spécifique (PVS). Nous identifions également 20 caractéristiques des outils de modélisation de point de vue nécessaires à la mise en œuvre de notre approche.

Dans ce chapitre, nous présentons notre proposition de modélisation par les rôles, qui couvre l'ensemble des caractéristiques identifiées précédemment, tout en offrant une grande flexibilité de modélisation et la prise en compte de comportements. Dans une première section, nous introduisons Role4All, notre solution de modélisation par les rôles. Dans les deux sections suivantes, nous détaillons l'utilisation de Role4All pour, respectivement, modéliser et analyser le point de vue d'un attaquant.

4.1 Role4All une proposition de modélisation de points de vue par les rôles

Dans cette section, nous présentons notre approche de modélisation fondée sur les rôles appelée Role4All. Dans un premier temps nous détaillons le métamodèle de Role4All, puis nous formalisons mathématiquement notre approche et pour finir nous présentons une implémentation de Role4All.

4.1.1 Présentation de Role4All

Dans cette section, nous présentons Role4All, un cadre de fédération de modèles utilisant le concept de rôle pour définir des points de vue et pour relier ces derniers à différents métamodèles sources. Role4All est basé sur les travaux antérieurs de Schneider et al. [142].

4.1.1.1 Description du métamodèle

Pour rappel (§2.4), un rôle est une interprétation d'un type originel dans un contexte spécifique. Par conséquent, un type, développé dans son propre contexte d'intention, peut être utilisé dans un autre contexte via un rôle.

Nous utilisons la notion de rôle à trois niveaux d'abstraction à savoir, la classe rôle (*roleClass*), le *rôle* et l'instance de rôle (*roleInstance*). Le terme *roleClass* correspond au concept nommé "Role" dans le métamodèle Role4All, représenté en figure 4.1. Un *rôle* est une spécification de ce concept et un *roleInstance* est une instance d'un *rôle*. Une convention de dénomination similaire est utilisée dans tout le manuscrit pour les autres concepts utilisés.

Le métamodèle de Role4All est fondé sur six classes principales nommées : "Player", "Type", "Adapter", "PlayRelation", "Role" et "Context" :

— Un *player* est un élément de modélisation abstrait qui est interprété au travers d'un *rôle* dans un nouveau contexte. Un *player* peut être spécialisé comme un *type* ou comme un *rôle*.

— Un *type* est un élément d'un métamodèle source.

— Un *adapter* est responsable de la transformation des propriétés d'un *player* en propriétés d'un *rôle* et inversement. Pour reprendre le contexte traditionnel d'ingénierie dirigée par les modèles, nous plaçons le code pertinent pour la transformation dans la définition des *adapters*.

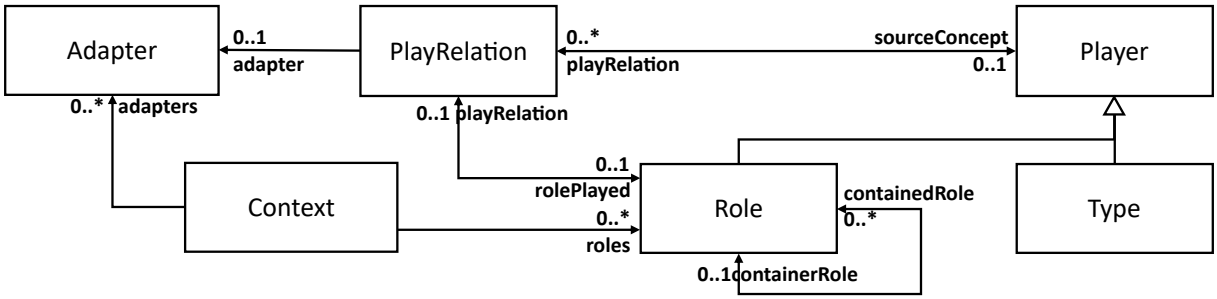


FIGURE 4.1 – Métamodèle de Role4All, concepts principaux

— Une *playRelation* relie un *player*, un *rôle* et un *adapter*. L'existence d'un tel élément permet de changer dynamiquement la sémantique d'une relation (interprétation) entre un *rôle* et un *player*. En effet, l'*adapter* d'une *playRelation* peut être remplacé en cours d'exécution sans avoir d'impact sur le *rôle* ou le *player* associé. Par construction du métamodèle de Role4All, une même *playRelationInstance* relie un *roleInstance* à un unique *playerInstance* et un unique *adapterIntance*. De plus, un *roleInstance* ne peut participer qu'à une seule *playRelation*, cependant, un *playerInstance* peut jouer différents *rôles* au travers de diverses *playRelationInstance*.

— Un *rôle* est une interprétation d'un *player* conformément à un *context*. La classe "Role" est une spécialisation de la classe "Player", ce qui signifie qu'un *rôle* peut être considéré comme un *player*. De plus, nous définissons une relation de contenance entre des *roleInstances* permettant à un *roleInstance* d'être contenu par un autre *roleInstance* et de contenir plusieurs *roleInstances*. Un conteneur peut lui-même agir en tant que contenu lorsqu'une hiérarchie complexe est requise. Par la suite, les *roleInstances* contenus seront appelés des *containedRoles* et les conteneurs seront appelés des *containerRoles*. Un *containerRole* est donc un *roleInstance* qui contient des *containedRoles*.

— Un *context* regroupe un ensemble de *rôles* et d'*adapters*, il définit également un ensemble de règles d'associations permettant de conditionner la relation entre un *player* et un *rôle*. Nous détaillons les notions de *context* et de règle d'association dans la section 4.1.3 présentant notre implémentation de Role4All.

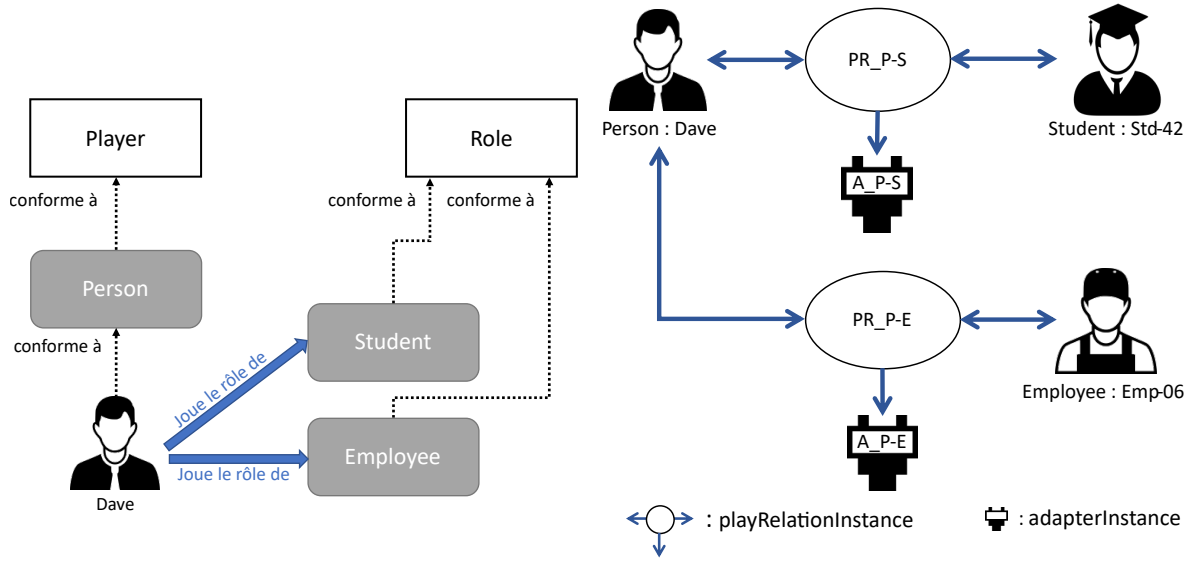
Nous utilisons le vocable "joue le rôle de" entre un *playerInstance* et un *rôle* pour dire qu'un *playerInstance* joue le rôle de quelque-chose, ce qui se traduit par :

— La création d'un *roleInstance* conforme au *rôle* joué.

— La création d'un *adapterInstance* responsable de la transformation des propriétés du *player* instancié en propriétés du *rôle* joué.

— La création d'une *playRelationInstance* reliant l'*adapterInstance*, le *roleInstance* et le *playerInstance*.

La figure 4.2 présente un exemple de relation entre un *playerInstance*, nommée Dave, et différents *rôles*. Sur la figure 4.2(a) on peut voir que Dave joue le rôle de "Student" et également celui de "Employee". Comme illustré en figure 4.2(b), Dave peut être vu comme un étudiant, via le *roleInstance* "Std-42", ou comme un employer, via le *roleInstance* "Emp-06". Les liens entre Dave et les différents *roleInstances* sont modélisés par des *playRelationInstances* et des *adapterIntances*. Dans cet exemple, nous utilisons deux *adapteurs* traduisant les propriétés du *player* "Person" en propriétés des *rôles* "Student" et "Employee". Au niveau des instances, la relation entre Dave et "Std-42" est rendue possible grâce à la *playRelationInstances* "PR_P-S" et l'*adapterInstance* "A_P-S" transformant les propriétés de "Person" en propriétés de "Student" et inversement.



(a) Relation niveau métamodèle (b) Relation niveau modèle

FIGURE 4.2 – Exemple de relation "joue le rôle de"

Finalement, le métamodèle de Role4All nous permet d'interpréter des *players* au travers de *rôles*. Cette interprétation est modélisée par une *playRelation* et un *adapter*. Un *rôle* permet de représenter un *player* dans un nouveau contexte. Nous verrons par la suite que les notions de *rôle* et de *context* nous permettent de construire le point de vue d'un attaquant spécifique sur un système.

4.1.1.2 Modélisation de point de vue avec Role4All

Nous présentons dans une première partie comment construire un point de vue dans Role4All. Dans une seconde partie nous comparons notre solution à l'état de l'art.

Relation entre point de vue et rôle dans Role4All

Dans le chapitre 3, nous présentons la terminologie propre à la modélisation par les vues. Une vue, conforme à un point de vue, est une représentation d'un modèle source, lui-même conforme à un métamodèle source.

Un métamodèle source est modélisé dans Role4All par un ensemble de *players* et un point de vue par un modèle de rôles. Pour reprendre l'exemple de la section précédente (figure 4.2), le *player* "Personne" est un métamodèle source, le *playerInstance* Dave est un modèle source, le *rôle* "Student" est un point de vue et le *roleInstance* "Std-42" une vue représentant Dave, conformément à ce point de vue.

La relation entre un *roleInstance* et un *playerInstance* est réalisée à l'aide de *playRelations* et d'*adapters*. Appliquer une vue sur un modèle source revient alors à :

— Construire un *adapter* traduisant les propriétés d'un *player* en propriétés d'un *rôle*, et inversement.

— Instancier cet *adapter* et créer une *playRelationInstance* reliant le *roleInstance*, le *playerInstance* et l'*adapterInstance*.

Construire une vue sur un modèle source revient donc à faire jouer un *rôle* à un *playerInstance*. De plus, dans le chapitre 3 nous soulignons qu'il est possible de changer l'interprétation qu'un observateur se fait d'un objet. Dans le contexte de Role4All, cela revient à modifier l'*adapter* participant à une *playRelation*.

Dans le métamodèle de Role4All, un *roleInstance* ne peut être lié qu'à un seul *playerInstance*. Ce choix de modélisation permet de simplifier l'adaptation bidirectionnelle entre un *rôle* et un *player*. Il permet également de limiter le nombre d'adaptations possibles. En effet, pour x *roleInstances* et y *playerInstances* il existe, dans le cas où un *roleInstance* peut être lié à différents *playerInstancs* $x * (2^y - 1)$ adaptations différentes et seulement $x * y$ dans le cas où un *roleInstance* n'est lié qu'à un seul *playerInstance* [1].

Cependant, cette restriction occasionne des contraintes de modélisation, un *roleInstance* ne pouvant être lié qu'à un seul *playerInstance*, il n'est pas possible de créer un *roleInstance* regroupant des informations provenant de différentes sources. Pour pallier ce manque, nous utilisons la relation entre *roleInstances* avec des *containerRoles* qui contiennent des *containedRoles*. Comme l'illustre la figure 4.3, si chaque *containedRole* est en relation avec un *playerInstance*, alors le *containerRole* référence (indirectement) un ensemble de *playerInstances*. Cette relation de contenance permet de définir un comportement global

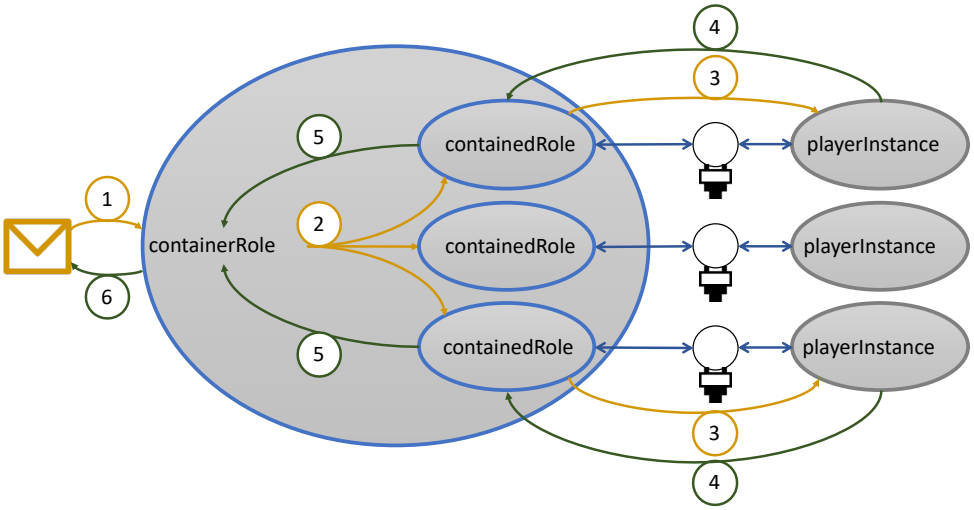


FIGURE 4.3 – Transmissions de messages via la relation de contenance

fondé sur les comportements locaux des éléments contenus et permet également la diffusion de messages à un ensemble de *playerInstances*. Pour ce faire, le *containerRole* prendre en charge la diffusion de notifications provenant des comportements locaux et délègue les appels de méthodes aux *containedRoles*.

La figure 4.3 schématise l'envoi d'un message depuis un *containerRole* vers des *playerInstances* grâce à la relation de contenance. L'envoi d'un message y est découpé en six étapes :

1. Envoi du message au *containerRole*. Le message peut provenir de l'environnement, d'un *playerInstance*, d'un *roleInstance* ou même d'un autre *containerRole*.
2. Diffusion du message à tous les *containedRoles*. Le message est envoyé à chaque *containedRoles*, il est alors traité et le comportement associé est déclenché. Si un message ne peut pas être traité, il est simplement ignoré.
3. Interaction avec les *playerInstances*. Les *containedRoles* pouvant traiter le message interagissent avec leurs *playerInstance* respective.
4. Réponse des *playerInstances*. Une fois le message traité, chaque *playerInstance* envoie une réponse à son *containedRole*.
5. Transmission de la réponse. Les *containedRoles* transmettent leur réponse au *containerRole*.

1. La justification de cette affirmation est faite en annexe C.

6. Traitement de l'information. Le *containerRole* compare les différentes réponses reçues. Si elles sont cohérentes, il transmet l'information à l'émetteur du message et s'il détecte une incohérence, il génère une erreur.

Comparaison avec l'état de l'art

Notre utilisation de la contenance hiérarchique est comparable aux travaux de Seifert et al. [143] présentés dans la section 2.4.2. Les auteurs définissent la notion de *RoleGrounding* qui est utilisée pour identifier les rôles qui matérialisent physiquement les données. Ils soulignent également que les rôles ne font pas de distinction entre les données physiquement représentées et les données obtenues à partir d'autres objets. L'utilisation de *RoleGrounding* permet alors de reporter la décision de la matérialisation des données jusqu'au moment de l'intégration des outils.

Notre approche est comparable car elle permet à un *containerRole* d'accéder à des informations sans savoir où elles sont physiquement présentes. Un *containerRole* référence des informations provenant de *containedRoles*, qui eux-mêmes adaptent des données contenues dans différents *playerInstances*. De plus, notre approche permet de gérer les informations dupliquées. En effet, si différents *playerInstances* contiennent une même information, cette dernière est référencée par un unique *containerRole*. Cela permet de simplifier la synchronisation des données, l'ensemble des modifications étant envoyées à tous les *playerInstances* impliqués.

Dans la section 2.4.2 nous présentons un ensemble d'exigences pour l'intégration d'outils tirés des travaux de [143]. Notre approche permet de couvrir l'ensemble de ces exigences, à savoir :

- ✓ Abstraction appropriée : les *players* ont une abstraction de données qui leur sont propres, les rôles ne font qu'adapter ces données et déclencher des comportements.
- ✓ Indépendance : dans notre approche les *players* sont autonomes et ont chacun leur propre comportement.
- ✓ Données partagées : différents *playerInstances* peuvent, au travers d'un *containerRole*, partager des données.
- ✓ Interaction : le comportement d'un *playerInstance* peut, au travers d'un *containerRole*, déclencher un comportement sur un autre *playerInstance*.

Finalement, Role4All permet de modéliser des points de vue à l'aide de modèles de *rôle*, il permet également de construire des modèles complexes grâce à l'utilisation de *containedRole* et de *containedRoles*. Notre approche couvre l'ensemble des exigences de Seifert et al. [143] et, contrairement à ce dernier, n'impose pas l'utilisation de rôles pour la modélisation des métamodèles sources. Cependant notre approche demande une gestion de la diffusion de messages et possiblement de la gestion de synchronisation. Les solutions techniques proposées pour ces aspects sont présentées dans la section 4.1.3.

4.1.2 Formalisation de Role4All

Dans cette section, nous formalisons notre approche de modélisation par les rôles. Pour ce faire, nous identifions les caractéristiques des rôles couvertes par Role4All, puis, nous en proposons une formalisation mathématique.

4.1.2.1 Caractéristiques couvertes par Role4All

Dans l'état de l'art 2.4, nous détaillons une liste de 27 caractéristiques proposées par Steimann [152] et étendue par Kühn [100] permettant de décrire une approche de modélisation

par les rôles. Nous avons également présenté l'outil *FRaMED* [104] qui génère un métamodèle à partir d'une sélection de ces caractéristiques. Dans cette section nous exposons les caractéristiques couvertes par Role4All et utilisons l'outil *FRaMED* pour générer un métamodèle qui sera comparé à celui de Role4All. Cette approche nous permet de nous comparer à l'état de l'art et aussi de justifier nos choix de modélisation dans le métamodèle de Role4All.

Caractérisation de Role4All

Dans la section 2.4.1, nous regroupons les caractéristiques de rôles en quatre catégories : structurelles, restrictives, identitaires et relationnelles, cette dernière étant elle-même divisée en trois sous-catégories, à savoir, rôle –> *player*, *player* –> rôle et compartiment <–> contenu. Les caractéristiques couvertes par Role4All sont présentées sur le tableau 4.1.

Caractéristiques		**Couverture**	
Catégorie	**n**	**Role4All**	**Taux**
Structurelles	1	●	100%
	13	●	
	20	●	
	25	●	
Restrictives	2	○	21%
	6	◐	
	16	◐	
	17	○	
	18	○	
	19	◐	
	27	○	
Identitaires	14	○	67%
	15	●	
	26	●	
Relationnelles Rôle –> *player*	8	●	100%
	10	●	
	11	●	
	12	●	
Relationnelles *Player* –> rôle	3	●	100%
	4	●	
	5	●	
	7	●	
	9	●	
Relationnelles Compartiment –> contenu	21	●	25%
	22	○	
	23	○	
	24	○	

○ : Non couverte ◐ : Partiellement couverte ● : Couverte

Tableau 4.1 – Ensemble des caractéristiques couvertes par Role4All

Pour chaque catégorie nous calculons un taux de couverture en fonction du nombre de caractéristiques couvertes. Il est à noter que la catégorie identitaire ne peut être couverte qu'à 67% au maximum car les caractéristiques 14 et 15 sont mutuellement exclusives.

Role4All couvre entièrement les caractéristiques structurelles, identitaires, relationnelles - *rôle* –> *player* et relationnelles - *player* –> *rôle*. Cependant il ne couvre que partiellement les caractéristiques restrictives et relationnelles - compartiment –> contenu.

Les caractéristiques structurelles assurent que dans Role4All tous les concepts (*player*, *rôle* et *context*) possèdent un comportement, des propriétés et une notion d'héritage.

Les caractéristiques identitaires permettent la distinction entre les concepts de *rôle*, de *context* et de *player*. Cette distinction permet, par exemple, d'assurer l'indépendance entre les propriétés d'un *rôle* et celles d'un *player*.

La couverture totale des caractéristiques relationnelles - *rôle* –> *player* et relationnelles - *player* –> *rôle* assure une grande flexibilité dans la construction des modèles de *rôle*.

Nous justifions nos choix de ces caractéristiques au regard de la construction du point de vue d'un attaquant dans la section 4.2.

Génération du métamodèle complet de Role4All

Nous présentons en section 2.4.2 *FRaMED* [104], un outil permettant de générer un métamodèle à partir d'un ensemble de caractéristiques. Nous générons le métamodèle de Role4All à partir des caractéristiques couvertes par notre approche. Ce dernier, qui comprend 24 concepts, est présenté en annexe B, la figure 4.4 représente la partie du métamodèle générée décrivant les relations entre les concepts de "Player" et de "RoleType".

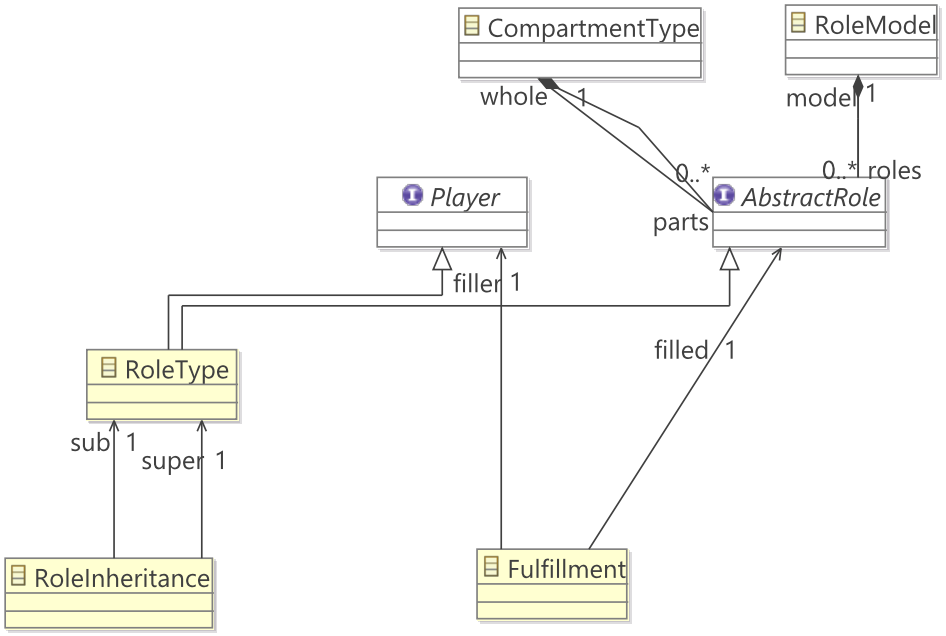


FIGURE 4.4 – Section relative aux relations entre *rôle* et *player* du métamodèle de Role4All générée par *FRaMED* [104]

Bien que représentant la même approche, notre métamodèle et la version générée ne sont pas identiques. Les différences observables entre les deux métamodèles sont :

— Sur la version générée la notion de *rôle* est explicitée en deux concepts : "RoleType" et "AbstractRole".

— Le métamodèle généré explicite la notion de modèles de rôle et celle d'héritage.

— La version complète du métamodèle généré, présenté en annexe B, modélise également les notions de nom, de relation, de type, etc.

— La notion d'*adapter* est absente de la version générée.

Les trois premières différences identifiées mettent principalement en évidence des notions génériques dans la définition d'un langage avec l'héritage ou les relations. Pour simplifier notre implantation et nous focaliser sur les concepts principaux de notre approche, nous avons opté pour le choix d'un DSL interne à un langage de prototypage, en l'occurrence Pharo. L'ensemble des concepts hérite alors de la notion d'objet. Cela nous permet de nous soustraire de la modélisation concrète des concepts qui ne sont pas spécifiques à notre approche.

De plus, nous identifions comme quatrième différence majeure : le concept d'*adapter* qui est propre à notre approche. Nous avions déjà présenté dans la section 3.1.2 notre utilisation de la relation d'adaptation comme spécifique. En effet dans notre métamodèle, le concept d'*adapter* est utilisé pour décrire la transformation entre un *rôle* et un *player* liés par une *playRelation*.

Finalement, nous pouvons conclure que le métamodèle de Role4All est cohérent avec le métamodèle généré depuis la sélection des caractéristiques des rôles. Les différences observables étant dues au choix d'implémenter Role4All comme un DSL interne à Pharo. Cette implémentation de Role4All est décrite en section 4.1.3.

4.1.2.2 Formalisation mathématique de Role4All

Nous soulignons dans l'état de l'art sur les rôles (§2.4) que, pour éviter les écueils des travaux antérieurs, il est nécessaire de formaliser la notion de rôle utilisée dans notre approche. Dans cet objectif, nous formalisons, dans la section 2.4.1, les notions de modèle d'objet de rôle et de compartiment avec héritage ($CROM_I$) et d'instance d'objet de rôle et de compartiment avec héritage ($CROI_I$). Dans cette section, nous formalisons mathématiquement la notion de rôle utilisée dans Role4All et présentons un ensemble d'axiomes et de propriétés propres à notre approche dérivée des axiomes de Kühn [100]. De plus, nous décrivons les similitudes observables entre notre formalisation des rôles et la formalisation des points de vue présentée dans la section 2.3.2.

Formalisation des modèles de rôle

Certaines notations introduites dans la section 2.4.1 peuvent être reliées aux concepts de Role4All :

— NT correspond au *type*.

— RT correspond au *rôle*.

— CT correspond au *context*.

— rel est une fonction prenant en entrée une relation et un *context* et retournant les deux *rôles* liés par cette relation.

Nous introduisons également la notation $PT := NT \cup RT$ comme l'ensemble des *players*. Nous définissons deux axiomes ainsi que quatre propositions dérivées des travaux de [100] [1]. Un modèle de rôle correctement formé, pour Role4All, doit respecter les axiomes A1 et A2, ce qui signifie que deux *rôles* en relation appartiennent au même *context* (A1) et qu'un *player* hérite de tous les attributs et méthodes de son super-type (A2). Dans notre approche, un *rôle* étant aussi un *player*, l'axiome A2 assure donc également qu'un *rôle* hérite de tous les attributs et méthodes de son super-type.

Les propositions P1, P2, P3 et P4 ne sont pas obligatoirement respectées par tous les modèles de rôle générés par Role4All. Cependant ces dernières, si respectées, permettent de simplifier significativement les analyses formelles [100]. En l'occurrence, elles assurent que chaque *rôle* est lié à au moins un *player* (P1), qu'un *context* contient au moins un *rôle*

1. Pour une fonction $f : A \to B$, $domain(f) = A$ et $codomain(f) = B$.

$$\forall(rst, ct) \in domaine(rel) : rel(rst, ct) = (rt_1, rt_2) \wedge (\_, ct, rt_1 \wedge (\_, ct, rt_2) \in fills \quad \textbf{(A1)}$$

$$\forall(pt_1, pt_2) \in \preceq_{PT}: fields(pt_2) \subseteq fields(pt_1) \wedge methods(pt_2) \subseteq methods(pt_1) \quad \textbf{(A2)}$$

$$\forall rt \in RT\ \exists ct \in CT\ \exists pt \in PT : (pt, ct, rt) \in fills \quad \textbf{(P1)}$$

$$\forall ct \in CT\ \exists (pt, ct, rt) \in fills \quad \textbf{(P2)}$$

$$\forall rst \in RST\ \exists ct \in CT : (rst, ct) \in domaine(rel) \quad \textbf{(P3)}$$

$$\forall (rt_1, rt_2) \in codomaine(rel) : rt_1 \neq rt_2 \quad \textbf{(P4)}$$

lié à au moins un *player* (P2), que toute relation est définie dans l'enceinte d'un *context* (P3) et qu'un *rôle* ne se réfère pas lui-même (P4). Si ces propositions ne sont pas respectées, le modèle de rôle reste correctement formé mais l'explosion combinatoire propre aux analyses formelles sera plus importante [100].

Formalisation des *roleInstances*

Certaines notations introduites dans la section 2.4.1 peuvent être reliées directement à notre approche :

— N correspond à l'ensemble des *typeInstances*.

— R correspond à l'ensemble des *roleInstances*.

— C correspond à l'ensemble des *contextInstances*.

Nous introduisons également les notations suivantes :

— $P := N \cup R$: l'ensemble des *playerInstances*.

— $contain \subseteq R \times R$: une relation qui définit un lien de contenance entre un contenant et un contenu.

— $container : R \to R$: une fonction retournant le contenant d'un *roleInstance*

— $content : R \to MM(R)$: une fonction retournant le contenu d'un *roleInstance*.

Nous définissons six axiomes ainsi qu'une proposition dérivés des travaux de [100] :

$$\forall r_1, r_2 \in R,\ container(r_1) = r_2 \Leftrightarrow r_1 \in content(r_2) \quad \textbf{(A3)}$$

$$\forall (p, c, r) \in plays\ \exists (pt, type(c), type(r)) \in fills : type(p) \preceq_{PT} pt \quad \textbf{(A4)}$$

$$\forall r \in R\ \exists! p \in P\ \exists! c \in C : (p, c, r) \in plays \quad \textbf{(A5)}$$

$$\forall rst \in RST\ \forall c \in C\ \forall (r_1, r_2) \in links(rst, c) : (rst, type(c)) \in domaine(rel) \wedge (\_, c, r_1), (\_, c, r_2) \in plays \wedge rel(rst, type(c)) = (type(r_1), type(r_2)) \quad \textbf{(A6)}$$

$$\forall p \in P\ \forall (f, v) \in attr(p)\ \exists (f, pt) \in fields(type(p)) : type(v) \preceq_{PT} pt \quad \textbf{(A7)}$$

$$\forall (p, c, r) \in plays\ \forall (f, v) \in attr(r)\ \exists (f, pt) \in fields(type(r)) : type(v) \preceq_{PT} pt \quad \textbf{(A8)}$$

$$(p, c, r), (p, c, r') \in plays : r \neq r' \Rightarrow type(r) \neq type(r') \quad \textbf{(P5)}$$

Un modèle d'instance de rôle correctement formé, pour Role4All, doit respecter les axiomes A3 à A8. Ces derniers assurent que la relation de contenance est bijective (A3), qu'une *playRelationInstance* entre un *playerInstance* et un *roleInstance* est possible seulement si le *rôle* en question est en relation avec le *player* ou un de ses sous-types (A4), qu'un *roleInstance* n'est lié qu'à un seul *playerInstance* (A5), que le type des *roleIntances* d'un modèle d'instance est bien conforme aux relations définies au niveau conceptuel (A6) et qu'un attribut d'un *playerInstance* ou d'un *roleIntance* est conforme au type défini au niveau conceptuel ou à un sous-type de ce dernier (A7 et A8).

La proposition P5 est facultative et assure qu'un *playerInstance* ne joue pas deux fois le même *rôle*. Ce dernier a pour objectif de limiter le nombre de relations lors d'une analyse formelle des modèles d'instances de rôle à une analyse combinatoire.

La formalisation présentée ici nous permet, en plus de proposer une description mathématique de notre approche, de justifier formellement l'utilisation des rôles pour la description de point de vue.

Équivalence avec la formalisation mathématique de point de vue

Dans notre approche, nous utilisons les rôles pour modéliser des points de vue, il est possible d'établir au niveau formel certains liens entre la formalisation des rôles dans Role4All et celle des points de vue présentée dans la section 2.3.2.

$PT - - > R_c$: L'ensemble des *players* est comparable à l'ensemble des concepts cohérents et celui des *rôles* à l'ensemble des points de vue, cependant R_c inclut également les types de relation, ce que ne fait pas PT.

$P - - > D_{vt}$: L'ensemble des *playerInstances* est comparable à l'ensemble des entités utilisées dans une vue, cependant D_{vt} inclut également les relations, ce que ne fait pas P.

$fills - - > f_{vt}$: La relation $fills$ regroupe les *players* avec les *rôles* qu'ils peuvent jouer ce qui correspond à la fonction de filtrage f_{vt} qui détermine les élément d'une architecture pouvant appartenir à un point de vue.

$RT + rel - - > S_{vt} + i_{iv}$: RT représente l'ensemble des *rôles* d'un modèle de rôle et la fonction rel permet d'identifier les relations entre ces *rôles*. Ces deux ensembles correspondent, pour la formalisation de points de vue, à l'ensemble des syntaxes abstraites et l'ensemble des sémantiques utilisées par un type de vue.

$R+P+play - - > m_v(a)$: L'ensemble des *roleInstances*, l'ensemble des *playerIntances* et la relation $play$ associant ces deux éléments correspondent à la fonction $m_v(a)$ représentant l'application d'un point de vue.

Les liens présentés ici sont incomplets et unidirectionnels mais permettent tout de même d'adapter une partie des axiomes et propositions présentés précédemment à la formalisation de point de vue. Par exemple l'axiome A4 affirmant qu'une relation entre un *roleInstance* et un *playerInstance* doit être définie au niveau modèle se traduit par le fait qu'une vue sur un objet doit être conforme à un point de vue. En particulier le type de l'objet observé doit correspondre au domaine de la fonction de filtrage du point de vue, ce qui donne formellement :

$$\forall a \in A, \forall vt =< \_, \_, \_, f_{vt} >\in \mathcal{VT}, a \in domaine(m_{vt}) \Rightarrow a \in domaine(f_{vt}) \quad \text{(A4')}$$

Dans la section 4.1.1 nous décrivons comment Role4All permet de modéliser des points de vue. Nous illustrons ici que le rapprochement des notions de *rôles* et de points de vue, s'applique également au niveau formel.

Finalement, la formalisation mathématique de la notion de rôle présentée dans cette section suit les préconisations identifiées dans notre état de l'art (§2.4). Surtout, elle nous permet de justifier formellement l'utilisation des rôles pour la création de point de vue. Par ailleurs, la formalisation de notre approche offre la possibilité de mettre en place des solutions d'analyse formelle pour identifier des scénarii d'attaque. L'implémentation de telle solution n'est cependant pas directement discutée dans ce manuscrit mais préconisée comme perspective à ce travail.

4.1.3 Implémentation de Role4All

Comme justifié lors de la formalisation de Role4All 4.1.2, nous avons choisi de développer un DSL interne à Pharo. Pharo [1] est un langage de programmation orienté objet fortement inspiré de Smalltalk. Il s'agit d'un langage réflexif, qui admet un typage dynamique et dans lequel tout est objet. Ces caractéristiques, associées à des outils de débogage très puissants, font de Pharo un excellent langage pour le prototypage. Nous présentons donc dans cette section l'implémentation de Role4All comme DSL interne à Pharo.

Notre implémentation s'articule autour de sept classes principales à savoir : *Player*, *Role*, *PlayRelation*, *Adapter*, *AssociationRule*, *Context* et *Organiser*. Dans la suite de cette section nous présentons succinctement chacune de ces classes.

4.1.3.1 Présentation de la classe *Player*

```
1       Object subclass: #Player
2       instanceVariableNames: 'playRelations playerId'
3       "CoreMethodes: 'play: play:underCondition:'"
4
5       Player subclass: #Type
6       instanceVariableNames: ''
7       "CoreMethodes: ''"
```

La classe *Player* définit 14 méthodes dont les principales sont *plays:* et *plays: underCondition:*. Pour une instance d'un *player P*, la fonction *plays:* prend comme entrée un *rôle R* et génère une *playRelationIntance*, un *roleIntance* conforme à *R* et un *adapterIntance* conforme à l'*adapter* traduisant les propriétés de *P* en propriétés de *R*. La fonction *plays:underCondition:* possède le même comportement mais y associe une condition, la mise en place de la relation est possible uniquement si cette dernière est respectée.

La classe *Player* référence la classe *PlayRelation* conformément au métamodèle de Role4All. De plus, un *playerInstance* possède un identifiant, ce dernier est unique et permet de dissocier deux instances d'un même *player*. Cet identifiant est utilisé par la fonction *broadcast:* de la classe *Role*.

La classe *Type* hérite de *Player* et représente les *players* qui ne sont pas des *rôles*.

4.1.3.2 Présentation de la classe *Role*

```
1       Player subclass: #Role
2       instanceVariableNames: 'playRelation containedRoles
3       containerRole roleId broadcastIds'
4       "CoreMethodes: 'attachTo: attachTo:underCondition:
5       detach broadcast: addToContext: addAttribute: doesNotUnderstand:'"
```

Dans notre approche, un *rôle* est un *player*, ce qui s'illustre ici par la relation d'héritage entre les classes *Role* et *Player*.

La classe *Role* définit 46 méthodes dont les principales sont *attachTo:*, *attachTo: underCondition:*, *detach* et *broadcast:*. La méthode *attachTo:* permet de créer une association entre un *roleIntance* et un *playerInstance* via une *playRelationIntance*, la méthode *attachTo: underCondition:* a le même comportement mais soumis à une condition. *AttachTo:* est comparable à la méthode *play:*, soit un *playerIntance play:* un *rôle*, soit un *roleIntance* est *attachTo:* à un *playerIntance*, et il en est de même pour la méthode *attachTo: underCondition:*. La méthode *detach* permet de dissocier un *roleIntance* de son *playerIntance*. Les méthodes d'association et de dissociation permettent de modifier dynamiquement les relations entre instances.

1. Pharo : https ://pharo.org/

La méthode *broadcast:* permet d'envoyer un message à l'ensemble des *roleIntances* contenus dans un *containerRole*. Elle est utilisée afin de propager des messages à un ensemble de *playerInstances*.

Afin d'éviter les boucles et pour garder une trace des messages échangés, nous affectons un identifiant à chaque message diffusé par cette méthode. La variable d'instance *roleId* ainsi que l'identifiant des *playerIntances* sont utilisés pour identifier l'émetteur d'un message. Ce dispositif permet de retracer le parcours de chaque message échangé entre différentes instances.

Comme présenté dans la section 4.1.1, l'utilisation de *containedRoles* et de *containerRoles* permet d'envoyer un message à un ensemble de *playerIntances* sans connaissance préalable de l'instance possédant l'information, et sans même savoir si le message peut être traité. Les limites soulevées dans la section 4.1.1 sont levées grâce à l'utilisation d'identifiants permettant d'éviter les boucles dans les envois de messages et d'ordonner leur envoi. De plus, la méthode *broadcast:* identifie les instances qui ont répondu à un message. Cette information pourrait être utilisée par un algorithme afin d'optimiser leurs envois, cependant cette fonctionnalité n'est pas implémentée à l'heure actuelle.

La méthode *doesNotUnderstand :* est appelée lorsqu'une méthode n'est pas encore implémentée dans un *rôle*. L'appel est d'abord transmis aux *containedRoles*, s'ils ne sont pas capables de traiter l'information, un message d'erreur est envoyé. Lors de la diffusion d'un message avec la méthode *broadcast:*, si l'ensemble des réponses obtenues sont des messages d'erreur, alors nous levons une erreur (absence de réponse), sinon nous transmettons à l'émetteur du message la réponse obtenue. Dans le cas où plusieurs réponses sont reçues et qu'elles ne sont pas cohérentes, nous levons également une erreur (problème de cohérence).

4.1.3.3 Présentation des classes *PlayRelation* et *Adapter*

Classe *PlayRelation*

```
Object subclass: #PlayRelation
instanceVariableNames: 'player role adapter''
"CoreMethodes: 'createAdapterMethod: newMessage:'"
```

La classe *PlayRelation* fait référence aux classes *Player*, *Role* et *Adapter*. Elle définit également 10 méthodes dont les principales sont *createAdapterMethod:* et *newMessage:*. Ces deux méthodes permettent de créer de nouvelles méthodes d'adaptation lors d'une exécution. Cela permet de modifier le fonctionnement d'un *adapter* et donc d'ajuster dynamiquement la relation entre un *roleInstance* et un *playerInstance*.

Classe *Adapter*

```
Object subclass: #Adapter
instanceVariableNames: ''
"CoreMethodes: ''"
```

La classe *Adapter* ne définit pas directement de méthodes, seuls les *adapters* construits comme des sous-classes définissent un comportement. Un *adapter* contient les méthodes permettant de traduire les propriétés d'un *player* en propriétés de *rôle* et inversement. Notre approche nécessite donc de créer un *adapter* pour chaque relation entre un *rôle* et un *player*. La relation d'héritage permet de simplifier la création de nouveaux *adapters* via la spécialisation d'un *adapter* préexistant pour une nouvelle relation.

4.1.3.4 Présentation des classes *AssociationRule*, *Context* et *Organiser*

Classe *AssociationRule*

```
Object subclass: #AssociationRule
instanceVariableNames: 'player role'
"CoreMethodes: ''"

AssociationRule subclass: #AdapterAssociationRule
instanceVariableNames: 'adapter'
"CoreMethodes: ''"

AssociationRule subclass: #RoleAssociationRule
instanceVariableNames: 'condition'
"CoreMethodes: ''"
```

La classe *AssociationRule* et ses sous-classes *AdapterAssociationRule* et *RoleAssociationRule* permettent de définir des règles d'association d'*adapters* et de *rôles*. Une instance d'*AssociationRule* décrit la relation entre un *roleInstance* et un *playerInstance*. Une instance d'*AdapterAssociationRule* décrit l'allocation d'un *adapterInstance* à un *playRelationInstance*. Une instance de *RoleAssociationRule* représente une règle décrivant l'attribution d'un *rôle* à un *playerInstance* sous une condition donnée. Ces règles servent à spécifier, via un *rôle*, l'état d'un *player*, ce qui correspond à la caractéristique de *rôle* numéro 10 (voir annexe B). Elles permettent également de restreindre l'accès à certaines fonctionnalités (caractéristiques 11 et 12). De plus, le fait de jouer un *rôle* pour un *playerInstance* peut être contraint (caractéristique 6). Par exemple, pour une équipe de foot, un individu ne peut jouer le *rôle* de capitaine que s'il joue déjà le *rôle* de membre de l'équipe.

Classe *Context*

```
Object subclass: #Context
instanceVariableNames: 'adaptationDefinitions allocationRules
roles'
"CoreMethodes: 'addAdapter:to:whenLinkedTo:
addRole:to:underCondition:'"
```

La classe *Context* regroupe un ensemble de *rôles* et de règles d'association. Un même rôle peut faire partie de différents *context* et en fonction de ce dernier, les règles d'association avec un *player* sont différentes. Par exemple, un commandant joue le *rôle* de décideur dans le *context* d'un navire mais, dans le *context* d'une flotte, c'est l'amiral qui joue ce *rôle*. Pour la modélisation du point de vue d'un attaquant, le *context* permet de définir ces spécificités. Changer de *context* revient alors à changer de type d'attaquant.

Classe *Organiser*

```
Object subclass: #RoleOrganizer
instanceVariableNames: 'players contexts attachedRoles
attachedAdapters'
"CoreMethodes: ''"
```

La classe *Organiser* est un singleton[1] qui permet de construire la liste initiale des *rôles*, *adapters*, *players* et *contexts* utilisés lors d'une exécution.

Finalement, l'implémentation du métamodèle de Role4All comme DSL interne à Pharo représente 10 classes, auxquelles s'ajoutent 5 classes permettant la construction d'une interface graphique. Ce code compact justifie également le choix d'un DSL interne car on observe ici qu'avec peu de code, nous prototypons entièrement un langage. En plus, nous avons défini de nombreuses classes utilisées pour mettre en place notre framework basé sur des modèles de rôle représentant nos PVR, PVS et point de vue d'interprétation.

1. Un singleton est une classe qui ne possède qu'une seule instance

4.2 Modélisation du point de vue avec Role4All

Dans le chapitre 3, nous présentons notre approche de modélisation du point de vue d'un attaquant via l'utilisation de points de vue de références (PVR) et de points de vue spécifiques (PVS). Dans la section 4.1 nous décrivons, formalisons et implémentons Role4All, notre solution de modélisation de points de vue utilisant la notion de rôle. Dans cette section, nous nous focalisons sur la modélisation du point de vue d'un attaquant avec Role4All. Nous présentons tout d'abord en quoi Role4All permet de modéliser des points de vue et en particulier celui d'un attaquant. Puis nous décrivons comment utiliser notre approche pour construire des PVR et des PVS. Au cours de cette section nous justifions le choix des caractéristiques de rôles couvertes par Role4All, à savoir les n°1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 19, 20, 21, 25 et 26.

4.2.1 Utilisation de Role4All pour modéliser un point de vue

Nous mettons en évidence dans la section 3.4.3 les caractéristiques propres à la description du point de vue et de la vue d'un attaquant. Nous décrivons ici en quoi la modélisation par les rôles, proposée par Role4All, permet de couvrir l'ensemble de ces caractéristiques en particulier via la couverture des caractéristiques de rôles n°1, 5, 6, 7, 10, 11, 12, 13 et 15.

4.2.1.1 Point de vue d'un attaquant par les rôles

Nous avons identifié quatre ensembles de caractéristiques nécessaires à la description de points de vue d'attaquants, à savoir :

— ① Multiple Metamodel Arity
— ② Viewtype Manifestation Materialized
— ③ Viewtype language Opération Add, Modify and Filter
— ④ Viewtype language Paradigm Extentional

① décrit le fait qu'un point de vue est composé d'un ensemble de concepts, autrement dit, il n'est pas focalisé sur un seul métamodèle source mais il en représente tout un ensemble. Dans Role4All, cette caractéristique se traduit par le fait qu'un *rôle* est composé de concepts pouvant provenir de différentes sources, cela se manifeste par le fait qu'un *rôle* possède ses propres concepts et qu'il peut être joué par différents types de *players*. Cette spécificité correspond aux caractéristiques de rôles n° 1 et 7 (voir annexe B).

② décrit le fait qu'un point de vue est composé de concepts définis localement et non pas que de références vers des concepts des métamodèles sources. Dans le contexte de la modélisation par les rôles, cela se traduit par le fait qu'un *rôle* possède un comportement et des propriétés qui lui sont propres, ce qui correspond à la caractéristique n°1.

③ décrit le fait qu'il est possible d'ajouter, de modifier et de filtrer des données ou des concepts dans un point de vue. Notre approche permet d'ajouter ou de supprimer un *rôle* en cours d'exécution grâce à l'*Organiser*, un singleton qui référence les *rôles* jouables lors d'une exécution. De plus, les notions de filtre et de modification sont facilitées par le fait qu'un *rôle* peut spécialiser un *rôle*. Cette notion de spécialisation est symbolisée par la caractéristique n°13.

④ décrit le fait qu'un point de vue définit explicitement le type des éléments pouvant appartenir à une vue. Dans Role4All, nous utilisons des *adapters* définissant au niveau conceptuel la transformation des propriétés d'un *player* en propriétés de *rôle* et inversement. Les éléments pouvant appartenir à une instance de *rôle* sont donc définis au niveau conceptuel à l'aide d'un *adapter*. Cette caractéristique ne se traduit pas directement

en caractéristiques de rôle mais se reflète dans le métamodèle de Role4All via le concept d'*adapter*.

Finalement, l'ensemble des caractéristiques propres à la description du point de vue d'un attaquant sont couvertes par Role4All. Nous mettons en évidence le lien entre point de vue et *rôle* en rapprochant les caractéristiques de description de point de vue et celles de modélisation par les rôles dans le cadre de Role4All. Il est à noter que certaines caractéristiques propres au point de vue d'un attaquant ont un impact direct sur le métamodèle de Role4All bien qu'elle ne se traduisent pas directement en caractéristiques de rôles.

4.2.1.2 Modélisation de la vue d'un attaquant grâce à des *roleInstance*

Nous avons identifié en section 3.4.3 sept ensembles de caractéristiques nécessaires à la description de la vue d'un attaquant, à savoir :

— ⑤ Intrisiveness Non-Intrusive
— ⑥ Runtime Consistency Direction Model to View and View to Model
— ⑦ Multiple Model Arity
— ⑧ Closedness Augmenting and Query Language Opération Selection, Projection, Join, Aggregate and Rename
— ⑨ Query language Paradigm Intensional Explicit
— ⑩ View Manifestation Virtual
— ⑪ Runetime Consistency Recomputation Incremental and Frequency Immediate

⑤ décrit le fait que la mise en place d'une vue sur un modèle source est non-intrusive. Dans Role4All, les *rôles* ont leur propre identité, indépendante de celles des *players* et nous utilisons la notion de *playRelation* pour décrire les relations entre ces deux éléments. Un *playerInstance* peut être associé ou dissocié d'un *roleInstance* dynamiquement et ce sans impact sur le *player* instancié. Cette particularité est décrite par les caractéristiques de rôles n°5 et 15.

⑥ décrit le fait qu'une vue puisse interagir sur un modèle source et que les modifications qu'on y apporte se répercutent sur les vues qui lui sont associées. Dans un contexte de modélisation par les rôles, cela signifie qu'un *roleInstance* peut interagir avec un *playerInstance* et que les modifications apportées à ce dernier sont visibles sur les *roleIntances* associés. Dans Role4All, ces interactions sont rendues possibles par les concepts de *playRelations* et d'*adapters* bidirectionnels. Le fait qu'un *rôle* puisse interagir avec un *player* est décrit par les caractéristiques n°10 et 11.

⑦ décrit le fait qu'une vue regroupe des informations provenant de différents modèles sources. Dans Role4All, une vue est modélisée par un *roleInstance* qui n'est relié au maximum qu'à un seul *playerIntance*. Cependant, nous utilisons la notion de contenance afin de définir un *containerRole* qui contient différents *containedRoles*. Cette relation permet de regrouper dans une même instance des informations provenant de différents *playerInstances*. Cette caractéristique ne se traduit pas directement en caractéristiques de rôles mais se reflète tout de même dans le métamodèle de Role4All au travers la relation de contenance.

⑧ décrit le fait qu'une vue peut contenir des informations qui lui sont propres ou calculer à partir des modèles sources. En terme de *rôle*, cela signifie qu'un *roleInstance* peut contenir des informations qui ne sont pas une référence directe aux données contenues dans des *playerInstances*. Dans notre approche, un *rôle* possède ses propres propriétés, une instance de *rôle* peut alors contenir des informations ne provenant pas directement de *players*. De plus, un *rôle* possède son propre comportement ce qui permet, entre autre, de

sélectionner, fusionner, altérer ou calculer des informations à partir des données provenant de différents *players*. Ces diverses propriétés sont décrites par les caractéristiques n°10, 11 et 12.

⑨ décrit le fait qu'une vue peut contenir tous types d'éléments à condition qu'ils respectent certains prérequis définis explicitement. Dans notre approche, nous utilisons les concepts d'*AssociationRule* et de *Context* permettant de décrire les conditions nécessaires afin de permettre une relation de jeu entre un *playerInstance* et un *rôle*. Notre approche permet alors de décrire explicitement si un *player* ou un *rôle* peut ou non participer à une *playRelation*. Cette spécificité correspond à la caractéristique n°6.

⑩ décrit le fait qu'une vue ne duplique pas les données contenues dans les modèles sources mais qu'elle est composée uniquement de références vers ces dernières. Dans Role4All, les informations provenant des modèles sources contenus dans un *roleInstances* sont des références vers des *playerInstances*. Cette spécificité est assurée par l'utilisation de *playRelations* et d'*adapters* qui permettent, entre autres, de mettre en forme des informations contenues dans un *playerInstance* pour un *roleInstances*. Cette caractéristique ne se traduit pas directement en caractéristiques de rôle mais se reflète dans le métamodèle de Role4All via les concepts de *playRelation* et d'*adapter*.

⑪ décrit le fait qu'à chaque changement apporté à un modèle source, les modifications sont immédiatement prises en compte par les vues associées et que ces dernières ne sont pas entièrement recalculées mais seulement mises à jour. Dans un contexte de modélisation par les rôles, cela se traduit par le fait que, lorsqu'un *playerInstance* est modifié, les *roleInstances* en relation avec ce dernier sont immédiatement informés de la modification et qu'ils n'ont pas à être instanciés de nouveau. Dans Role4All, ce comportement est directement induit par le fait que les *roleInstances* ne contiennent que des références vers les informations contenues dans les *playerInstances*. La transmission d'une modification apportée à un *playerInstance* est alors directement appliquée et ne requiert aucun comportement particulier. Cette caractéristique est directement supportée par Role4All car ce dernier couvre ⑩.

Finalement, l'ensemble des caractéristiques nécessaires à la description de vues sont couvertes par Role4All, en particulier celles requises pour décrire la vue d'un attaquant. Nous justifions également l'utilisation des concepts de *playRelation*, *adapter*, *AssociationRule* et de *Context* ainsi que la relation de contenance dans notre implémentation de Role4All. Le tableau D.1 présenté en annexe D résume les implications du support des caractéristiques de description de points de vue et de vues d'attaquants.

4.2.2 Application à la génération de PVR

Dans le chapitre 3, nous construisons quatre points de vue de référence (PVR) pour permettre de représenter l'ensemble des préoccupations d'un attaquant. Les PVR mettent en forme des informations provenant de sources hétérogènes, dans Role4All les PVR sont modélisés par des modèles de rôle et les sources hétérogènes par des *players*. Dans cette section, nous présentons comment Role4All permet de modéliser des PVR, d'adapter des sources hétérogènes et de fédérer des informations complémentaires ou redondantes. De plus, nous justifions notre choix de couvrir les caractéristiques de rôles n°3, 4, 9 et 16.

4.2.2.1 Modélisation des PVR par les rôles

Dans la section 3.1.2 nous spécifions que les PVR sont construits indépendamment des métamodèles sources. Ce qui signifie qu'un PVR ne contient que des concepts qui lui sont propres. Dans la section 4.1.2 nous spécifions qu'un *rôle* peut référencer, contenir ou hériter

d'un autre *rôle*. Nous utilisons ici ces relations afin de construire des modèles de rôles représentant les PVR.

Nous construisons nos modèles de rôle conformément aux notations de CROM [105] présentées dans la section 2.4 enrichies des concepts propres à Role4All, à savoir les notions de *roleInstance*, de *playerIntance*, de *playRelation* et d'*adapter*, ainsi que les relations de "joue le rôle de" et de "contenance". La figure 4.5 présente les notations graphiques utilisées dans le reste de ce manuscrit pour la construction de modèles de rôles dans Role4All. Dans la suite de ce manuscrit, un modèle de rôles dans Role4all respectant les notations présentées en figure 4.5 sera appelé un modèle de *rôle*.

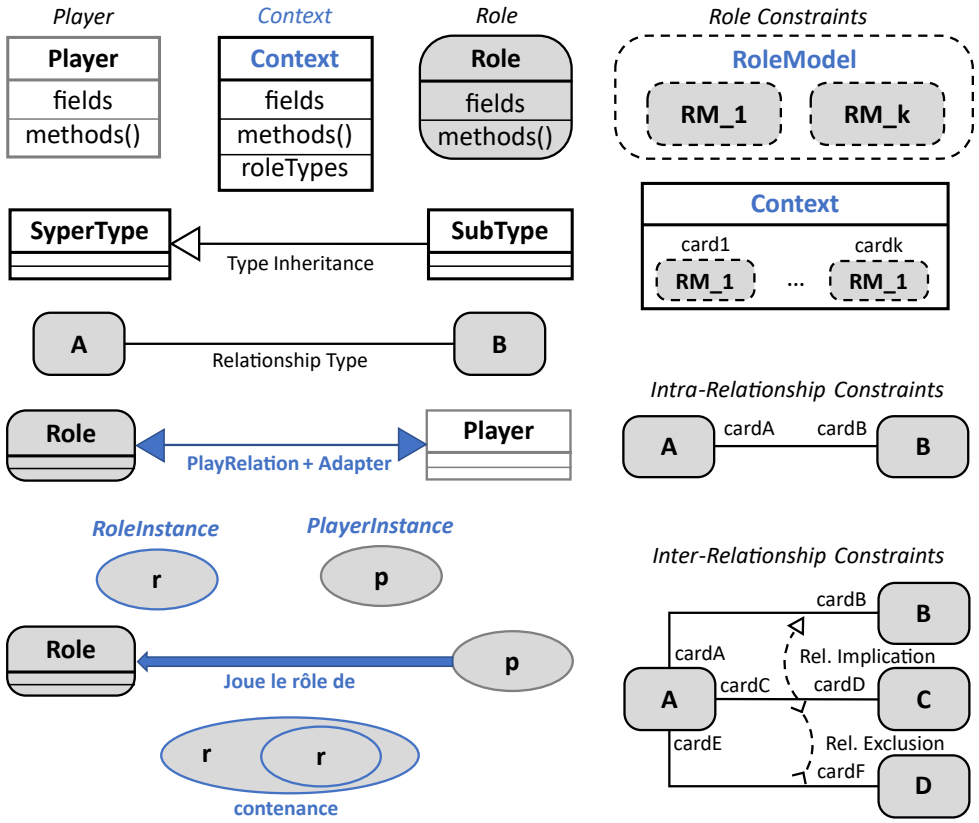


FIGURE 4.5 – Notation graphique utilisée pour la construction de modèles de *rôles*

Un modèle de *rôle* représente un ensemble de *rôles* et de relations. Ces dernières peuvent être contraintes, par exemple via des cardinalités. De plus, chaque *rôle* peut être lié à un ou plusieurs *players* via une *playRelation* et un *adapter*. En outre, le lien de contenance permet de créer des ensembles de *roleInstance* fonctionnant comme une seule entité sans connaissances préalables de la localisation des informations. L'instanciation des *playRelation* ainsi que l'utilisation de la contenance sont détaillées dans les sous-sections suivantes.

Pour illustrer notre notation, nous prendrons comme exemple la construction du modèle de *rôle* représentant le PVR système comportant neuf *rôles* et conceptualisant les notions de *relation*, de *composant*, d'*acteur*, et d'autres concepts représentés en figure 4.6.

Chaque concept de référence du PVR est modélisé par un *rôle* en relation avec différents *players*. Un même *rôle*, par exemple "*Composant*", peut être en relation avec différents *players*. Cette multiplicité correspond aux caractéristiques n°3 et 4 (annexe B).

Sur ce modèle de *rôle*, nous restreignons les relations entre certains éléments. Par exemple, un "*Composant*" contient au maximum une "*Configuration*". L'utilisation de cardinalité dans les modèles de *rôles* est caractérisée par la caractéristique n°16. Dans cet exemple, nous référençons quatre *players* distincts qui contiennent les informations néces-

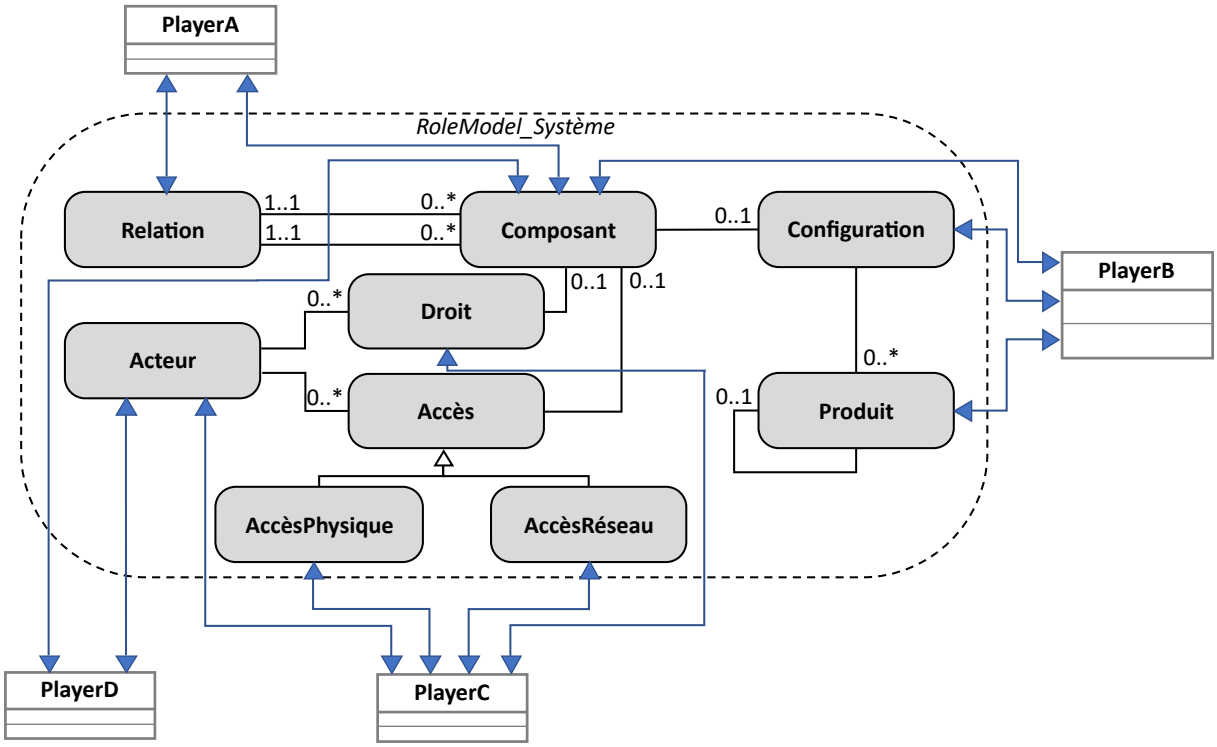


FIGURE 4.6 – Modélisation par les *rôles* du PVR système et relations avec les *players*

saires à la construction des différents *rôles* de notre modèle. Nous détaillons leur choix et leur utilisation dans le chapitre 5. Cependant, ce choix n'est pas fixe et il est possible de changer à tout moment l'un des *player* utilisé. Les *rôles* en relation avec l'ancien *player* sont alors transférés au nouveau. Cette fonctionnalité, propre à Role4All correspond à la caractéristique n°9.

Comme présenté dans la section 2.4.2, l'utilisation de *rôles* en remplacement du concept de classe classique permet de reporter la décision de matérialisation des données jusqu'au moment de l'intégration des *players*. L'utilisation de *rôles* permet alors de simplifier les problèmes de réplication et de cohérence des données, un *rôle* ne faisant pas de distinction entre les données physiquement représentées et celles obtenues à partir d'un autre objet [143]. Cela permet de construire des PVR indépendamment des métamodèles sources, il est alors possible de changer une source sans modifier le PVR. En outre, la relation "joue le rôle de" entre un *playerInstance* et un *rôle* permet d'ajouter dynamiquement de nouveaux *player*, potentiellement associés à l'intégration de nouveaux outils. Elle permet de créer, en cours d'exécution, un nouveau couple *playRelation* + *adapter* afin d'adapter les données contenues sur un nouveau *player*.

Finalement, Role4All permet de construire un modèle de *rôle* représentant un PVR indépendamment des métamodèles sources. Les *rôles* composant ce modèle peuvent être joués par des *playerInstances* de tous types, la seule contrainte étant la création préalable d'un *adapter* entre le *player* instancié et le *rôle* joué.

4.2.2.2 Adaptation des métamodèles sources

La relation entre un *rôle* et un *player* est effectuée par une *playRelation* et un *adapter*. Le premier matérialise la relation et le second contient le code relatif aux transformations permettant de traduire les propriétés d'un *player* en propriétés d'un *rôle*. Nous soulignons

dans le chapitre 3 que, pour la modélisation du point de vue d'un attaquant, l'adaptation entre un *rôle* et un *player* doit être bidirectionnelle et dynamique.

Un *adapter* définit deux types de fonctions : des accesseurs permettant d'adapter les informations d'un *player* et des mutateurs permettant de mettre à jour, depuis un *rôle*, les informations contenues dans un *player*. Si les fonctions d'un *adapter* sont réversibles, alors, un accesseur et un mutateur peuvent former une relation bijective entre le *rôle* et le *player*. Par exemple, si un *player* contient comme information une température en °K et que le *rôle* adapte cette information pour fournir une température en °C, l'accesseur et le mutateur associé seraient :

```
accesTemp: obj1
↑ (obj1 temp + 273.15).

mutatTemp: newTemp player: obj1
obj1 temp: (newTemp - 273.15).
```

Dans le cas d'une fonction non réversible, un accesseur n'est pas forcément lié à un seul mutateur. Par exemple, dans le cas où un *player* contient comme information un poids et une taille et qu'un *rôle* représente un Indice de Masse Corporelle (IMC), l'accesseur et les mutateurs seraient :

```
accesIMC: obj1
↑ (obj1 poids / (obj1 taille * obj1 taille) ).

mutatIMC: newIMC player: obj1
self error: "Il est impossible de modifier directement la valeur de IMC, voir
    mutatTaille: ou mutatPoids:".

mutatTaille: newTaille player: obj1
obj1 taille: newTaille.

mutatPoids: newPoids player: obj1
obj1 poids: newPoids.
```

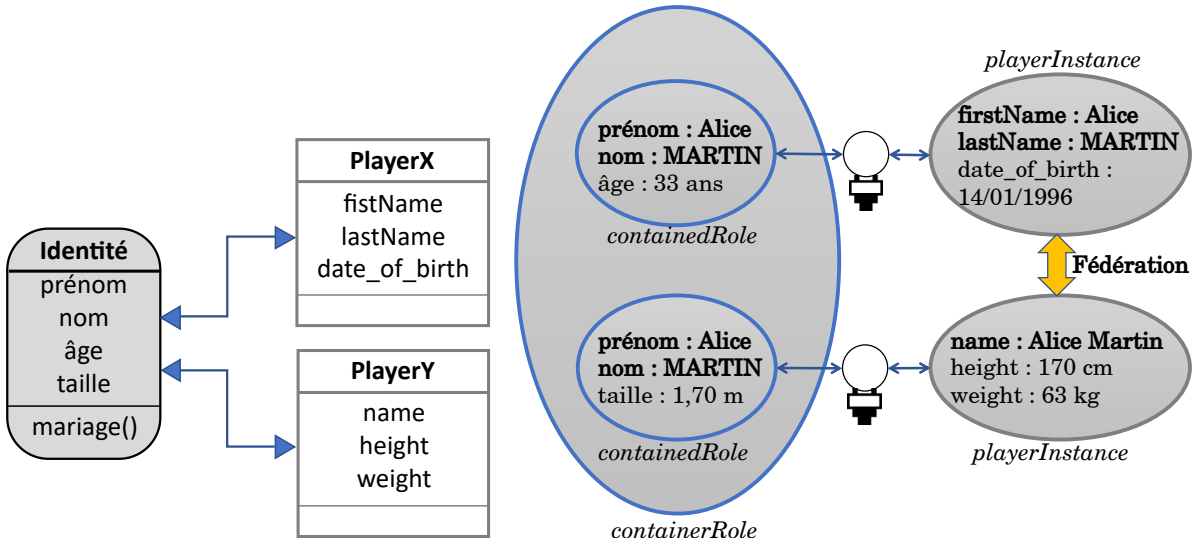
Dans cet exemple, il est impossible de changer directement l'IMC car ce dernier est une propriété dérivée de la taille et du poids. En revanche, il reste possible de faire varier sa valeur en modifiant le poids ou la taille via les fonctions *mutatTaille :* et *mutatPoids :*.

Bennett [20] prouve dans ses travaux que tout calcul irréversible peut être simulé de manière réversible. Cela signifie dans notre contexte qu'il est toujours possible de créer des mutateurs permettant de modifier des propriétés dérivées.

Finalement, un *adapter* est un ensemble d'accesseurs et de mutateurs permettant d'adapter les informations d'un *player* pour un *rôle* et de transmettre les modifications apportées à un *rôle* au *player*. L'adaptation entre un *rôle* et un *player* est alors bidirectionnelle. Il est possible de définir différents *adapter* pour une même *playRelation*, le choix de l'adaptation utilisée dépendant alors du *context*. De plus, le *rôle* et le *player* d'une *playRelation* sont définis indépendamment de l'*adapter* utilisé. Par conséquent, il est possible de modifier l'adaptation d'un *player* directement au cours d'une exécution, c'est pourquoi nous qualifions cette adaptation de dynamique. La solution d'adaptation proposée permet donc de remonter des informations d'un *playerInstance* à un *roleInsntance*. Une fois référencées dans des *roleInsntances*, les informations sont de fédérer à l'aide de *containerRoles*.

4.2.2.3 Fédération des sources de données hétérogènes

Nous présentons dans la section 4.1.1 comment Role4All permet, via l'utilisation de *containerRoles* et de *containedRoles*, de regrouper dans un même *roleInstance* des informations provenant de sources hétérogènes. Cette relation de contenance nous permet également de fédérer des informations provenant de différents *playerInstances*.



(a) Relation niveau métamodèle (b) Relation niveau modèle

FIGURE 4.7 – Exemple de fédération d'informations via les *roleInstances*

Pour illustrer les principes de la fédération par les rôles, nous prenons comme exemple la création d'une carte d'identité représentée par le *rôle* "*Identité*" composé d'un nom, d'un prénom, d'un âge et d'une taille. Ce *rôle* est lié à deux *players* respectivement "*PlayerX*" et "*PlayerY*". La figure 4.7(a) schématise cet exemple. *PlayerX* et *PlayerY* contiennent des informations complémentaires et redondantes permettant de décrire Alice MARTIN. Dans cet exemple, nous regroupons ces informations dans un unique *roleIntance* afin de construire la carte d'identité d'Alice MARTIN.

Comme illustré en figure 4.7(b), nous créons tout d'abord deux *containedRoles* référençant les informations contenues dans les *playerInstances* représentant Alice. Puis, nous regroupons ces informations dans un *containerRole*. Ce dernier contient alors l'ensemble des données permettant de créer la carte d'identité d'Alice. Les relations mises en place nous permettent, à partir du *containerRole*, d'interagir avec les différents *playerInstance*. Ces interactions peuvent être de la modification ou de la collecte d'information.

La modification d'informations se fait indépendamment des données préexistantes. Par exemple, si Alice se marie et change de nom, l'information du changement de nom est envoyée au *containerRole* qui transmet le message aux deux *containedRoles* qui, à leurs tours, éditent les informations contenues dans les *playerInstances*. Quelles que soient les données préexistantes sur les *playerInstances*, la modification d'informations est toujours possible.

Cependant, la collecte d'informations est quant à elle dépendante des données préexistantes. Si ces données ne sont pas cohérentes, la collecte d'information est impossible. Par exemple, si le nom d'Alice n'est pas le même sur les deux *playerInstances* il n'est pas possible de déterminer le "vrai" nom d'Alice. Ce problème est connu et il existe différentes solutions pour y répondre tels que [61] les classificateurs par vote rigide, les approches par empilement ou encore l'apprentissage automatique. Cependant, dans notre approche nous fédérons les données des *players*, nous ne les dupliquons pas. Toute modification appliquée à un *playerInstance*, quelle qu'en soit la source, est alors instantanément propagée à l'ensemble des éléments fédérés. Cette solution permet, en théorie, d'éviter des problèmes de cohérence lors de la collecte d'information. Cependant, cette solution présente tout de même quelques limitations, en particulier sur la cohérence de la propagation et la mise en place de la fédération. Dans notre approche, la responsabilité de la cohérence des données

fédérées lors de la mise en place d'une relation de contenance est imputée à l'utilisateur. Nous admettons alors donc que, lors de la création d'une telle relation, les données fédérées sont cohérentes.

Pour pallier le problème de propagation des messages, nous proposons dans Role4All de signer et dater l'ensemble des messages. La date associée à un message permet alors d'exécuter le comportement relatif de différents messages dans l'ordre souhaité. Dans notre implémentation, si un message reçu est antérieur au dernier message traité, nous levons une erreur. De plus, si, pour une raison inconnue, nous détectons tout de même un problème de cohérence, nous levons alors une erreur.

Finalement, Role4All permet de fédérer dans un même *roleInstance* des informations provenant de différents *playerInstances* à l'aide de *containerRoles*. Il permet également d'assurer la cohérence des données fédérées.

4.2.3 Génération de PVS

Dans le chapitre 3, nous présentons l'utilisation des points de vue spécifiques (PVS) pour modéliser la perception qu'un attaquant a d'un système et spécifions qu'un PVS est un point de vue sur les PVR. Dans cette section, nous présentons comment sont modélisés les PVS dans Role4All et justifions notre choix de couvrir les caractéristiques de rôles n°8, 19, 20, 21, 25 et 26.

La modélisation de PVS est semblable à celle des PVR avec l'utilisation de *rôles* et de *players*. Contrairement aux PVR qui sont construits indépendamment des métamodèles sources, un PVS est construit grâce à des Concepts Spécifiques (CS) définis comme une représentation des Concepts de Référence (CR). Dans Role4All, un CS est alors modélisé comme *rôle* et nous avons montré dans la section 4.2.2 que les CR sont également modélisés via des *rôles*. Un CS est alors un *rôle* qui est joué par d'autres *rôles*. Cette particularité est caractérisée par la caractéristique n°8.

Pour contextualiser un PVS pour un type d'attaquant particulier nous utilisons la notion de *context*. Elle est utilisée pour représenter les attributs propres à un attaquant, telles que ses aptitudes ou ses connaissances. Un même PVS peut alors être utilisé pour représenter un même point de vue selon différents types d'attaquants. Un PVS est alors représenté par un modèle de *rôles* en relation avec les PVR et par un *context* décrivant les attributs propres à un attaquant. Notre utilisation des *contexts* pour construire un PVS correspond aux caractéristiques 19, 20, 21, 25 et 26 (annexe B).

Nous présentons un exemple de PVS en figure 4.8 avec point de vue de Mallory sur le système fil rouge. Le modèle de *rôle* de la figure 4.8 représente comment Mallory perçoit le système fil rouge. Les *rôles* "*Machine*" et "*Logiciel*" sont une représentation du PVR système. Le *rôle* "*Logiciel Vulnérable*" est une spécialisation du *rôle* "*Logiciel*", il permet de représenter les *logiciels* comportant une vulnérabilité exploitable par l'attaquant. La notion d'exploitabilité est déterminée en fonction du *context*.

Le *context* "*Context_networkAttacker*" utilise les concepts définis dans le PVR attaquant afin d'ajuster le point de vue modélisé pour un certain type d'attaquant. Pour cet exemple, le type est "*NetworkAttacker*", il représente un attaquant externe au système et qui utilise le réseau afin de mener son attaque. Dans l'exemple fil rouge, Mallory est donc un *networkAttacker*. Les attributs du *context* sont ici utilisés pour spécifier la relation entre le *rôle* "Logiciel Vulnérable" et le PVR vulnérabilité. Par exemple, si un *logiciel* contient une vulnérabilité mais qu'un attaquant ne possède pas les aptitudes nécessaires, alors le *logiciel* n'est pas considéré comme un *logiciel vulnérable* dans le *context* de cet attaquant.

Finalement un PVS est une représention d'une représentation, c'est un point de vue sur les points de vue de référence. Dans Role4All, il est modélisé par un modèle de *rôles* associé à un *context*. Un PVS représente la perception d'un attaquant spécifique sur un système,

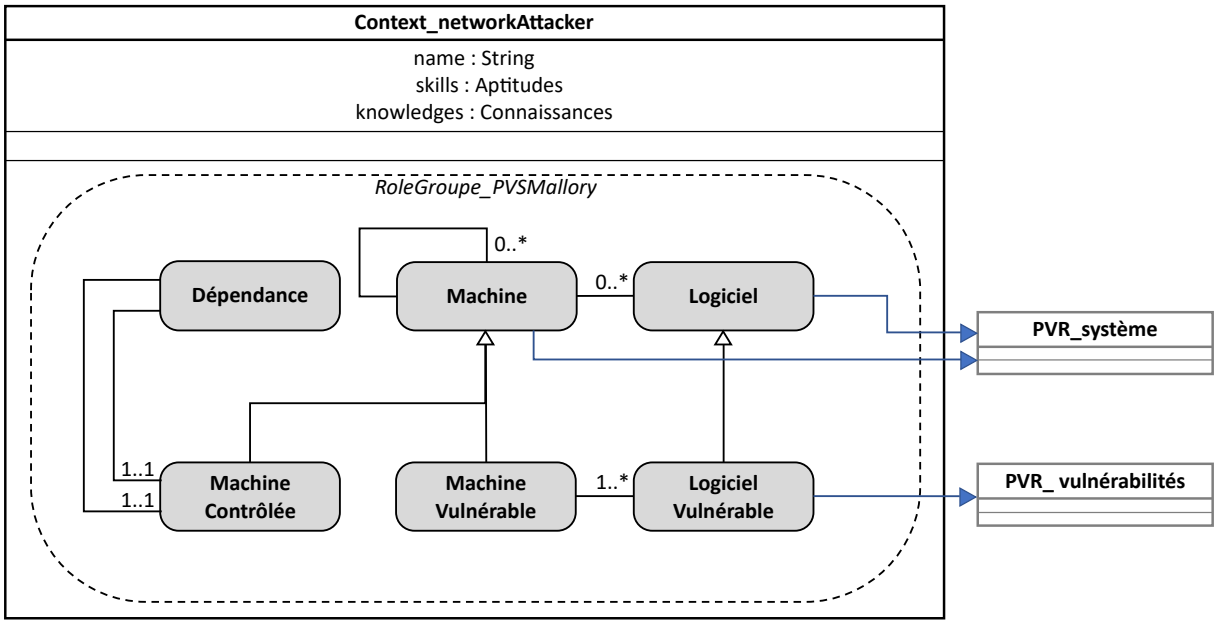


FIGURE 4.8 – Modélisation du PVS de Mallory dans Role4All

il peut être utilisé afin de décrire ce qu'un attaquant particulier perçoit d'un système à un instant particulier. Il peut également être utilisé afin d'analyser le comportement d'un attaquant en fonction de ce qu'il perçoit d'un système.

4.3 Analyse du point de vue d'un attaquant via Role4All

Dans le chapitre 3, nous présentons un ensemble de concepts de référence (CR) lié aux comportements nominaux et malveillants. Afin d'analyser le comportement de l'attaquant lors d'une attaque, il faut être capable de définir et d'exécuter un scénario d'attaque. Pour ce faire, Role4All est en mesure d'interpréter l'évolution d'une attaque et de mettre à jour la vue de l'attaquant en conséquence.

4.3.1 Interpréteur fondé sur les *rôles*

Role4All, afin d'analyser et exécuter le comportement d'un attaquant, permet d'interpréter un modèle de *rôle* en utilisant un point de vue dédié. Dans ce manuscrit, nous proposons un point de vue d'interprétation fondé sur le langage garde-action formalisé par Dijkstra [45]. Ce langage est particulièrement adapté pour l'expression de la sémantique de langage, ceci pour servir de support à l'analyse des programmes modélisés. Chaque pas d'exécution est représenté par une fonction évaluant un prédicat, une garde, sur l'espace d'état du programme et par une fonction appliquant une transformation, une action, sur ce même espace d'état. Ce formalisme permet l'exécution et l'analyse de comportements concurrents et potentiellement non-déterministes.

Dans notre cas, les prédicats sont évalués sur un ensemble de *rôles* constituant l'espace d'état de notre système et les actions sont appliquées sur cet même ensemble de *rôles*. L'espace d'état considéré ici sera celui des différents PVS d'un attaquant.

La figure 4.9 schématise la relation entre les différents points de vue et leur modélisation dans Role4All. Ce schéma illustre bien que l'espace d'état de notre langage garde-action est constitué des *rôles* du PVS de l'attaquant regroupés dans un *context* de Role4All. Dans notre point de vue d'interprétation, constitué de *rôles* dédiés à l'interprétation, les

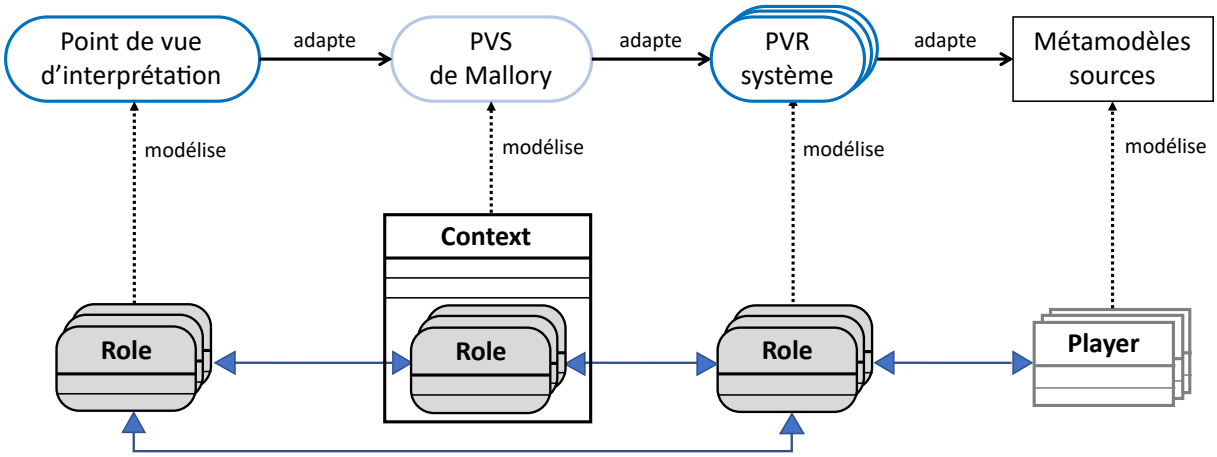


FIGURE 4.9 – Relation entre les différents points de vue utilisés pour analyser le comportement d'un attaquant.

players associés aux *rôles* de *Guard* et d'*Action* sont donc eux-mêmes des *rôles*.

La figure 4.10 présente le modèle de *rôle* décrivant notre point de vue d'interprétation. Le *rôle Program* référence un ensemble de *behaviorBlocks* et un *environment*, correspon-

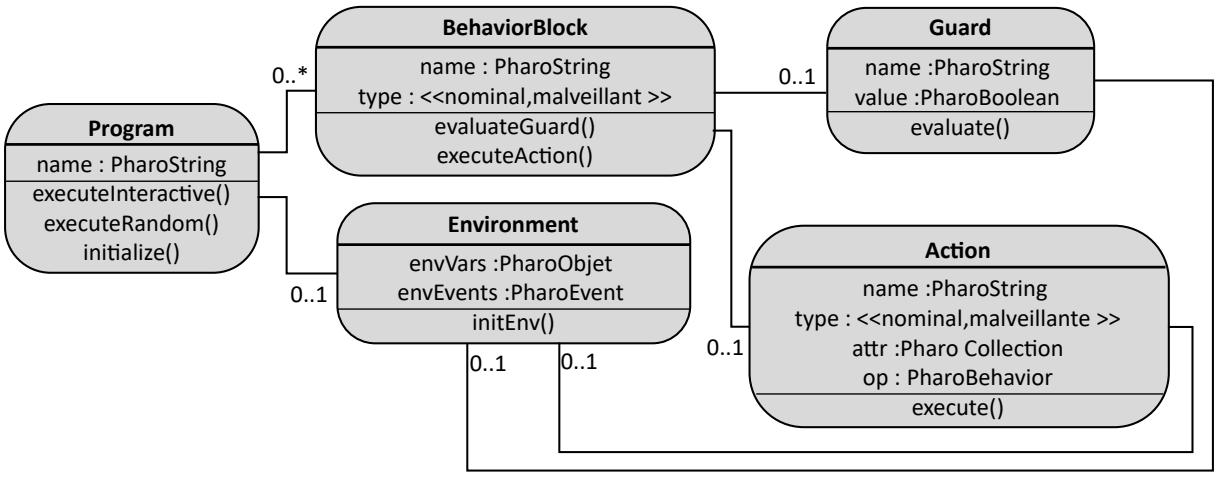


FIGURE 4.10 – Modèle de *rôle* du point de vue d'interprétation

dant respectivement à l'ensemble des prédicats et à l'espace d'état.

Un *behaviorBlock* est composé d'une *garde* et d'une *action*. La fonction nommée *evaluateGuard()* évalue la *garde* d'un *behaviorBlock* et, si le résultat de cette évaluation est vrai, la fonction *executeAction()* évaluer l'*action* associée en modifiant l'espace d'état. Par la suite, les *behaviorBlocks* avec une *garde* évaluée à vrai seront dits déclenchables. L'attribut "*type*" du *rôle BehaviorBlock* permet de différentier les comportements nominaux des malveillants.

Une *garde* est un prédicat sur l'environnement. Elle évalue des *rôles* et des objets propres à l'environnement, l'implantation sous forme d'objet impose l'évaluation de méthodes retournant un booléen.

Une *action* représente l'évaluation d'un comportement entraînant des changements de l'environnement par : la modification de variables d'environnement ou l'appel de méthodes de *players*.

Un *environment* contient un ensemble de *rôles*, d'objets et d'événements constituant l'espace d'état du programme en cours d'exécution. Cet espace d'état sert de support à

l'analyse et contient une fonction *initEnv()* qui fournit les valeurs initiales de notre exécution pour le système et le scénario considéré.

Le *rôle Program* définit deux types d'exécution : l'exécution aléatoire, au cours duquel un *behaviorBlock* est sélectionné aléatoirement parmi ceux déclenchables, et l'exécution interactive, qui permet à l'utilisateur de sélectionner un *behaviorBlock* à exécuter parmi ceux déclenchables. Cette seconde solution offre alors la possibilité d'exécuter des scénarii contrôlés par l'utilisateur. À chaque pas d'exécution, toutes les *gardes* sont évaluées offrant ainsi un choix non-déterministe au programme.

Pour l'exécution d'un scénario, deux ensembles de *gardes-actions* sont pris en compte par l'interpréteur : un relatif à l'environnement, qui permet à celui-ci d'évoluer indépendamment des comportements externes, et un relatif à un *player*, accessible grâce à un *rôle*. Dans ce second cas, les couples *gardes-actions* évaluent alors des propriétés de *rôles*. Cette évaluation est rendu possible grâce au support de la relation entre un *rôle* et un *player* et à l'algorithme de l'*adapter* associé. Par exemple, pour appliquer notre point de vue d'interprétation au PVS de Mallory, il est nécessaire de définir des *adapters* traduisant les prérequis et les effets d'un *logicielVulnérable* en *gardes* et *actions*.

Finalement, ce point de vue d'analyse permet d'interpréter le comportement d'un attaquant. Il permet également de générer des scénarii d'attaque pouvant être représentés selon différent modèles de sécurité, tels que des arbres d'attaque. De plus, Role4All permet de spécialiser le point de vue d'interprétation pour effectuer des analyses complémentaires, telles que de la vérification de modèles ou des calculs de coût. Dans ce manuscrit, nous limitons cependant les analyses à la production de modèles de sécurité.

4.3.2 Exemple d'interprétation avec le système fil rouge

Nous construisons et exécutons de manière interactive un *program* permettant d'interpréter le comportement de Mallory au cours de son attaque sur le système fil rouge, présenté en section 3.2.

Pour ce faire, nous définissons quatre objets d'environnement dont trois représentent l'état des machines (allumé ou éteint) d'Alice, Bob et Carol et un représentant l'activité de l'attaquant (actif ou inactif). Initialement, les machines d'Alice et de Bob sont allumées et l'attaquant est inactif, ce qui signifie que seules *behaviorBlocks* de type "*nominal*" peuvent être exécutés et ce, uniquement sur les *rôles* représentant les machines d'Alice et de Bob, comme l'illustre la figure 4.11. Nous définissons également deux *behaviorBlocks* permettant de démarrer ou de stopper l'attaque : '*Start Attack*' et '*Stop Attack*'. Nous associons à chaque *machine X* du système les *behaviorBlocks* '*Start_X*' et '*Shutdown_X*' permettant d'allumer et d'éteindre la machine. Nous associons à chaque *machine X* découverte par l'attaquant le *behaviorBlock* '*Scan_X*' qui permet, en fonction des droits et des connaissances de l'attaquant, d'identifier une partie des *logiciels* présents sur la *machine*. Pour chaque *logiciel Y* découvert, nous associons le *behaviorBlocks* '*SearchVuln_Y*' permettant, en fonction des compétences de l'attaquant, de déterminer si le *logiciel* est vulnérable ou non. Pour chaque *logicielVulnérable Y* découvert, nous associons le *behaviorBlocks* '*ExecuteVuln_Y*' permettant, en fonction des prérequis du *logicielVulnérable*, de modifier le système conformément aux effets définis. Pour chaque *machineContrôlée X*, nous associons le *behaviorBlock* '*Ping_X*' permettant de découvrir l'ensemble des *machines* allumées et en relation directe.

Une partie du scénario d'attaque de Mallory est schématisé en figure 4.12 via un diagramme de séquence. Nous présentons en annexe E une version complète de ce diagramme. Sur cette figure, les flèches pleines représentent l'exécution de *behaviorBlocks* et celles en pointillés l'association d'un nouveau *rôle* ou la modification d'un *player*. Les blocs gris clair

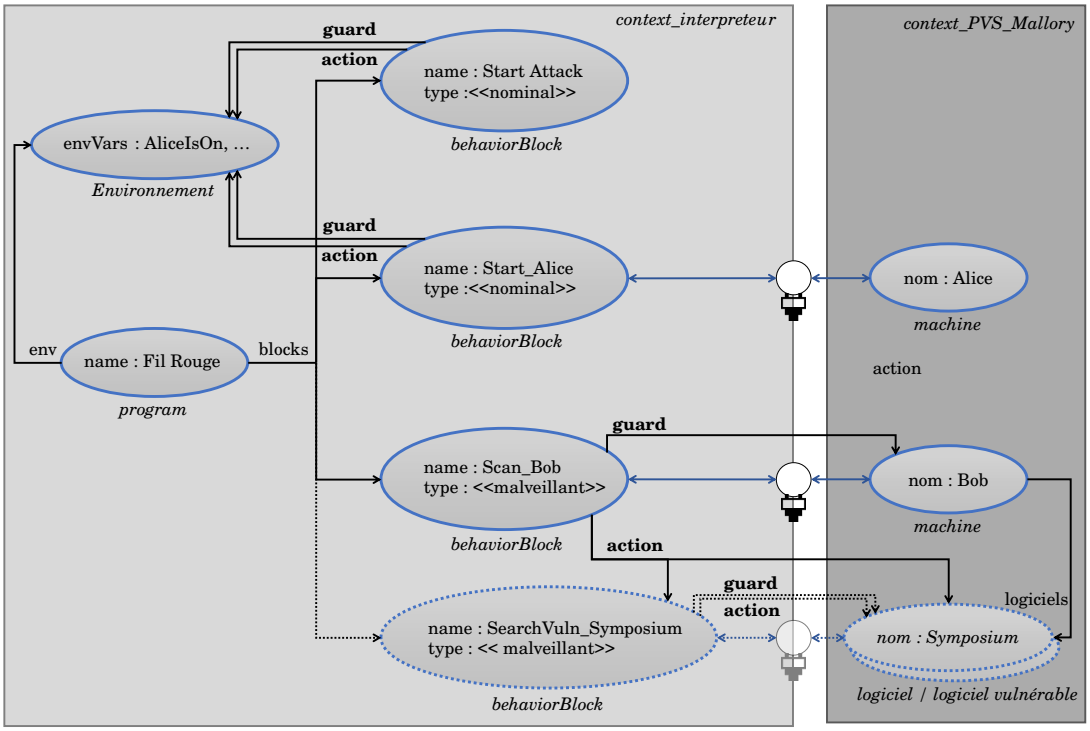


FIGURE 4.11 – Représentation schématique de l'interpréteur du système fil rouge

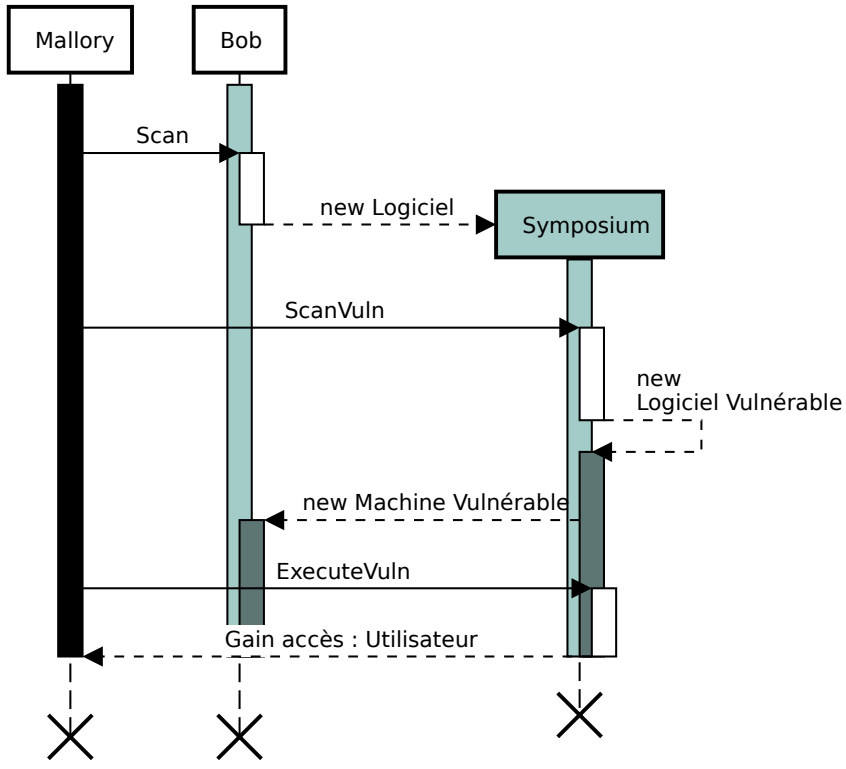


FIGURE 4.12 – Diagramme de séquence représentant une partie de l'attaque de Mallory sur le système fil rouge

symbolisent des *machines* ou des *logiciels*, les gris foncé, des *machines vulnérables* ou des *logiciels vulnérables* et les noirs, des *machines contrôlées*.

Dans ce diagramme de séquence, Mallory exécute le *behaviorBlock* 'Scan' sur 'Bob' permettant d'associer le *rôle* de *Logiciel* aux applications utilisées par Bob. Par souci de clarté, nous ne représentons ici que le *logiciel* 'Symposium'. Par la suite, l'attaquant exécute 'SearchVuln' sur Symposium. Ce *logiciel* possède une vulnérabilité exploitable par Mallory, nous lui associons donc le *rôle* de *Logiciel Vulnérable* et, par effet de bord, Bob est dorénavant représenté comme une *machine vulnérable*. Mallory exécute ensuite 'ExecuteVuln' sur Symposium. L'exploitation de la vulnérabilité présente sur ce *logiciel vulnérable* permet à Mallory d'obtenir des droits utilisateur sur Bob. Par la suite (voir annexe E), Mallory utilise ces nouveaux droits pour poursuivre son attaque.

Au cours de son attaque, Mallory est amenée à découvrir de nouveaux éléments du système. Ses découvertes correspondent à la création de nouveaux *behaviorBlocks* représentant ces éléments. Par exemple, le *behaviorBlock* 'Scan_Bob' entraîne la création de *logiciels* représentant les applications découvertes sur Bob. Cette création correspond en pratique à l'association d'un *rôle* à un éléments du système qui n'en possédait par encore. Le fait d'associé un *rôle* à un élément permet d'ajouter une représentation de cette éléments dans une vue. De plus, l'exécution d'un *behaviorBlock* peut entraîner l'association de nouveaux *rôles* à des éléments déjà représenté. Par exemple, l'exécution de 'SearchVuln_Symposium' permet d'identifier que le *logiciel* Symposium est vulnérable. L'élément en question est alors représenté de différentes manières dans une même vue, ici comme un *logiciel* et comme un *logiciel vulnérable*.

Cette exemple ici illustre l'un des aspects fondamentaux de notre approche, à savoir, l'association de nouveaux *rôle* en cours d'exécution. Cette fonctionnalité permet d'adapter dynamiquement la représentation qu'un attaquant ce fait d'un du système. Par exemple, la représentions que Mallory ce fait de Bob change au cours de son attaque pour passer d'une machine classique à un machine contrôlé.

4.4 Conclusion

Dans ce chapitre nous présentons Role4All, notre solution de modélisation de point de vue par les rôles. Nous décrivons tout d'abord notre cadre de modélisation (§4.1), puis nous démontrons en quoi Role4All permet de modéliser des points de vue d'attaquant (§4.2) et pour finir nous analysons, via Role4All, le comportement d'un attaquant à partir d'un point de vue d'interprétation (§4.3). La figure 4.13 présente une carte heuristique regroupant les points clef de notre approches.

Dans la section 4.1.1 nous présentons le métamodèle de Role4All reposant sur les concepts de *rôle*, de *player*, d'*adapter*, de *playRelation* et de *context*. Un *rôle* est la représentation d'un *player* dans un certain *context*, la relation entre ces deux éléments est modélisée par une *playRelation* et un *adapter*. Role4All propose une adaptation bidirectionnelle permettant de modifier l'*adapter* d'une *playRelation* en cours d'exécution.

Par la suite (§4.1.2), nous formalisons notre approche. Tout d'abord, à l'aide de 19 des 27 caractéristiques de rôles proposé par Kühn [100]. Cette formalisation nous a permit de comparé le métamodèle de Role4All par rapport à l'état de l'art et de justifier nos choix de modélisation. Nous formalisons ensuite mathématiquement les notions de *rôle*, de *player* et de *context* utilisés dans Role4All à l'aide de huit axiomes et de cinq propositions optionnelles. Les axiomes assurent que les modèles de *rôles* sont correctement formés et les propositions permettent de limiter l'explosion combinatoire liée aux analyses formelles. Cette formalisation suit les préconisations identifiées dans notre état de l'art et nous permet de rapprocher formellement notre formalisation des *rôles* de la formalisation de point de vue.

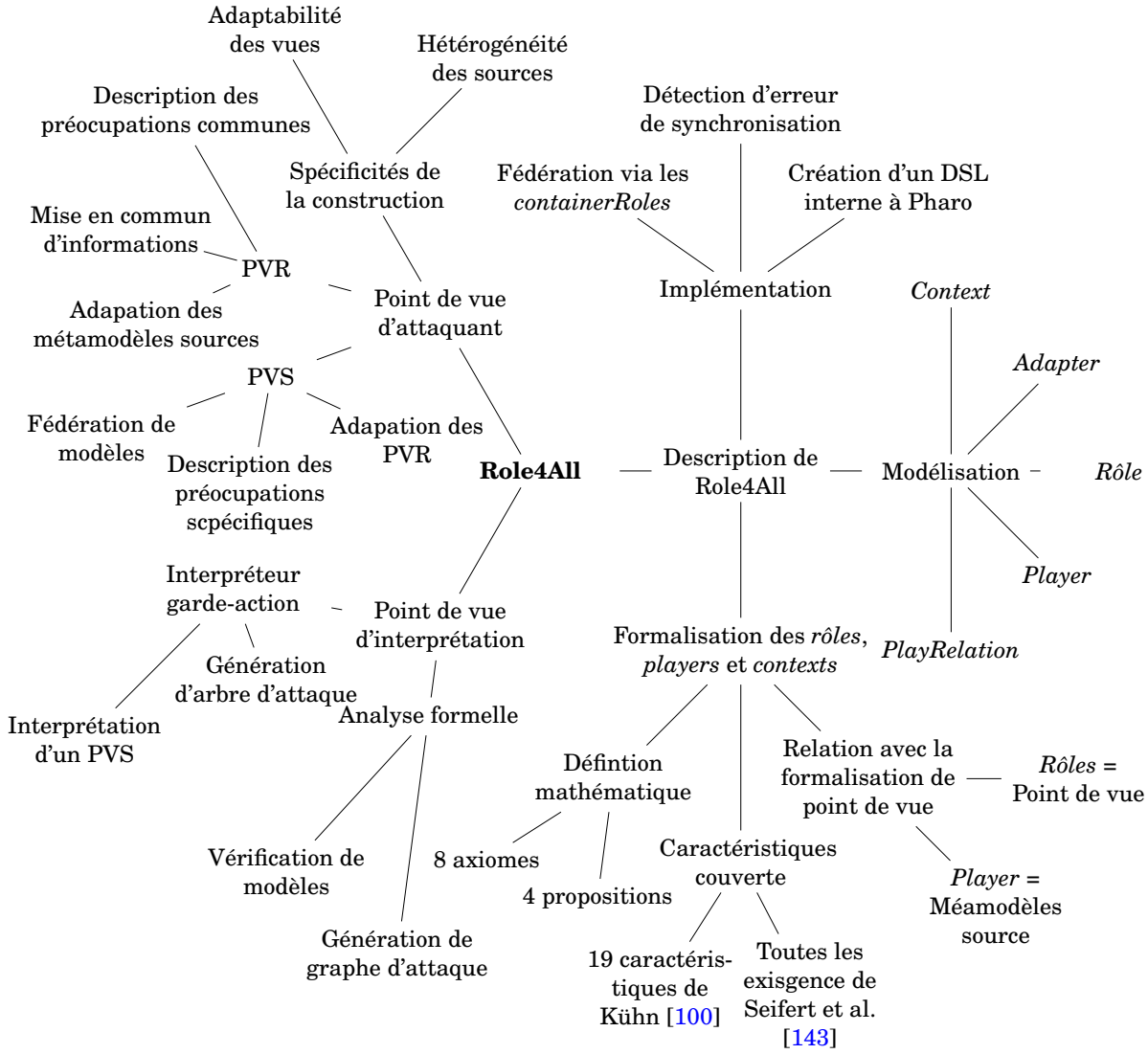


FIGURE 4.13 – Carte heuristique (*mind map*) de Role4All

Nous terminons cette section en présentant notre implémentation de Role4All comme un DSL interne à Pharo. Cette solution d'implémentation offre un prototypage rapide et nous permet d'utiliser l'ensemble des fonctionnalités de débogage incluses dans l'outil de développement associé à Pharo. La création d'un DSL interne nous a permis de nous focaliser sur le développement des concepts principaux de notre approche et d'utiliser les concepts propres à Pharo pour parfaire notre prototype.

Dans la section 4.2 nous démontrons l'efficacité de Role4All pour la description de point de vue et en particulier celui d'un attaquant. Nous présentons également comment Role4All permet de fédérer des informations prévenant de sources hétérogène lors de la construction des PVR et PVS. Pour ce faire, nous définissons une relation de contenance entre des *rôles*, un *containerRole* contenant des *containedRoles*. Cette relation permet de créer des *roleInstances* regroupant des informations provenant de sources hétérogènes et permet également de fédérer des informations contenues dans différents *playerInstances*. La fédération d'informations, implémentée dans Role4All, repose sur la non-matérialisation des données dans les *rôles*. Seuls les *players* contiennent des données, les *rôles* ne font que les référencer. Role4All permet de détecter les problèmes de cohérence de données et de lever des erreurs indiquant la cible, la date d'envoi et l'expéditeur des messages en défaut. À

l'heure actuelle, nous n'avons pas implémenté de solutions permettant de gérer automatiquement ce type de problèmes tels que des classificateurs par vote rigide ou des approches par empilement [61].

Dans la section 4.3 nous construisons un point de vue d'interprétation dédié à l'analyse et l'exécution du comportement d'un attaquant. Nous construisons un ensemble de gardes et d'actions correspondant respectivement à des évaluations et des modifications de l'espace d'état. Dans notre approche, l'espace d'état est construit à partir de PVS représentants ce qu'un attaquant spécifique perçoit d'un système. Role4All permet alors, via un ensemble de gardes et d'actions, d'exécuter un comportement à travers le point de vue d'un attaquant.

Finalement, Role4All est une solution de modélisation de point de vue d'attaquant utilisant la notion de *rôle* permettant la fédération de données hétérogènes et l'exécution de comportements au travers de différents points de vue.

Chapitre 5

Cas d'étude et validation de l'approche

Sommaire

Dans ce chapitre, nous illustrons et validons notre approche sur un cas d'étude proposé par DIATEAM, une société française indépendante de recherche et développement logiciel, spécialisée dans la sécurité informatique et les systèmes d'information innovants. Dans un premier temps, nous décrivons notre cas d'étude à l'aide de modèles issus d'outils hétérogènes, puis nous modélisons le point de vue d'un attaquant et finalement nous construisons un point de vue d'interprétation basé sur la perception que l'attaquant a du système.

5.1 Présentation du système et des outils utilisés

Dans cette section, nous présentons notre cas d'étude puis nous le modélisons via divers outils.

5.1.1 Description du système

Le cas d'étude utilisé dans ce chapitre est un réseau d'entreprises composé d'un serveur web, d'un Active Directory et d'utilisateurs. Il se compose de différents sous-réseaux protégés par un pare-feu. Pour ce cas d'étude nous savons, à priori, que le système est vulnérable mais nous ignorons quels sont effectivement ses vulnérabilités.

Afin de mettre en place notre cas d'étude, nous utilisons Hynesim [5], une plateforme distribuée de simulation de systèmes d'information permettant de simuler des réseaux complexes. Cette plateforme permet également de relier un réseau virtuel à des équipements réels et d'exécuter des actions à distance sur des agents virtualisés.

Le réseau virtuel utilisé comme cas d'étude dans ce chapitre est présenté en figure 5.1(a).

Ce système comporte deux attaquants, l'un utilisé pour effectuer une attaque et l'autre comme observateur. Il comporte également un utilisateur, un serveur web, un Active Directory (AD) et un pare-feu. Les interconnexions entre ces différents éléments sont réalisées à l'aide de routeurs. Les outils (outil de surveillance réseau et outil SCADA) présents sur le système permettent de faire des observations sur le système, ils ne seront pas utilisés durant l'attaque.

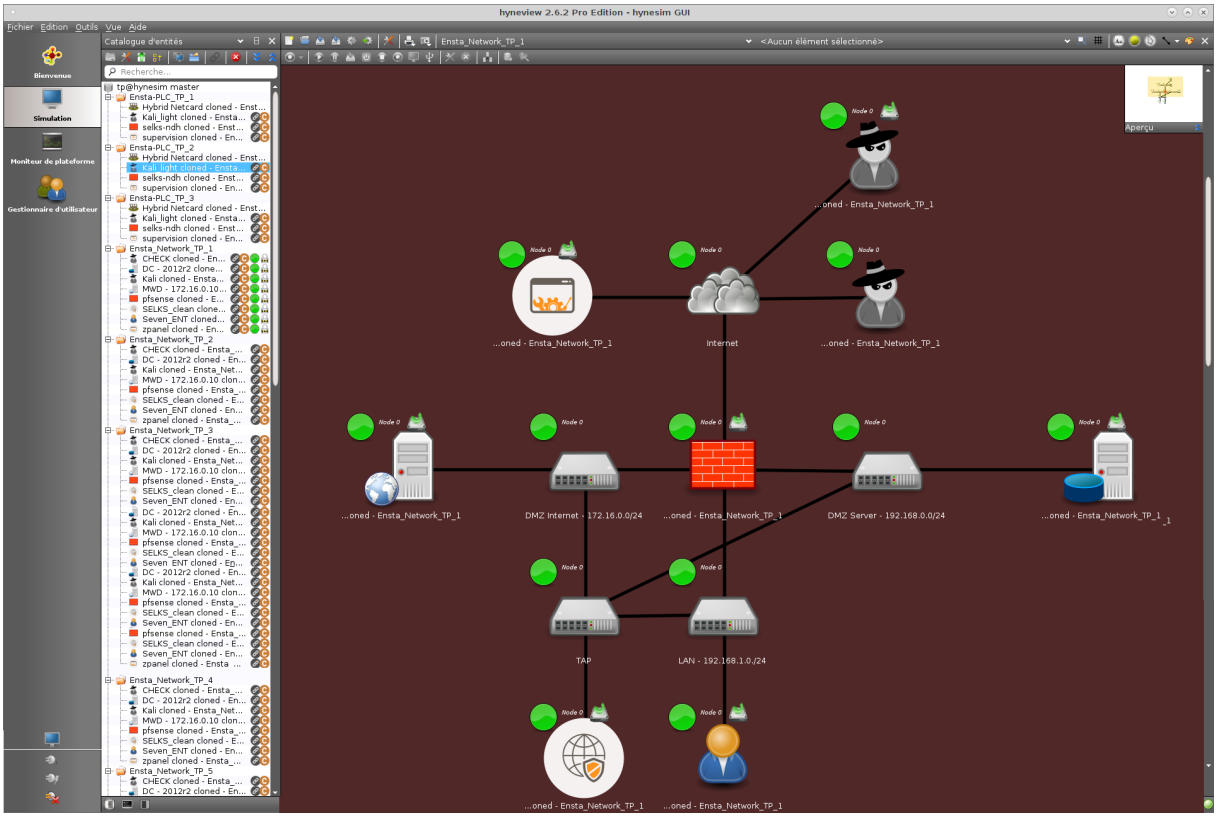
5.1.2 Modélisation du système

Dans cette section, nous présentons les différentes solutions utilisées afin de modéliser le système présenté précédemment (§5.1.1).

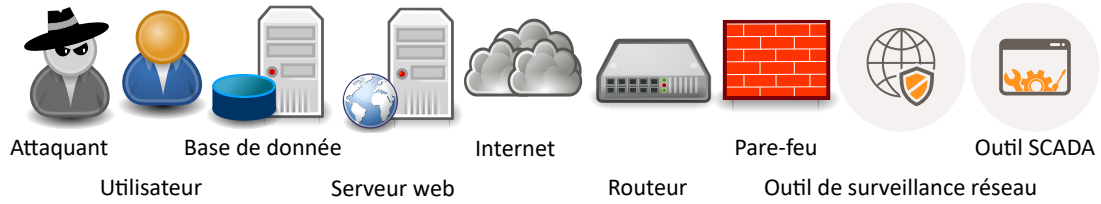
5.1.2.1 Topologie du système

Afin de décrire la topologie du système, nous utilisons Pimca [157], un outil de modélisation développé par une entité de cybersécurité du ministère des Armées, permettant de représenter et d'analyser un système à plusieurs niveaux d'abstraction. Le DSL Pimca est développé par la Direction générale de l'armement (DGA), il est utilisé pour faciliter les analyses de la surface d'attaque, d'impact d'attaques et de chemins d'attaques. Pimca dispose d'un framework de modélisation de sécurité open-source [157] développé par l'ENSTA Bretagne. Dans notre cas d'étude, nous utilisons Pimca pour représenter la topologie du système.

Une partie du métamodèle de Pimca est présenté en figure 5.2, un modèle Pimca est composé d'un ensemble de *machineries*, de *ressources* et de *relations* :



(a) Système simulé dans Hynesim



(b) Liste des différents types d'éléments du système

FIGURE 5.1 – Système utilisé pour notre cas d'étude

— Les *machineries* sont des éléments actifs du système capables d'interagir avec d'autres éléments via des *relations*, par exemple un ordinateur ou un serveur peut être représenté par une *machinerie* connectée à un réseau. Pimca permet de définir quatre spécialisations du concept de *machinerie*, à savoir :
  — Performer : il représente un être humain qui interagit avec le système tel un utilisateur, un administrateur système ou encore un attaquant.
  — Interface : il matérialise un pont conceptuel entre les dimensions physiques et virtuelles, tel un clavier ou un écran.
  — Checkpoint : il représente un élément vérifiant un *passeport* spécifique avant d'accorder le passage. Par exemple, une porte de sécurité nécessitant un badge ou un système d'authentification attendant un mot de passe.
  — Network : il représente les canaux d'échange entre différentes *machineries* tels qu'un commutateur réseau.
— Les *ressources* sont des éléments passifs du système, contrairement aux *machineries*, elles n'ont pas de comportement propre. Elles sont les cibles des actions effectuées par

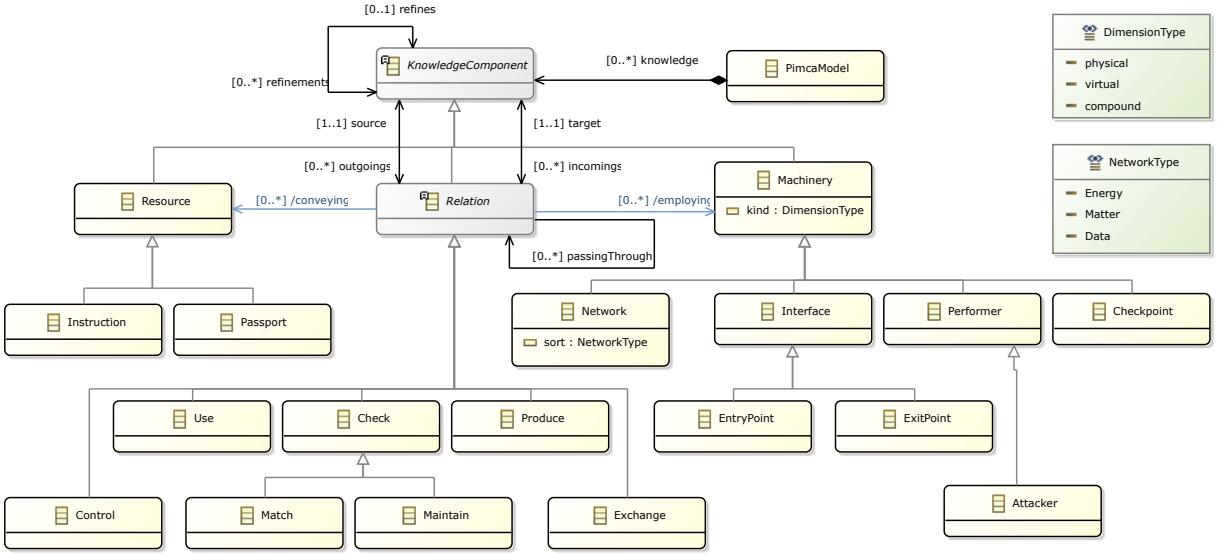


FIGURE 5.2 – Métamodèle de Pimca

les *machineries*. Par exemple, une *ressource* peut être produite par une machinerie et consommée par une autre. Pimca définit deux types de *ressources*, les *instructions* et les *passeports*. Une *instruction* représente une commande ou un ordre utilisé par une *machinerie* et un *passeport* est une clef permettant de franchir un *checkpoint* spécifique.

— Les *relations* modélisent les interactions entre les constituants d'un système. Pimca prend en compte six types de relations :

  — *Exchange* : une relation bidirectionnelle entre deux *machines* utilisées pour échanger des informations, par exemple, via un réseau informatique.

  — *Control* : une relation symbolisant l'influence d'un élément sur un autre, par exemple, un *performer* a une relation de *control* sur sa *machine*.

  — *Use* : une relation caractérisant la dépendance d'un élément à un autre, par exemple, l'authentification d'un utilisateur sur un réseau est dépendant d'un serveur d'authentification.

  — *Produce* : une relation modélisant la création d'une *ressource* ou d'un bien, par exemple, un serveur d'authentification produit une liste d'utilisateurs.

  — *Maintain* : une relation marquant la responsabilité d'un acteur sur un élément du système, par exemple, un administrateur système est responsable du serveur d'authentification.

  — *Match* : une relation associant un *passeport* à un *checkpoint*, par exemple, les identifiants d'un acteur sont associés à un compte utilisateur sur sa machine.

La figure 5.3 présente un modèle Pimca représentant notre cas d'étude. Sur cette figure le serveur web, l'AD et le PC utilisateur sont représentés par des *machineries* communiquant entre elles via des *networks* et un *checkpoint* symbolisant un pare-feu. Le serveur web produit une *ressource*, à savoir un site web. Il est utilisé légitimement par l'attaquant lorsque ce dernier consulte le site internet de l'entreprise. L'AD produit une *ressource* nommé "Liste d'utilisateurs" qui est utilisée par différents *checkpoints* afin d'authentifier des utilisateurs.

Le système comporte trois utilisateurs modélisés en tant que *performer* : un administrateur web qui contrôle et maintient le serveur web, un utilisateur (Sylvain) qui contrôle un

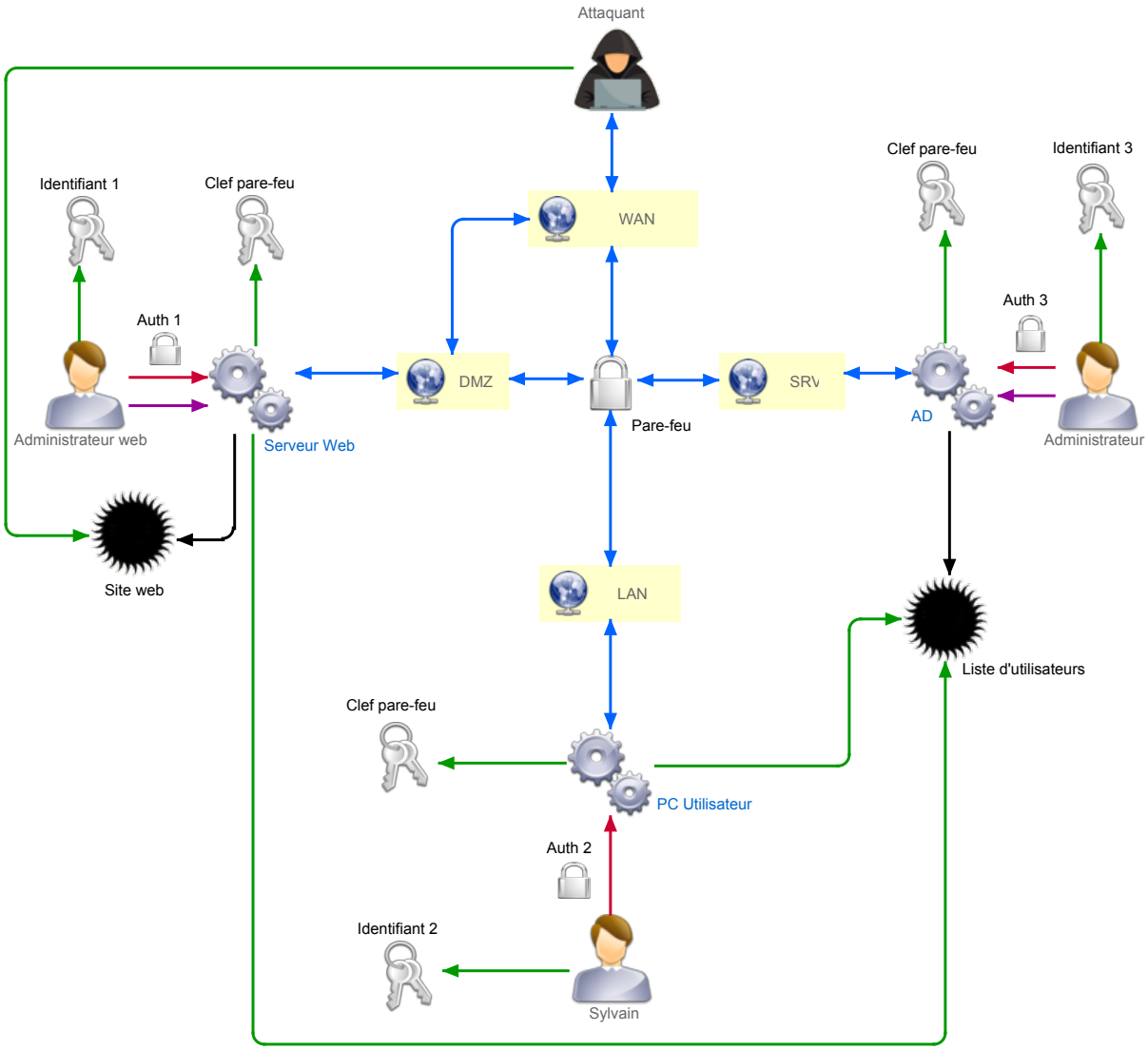


FIGURE 5.3 – Modèle Pimca représentant la topologie du système

PC et un administrateur système qui contrôle et maintient l'AD. Chaque *performer* accède à sa machine via une authentification représenté par un couple *passeport* - *checkpoint*.

L'attaquant communique avec le système au travers de différents *networks*. Il ne peut interagir qu'avec le serveur web car il ne possède pas le *passeport* nécessaire au franchissement du pare-feu protégeant le système. Afin de mener une attaque sur le système, l'attaquant devra alors trouver un moyen de franchir ou de contourner ce *checkpoint*.

La *ressource* "liste d'utilisateurs" est utilisée par différents éléments afin d'authentifier des utilisateurs, elle est donc essentielle au bon fonctionnement du système. Si l'attaquant arrive, au cours de son attaque, à nuire à cette ressource, il pourra alors paralyser l'ensemble du système. La *ressource* "site web" représente la vitrine de l'entreprise, c'est le seul élément accessible depuis l'extérieur du système. Si l'attaquant arrive, au cours de son attaque, à contrôler la production de cette ressource, il pourra alors nuire à l'image de l'entreprise. Les différentes *ressources* du système sont donc des cibles privilégiées lors d'une attaque.

Nous nous focaliserons dans la section 5.3 sur la recherche d'attaques permettant de prendre le contrôle du serveur web, de franchir le pare-feu et/ou de nuire à la *ressource* liste d'utilisateurs.

5.1.2.2 Acteurs du système

Pour notre cas d'étude nous modélisons les acteurs du système grâce à un fichier CSV représentant la liste des utilisateurs enregistrés sur l'AD, à savoir :

```
Nom,Type,Nom complet,Adresse de messagerie
target,Utilisateur,target,target@training.com
user,Utilisateur,user,
admin,Administrateur,admin,
```

Le système comporte trois acteurs nommés *target*, *user* et *admin*. Les deux premiers possèdent un accès limité respectivement au serveur web et au PC utilisateur, le dernier possède un accès administrateur à l'AD. Au cours de l'attaque, la liste des utilisateurs peut être modifiée, par exemple via l'ajout d'un nouvel utilisateur ou la modification de droits existants.

5.1.2.3 Configuration des éléments

Nous modélisons ici uniquement la configuration du PC utilisateur, du serveur web, de l'AD et des éléments de réseau (internet, routeurs et pare-feu). La configuration des machines utilisées par l'attaquant n'est pas pertinente dans ce cas d'étude.

Les machines composant le système utilisent 3 systèmes d'exploitation différents, il s'agit de Windows 7 version 6.1.7601 service Pack 1 pour la machine utilisateur, de Windows server 2012 R2 pour l'AD et de Ubuntu 14.04.5 LTS dit Ubuntu trusty pour le serveur web. La configuration des éléments de notre cas d'étude est contenue dans des fichiers textuels listant les applications / paquets installés sur les différentes machines. La structure de ces fichiers varie en fonction du système d'exploitation et de la technique utilisée pour extraire les informations. Par exemple, pour le serveur web, nous utilisons la commande *lsb_release -a* afin d'obtenir des informations sur le système d'exploitation utilisé et la commande *apt list –installed* afin de lister l'ensemble des paquets installés. Les fichiers textuels générés par ces différentes commandes sont :

— *lsb_release -a* :

```
Distributor ID: Ubuntu
Description:    Ubuntu 14.04.5 LTS
Release:        14.04
Codename:       trusty
```

— *apt list –installed* :

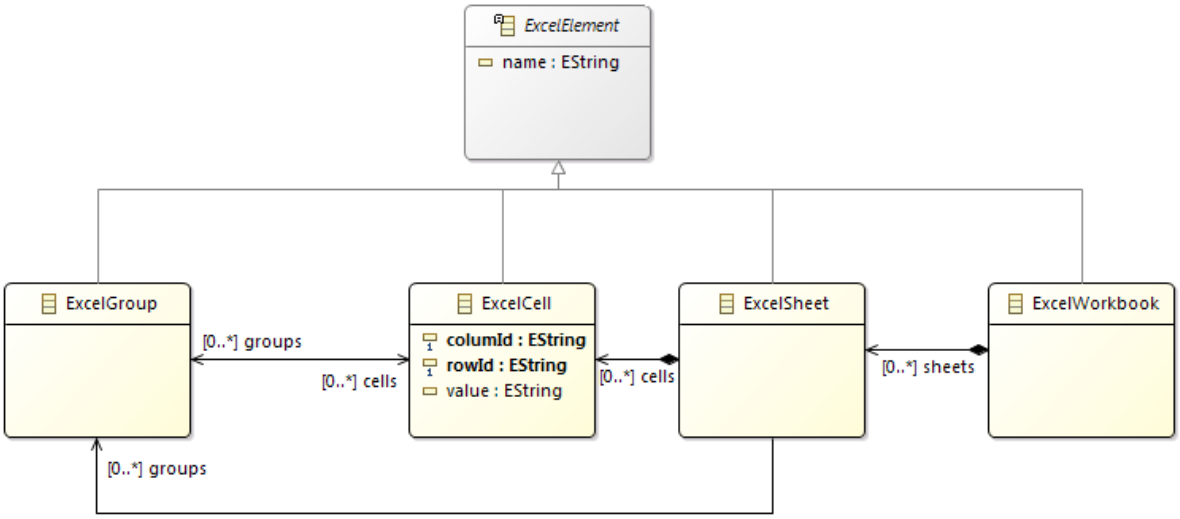
```
accountsservice/now 0.6.35-0ubuntu7.2 amd64

...

wordpress/unknown,unknown,now 3.8.2+dfsg-1ubuntu0.1 all
wordpress-l10n/unknown,unknown,now 3.8.2+dfsg-1ubuntu0.1 all
wordpress-theme-twentyfourteen/unknown,unknown,now 3.8.2+dfsg-1
    ubuntu0.1 all
```

```
603 wordpress-theme-twentytwelve/unknown,unknown,now 3.8.2+dfsg-1
    ubuntu0.1 all
604 wpasupplicant/unknown,unknown,now 2.1-0ubuntu1.4 amd64
605 xauth/trusty,now 1:1.0.7-1ubuntu1 amd64
606 xkb-data/trusty,now 2.10.1-1ubuntu1 all
607 xml-core/trusty,now 0.13+nmu2 all
608 xz-utils/trusty,now 5.1.1alpha+20120614-2ubuntu2 amd64
609 zlib1g/trusty,now 1:1.2.8.dfsg-1ubuntu1 amd64
```

Dans notre exemple nous modélisons quatre commutateurs, à savoir : Internet, DMZ Serveur, DMZ Internet et LAN. Un commutateur possède un comportement complexe qui peut varier selon le constructeur, dans cet exemple, nous le représentons via une liste associant à chaque élément une adresse IP. Cette liste est décrite sous forme d'un tableau Excel [1] présenté en figure 5.4(b). Nous utilisons dans cette section une simplification du métamodèle d'Excel présenté en figure 5.4(a) définissant uniquement les concepts de l'outil nécessaire pour couvrir notre cas d'étude. Le pare-feu du système est configuré afin de bloquer tous



(a) Métamodèle simplifié de Excel

	A	B	C
1	**Reseau**	**Nom**	**Ip**
2	SRV	Active Directory	192.168.0.2
3	SVR	Firewall	192.168.0.1
4	DMZ	Serveur Web	172.16.0.10
5	DMZ	Firewall	172.16.0.1
6	LAN	Client	192.168.1.30
7	LAN	Firewall	192.168.1.1
8	WAN	Kali1	8.8.10.10
9	WAN	Kali2	8.8.12.12
10	WAN	Firewall	8.16.32.64

(b) Liste des adresse IP sur les différent réseaux

FIGURE 5.4 – Représentation d'adresse IP dans Excel

1. Microsoft Excel est un logiciel tableur de la suite bureautique Microsoft Office développé et distribué par l'éditeur Microsoft.

les échanges provenant du réseau WAN (internet) sauf ceux à destination du réseau DMZ. Pour les autres réseaux, aucune règle spécifique n'est définie. Par conséquent, le PC utilisateur, l'AD et le serveur web peuvent communiquer sans limitation et Kali1 et Kali2 ne peuvent communiquer qu'avec le serveur Web sur les ports 80 (HTTP), 143 (IMAP) et 25 (SMTP). La figure 5.5 présente la configuration du pare-feu pour le réseau WAN. Nous choi-

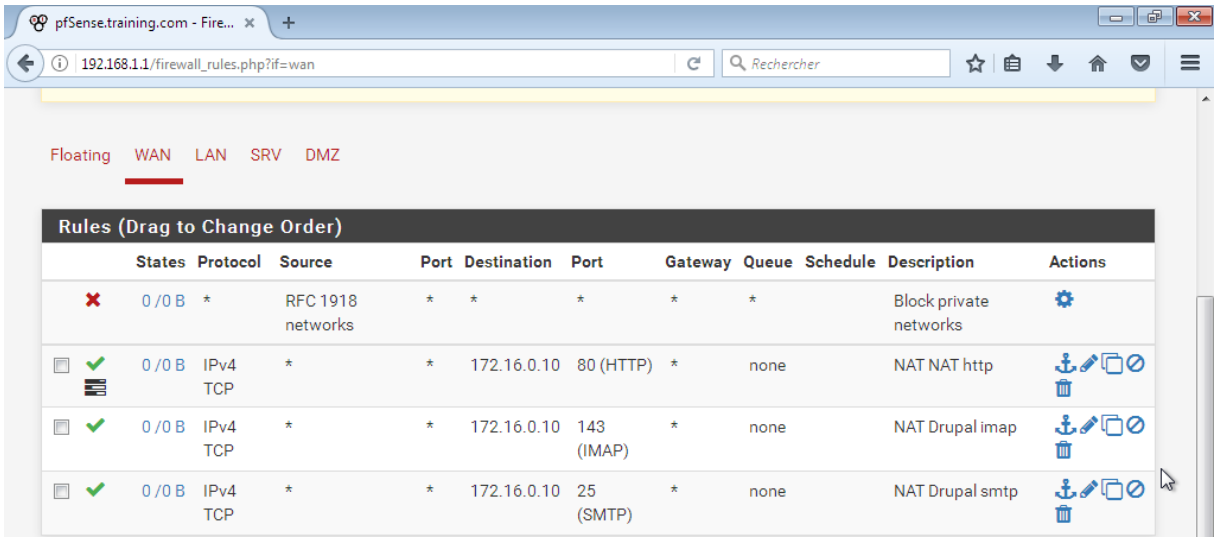


FIGURE 5.5 – Configuration du pare-feu pour le réseau WAN

sissons pour cet exemple de modéliser les règles de routage dans un fichier Excel reprenant les informations du tableau présenté en figure 5.5.

Dans la suite de ce chapitre nous définirons différents *adapter* permettant de traduire les informations contenues dans un fichier Excel ou dans un document texte pour les Points de Vue de Référence (PVR).

5.1.2.4 Vulnérabilités

Nous présentons dans la section 2.1.2 différents standards utilisés dans le domaine de la cybersécurité, parmi lesquels figurent NVD, CVE et CVSS. Le premier est une base de données de vulnérabilités répertoriant plus de 145 000 vulnérabilités nommées d'après la nomenclature CVE. Le second est un dictionnaire énumérant les vulnérabilités connues du public et proposant une nomenclature de nommage. Le dernier propose une métrique permettant de noter entre 0 et 10 la dangerosité d'une vulnérabilité.

Nous utiliserons dans ce cas d'étude NVD afin d'identifier les vulnérabilités du système. Ces dernières ne sont pas connues à priori et seront déterminées durant l'attaque en comparant les configurations détectées par l'attaquant avec la base de donnée de NVD. Cette dernière est disponible sous forme de fichier JSON rassemblant l'ensemble des vulnérabilités d'une année. Par exemple, pour l'année 2014 un fichier JSON de 1 024 188 lignes en libre accès sur [6] rassemble 8 842 vulnérabilités. Le code ci-dessous décrit une de ces vulnérabilités :

```
147683 {
147684   "cve" : {
147685     "data_type" : "CVE",
147686     "data_format" : "MITRE",
147687     "data_version" : "4.0",
147688     "CVE_data_meta" : {
```

```
      "ID" : "CVE-2014-10021",
      "ASSIGNER" : "cve@mitre.org"
    },

    ...

    "description" : {
      "description_data" : [ {
        "lang" : "en",
        "value" : "Unrestricted file upload vulnerability in
           UploadHandler.php in the WP Symposium plugin 14.11 for
           WordPress allows remote attackers to execute arbitrary
           code by uploading a file with an executable extension,
           then accessing it via a direct request to the file in
           server/php/."

        ...

        "cpe23Uri" : "cpe:2.3:a:wpsymposiumpro:wp_symposium:14.11:*
           :*:*:*:wordpress:*:*"
      } ]
    } ]
  },
  "impact" : {
    "baseMetricV2" : {
      "cvssV2" : {
        "version" : "2.0",
        "vectorString" : "AV:N/AC:L/Au:N/C:P/I:P/A:P",
        "accessVector" : "NETWORK",
        "accessComplexity" : "LOW",
        "authentication" : "NONE",
        "confidentialityImpact" : "PARTIAL",
        "integrityImpact" : "PARTIAL",
        "availabilityImpact" : "PARTIAL",
        "baseScore" : 7.5
      },
      "severity" : "HIGH",
      "exploitabilityScore" : 10.0,
      "impactScore" : 6.4,
      "obtainAllPrivilege" : false,
      "obtainUserPrivilege" : false,
      "obtainOtherPrivilege" : false,
      "userInteractionRequired" : false

    ...
```


Ce code JSON décrit la vulnérabilité CVE-2014-10021 affectant l'extension Symposium présent sur le logiciel Wordpress. Pour être exécutée, elle demande un accès réseau au système et affecte partiellement la confidentialité, l'intégrité et la disponibilité de l'actif vulnérable.

Afin de déterminer, durant l'attaque, les vulnérabilités du système, nous comparerons la configuration d'un élément au *cpe23Uri* décrivant l'actif vulnérable (ligne 147 728). Cette comparaison sera réalisée à l'aide de *rôles* et d'*adapters*.

5.2 Modélisation de la vue de l'attaquant sur le système

Dans la section précédente (§5.1), nous décrivons et modélisons un cas d'étude à l'aide de différents outils tel que Pimca, Excel, des fichiers de configuration et des données non structurées. Dans cette section nous adaptons les informations provenant de ces diverses sources pour nos PVR, puis nous construisons un PVS permettant de représenter la perception d'un attaquant sur le système.

5.2.1 Adaptation des métamodèles sources pour les PVR

Dans les chapitres 3 et 4, nous présentons et modélisons des PVR représentant respectivement le système, les vulnérabilités, l'attaquant et son comportement. Les modèles sources présentés dans la section 5.1 contiennent des informations relatives au système et aux vulnérabilités, nous présentons ici les différents *adapters* permettant de mettre en forme ces informations pour les PVR associés.

5.2.1.1 Adaptation pour le PVR système

Les informations nécessaires pour la construction du PVR système sont contenues dans le modèle Pimca, des tableaux Excel et des données non structurées.

Adaptation de Pimca

Le modèle Pimca présenté dans la section 5.1.2 modélise le système sous forme de *machinery*, de *resource* et de six types de *relation*. Les liens entre les concepts du métamodèle

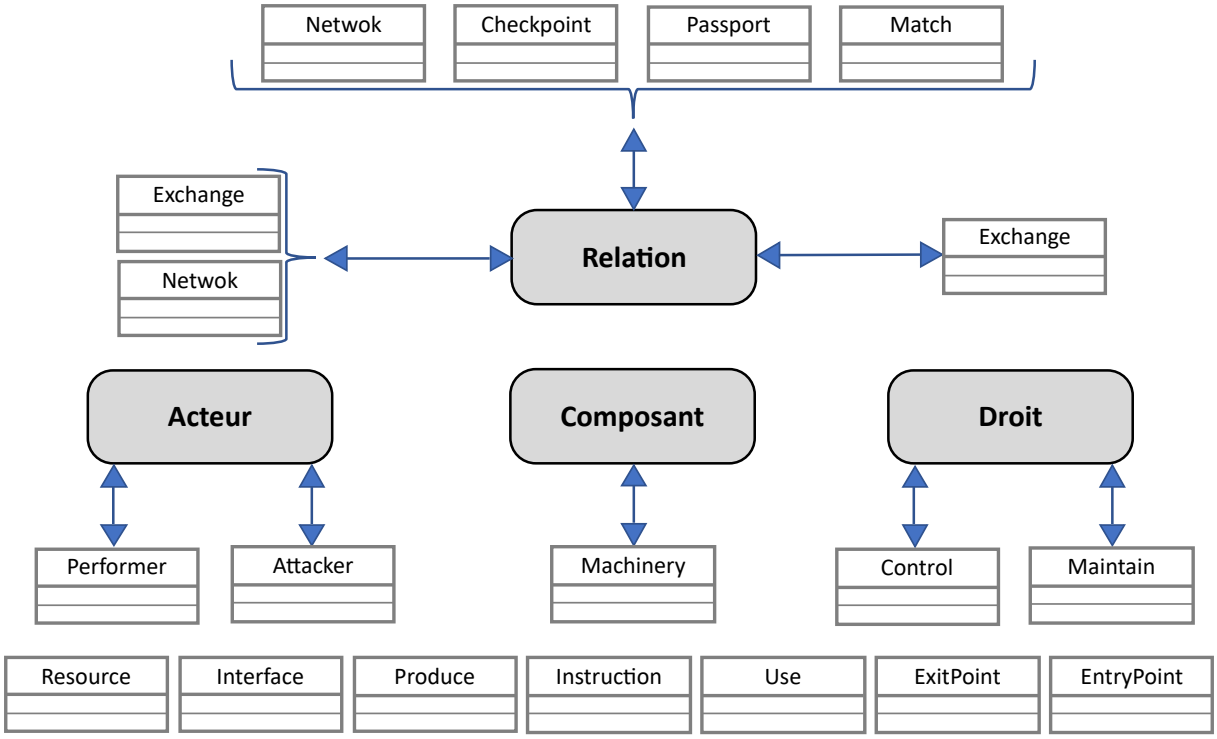


FIGURE 5.6 – Relation entre les concepts de Pimca et les *rôles* du PVR système

de Pimca et les *rôles* composants le PVR système sont illustrés sur la figure 5.6. Dans cette figure, chaque relation bidirectionnelle entre l'un des concepts de Pimca et un *rôle* représente une *playRelation* et un *adapter*. Par exemple, la connexion entre l'attaquant et le serveur web modélisée par un ensemble d'*exchanges* et de *networks* est représentée par une instance du *rôle Relation*. Les *performers* et les *attackers* jouent le *rôle* d'*Acteur*, les *machineries* celui de *Composant* et les relations *control* et *maintain* celui de *Droit*.

Adaptation d'Excel

Dans notre cas d'étude, Excel est utilisé pour associer des adresses IP aux éléments du système en fonction du réseau auquel ils appartiennent, mais aussi pour représenter les règles de routages du pare-feu.

Pour les configurations IP, une ligne du tableur contient le nom du réseau, le nom de l'élément et une adresse IP. Le code présenté ci-dessous correspond à l'*adapter* entre le concept *ExcelGroup* représentant une ligne et le *rôle Composant*.

```
Adapter subclass: #AdapterExcelGroupeToRoleComposant
  instanceVariableNames: ''
  classVariableNames: ''

    getComposantName: obj1
      |name|
      name := obj1 cells at:2.
      ↑name.

    setComposantName: newName player: obj1
      obj1 cells at:2 put:newName.
      ↑obj1 cells at:2.

    getComposantIP: obj1
      |ip|
      ip := obj1 cells at:3.
      ↑ip.

    setComposantIP: newIP player: obj1
      obj1 cells at:3 put:newIP.
      ↑obj1 cells at:3.
```

Les accesseurs et mutateurs définis ci-dessus sont utilisés par le *rôle Composant* afin de définir le nom d'un *composant* ainsi qu'une partie de sa configuration, ici son adresse IP.

Nous définissons aussi deux autres *adapters* pour le même concept, l'un pour le *rôle Accès* et l'autre pour le *rôle Composant*. Le premier permet d'identifier les accès réseau des différentes machines du système et le second, dont le code est présenté ci-dessous, permet d'extraire une adresse IP des informations de configuration du pare-feu.

```
Adapter subclass: #AdapterExcelGroupeToRoleComposant2
instanceVariableNames: ''
classVariableNames: ''

  getComposantIP: obj1
    |ip|
    ip := obj1 cells at:7.
    ↑ip.

  setComposantIP: newIP player: obj1
    obj1 cells at:7 put:newIP.
    ↑obj1 cells at:7.
```

Nous définissons ici plusieurs *adapters* entre le même *player* (*ExcelGroup*) et le même *rôle* (*Composant*). Le choix de l'*adapter* utilisé dépend dans notre exemple du type de fichier Excel référencé. De plus, l'information "nom d'un *composant*" est référencée dans deux

sources distinctes, à savoir un fichier Excel et un modèle Pimca. De même, des adresses IP sont modélisées dans deux fichiers Excel différents. Nous verrons dans la suite de ce chapitre comment ces diverses sources d'information sont fédérées dans un même *roleInstance*.

Adaptation de données non structurées

La configuration des éléments de notre cas d'étude est contenue dans des fichiers textuels listant les applications / paquets installés sur les différentes machines du système. Nous présentons dans la section 5.1.2 deux fichiers décrivant le système d'exploitation et les paquets installés sur le serveur web. Nous définissons deux *adapters* permettant de mettre en forme les informations contenues dans ces fichiers pour le PVR système. Le premier, *AdapterReleaseToProduit* extrait le nom et la version du système d'exploitation utilisé, ici "Ubuntu 14.04". Comme l'illustre la figure 5.7 cet *adapter* est utilisé par les instances du *rôle Produit* afin de définir un *produit* de type *Système d'exploitation*. Nous définissons de manière analogue un *adapter*, *AdapterPackagesToConfiguration*, permettant de définir un ensemble de *produits* de type *Application* à partir du fichier textuel listant les paquets installés sur une machine.

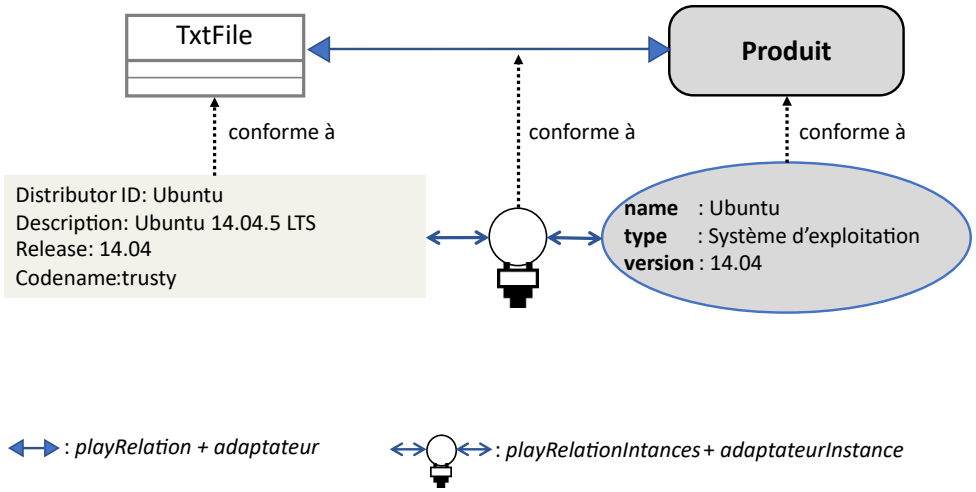


FIGURE 5.7 – Adaptation de fichier textuel décrivant un système d'exploitation

Pour notre cas d'étude, les acteurs et leurs droits sur les éléments du système sont décrits dans un fichier CVS. Une analyse textuelle nous permet d'identifier simplement le nom de l'acteur ainsi que ses types de droits. Les *adapters* entre un fichier CVS et les *rôles Acteur* et *Droit* consistent en une analyse textuelle identifiant dans un fichier CVS la valeur du "Nom" et du "Type" pour chaque ligne.

5.2.1.2 Adaptation pour le PVR vulnérabilité

Dans notre cas d'étude, nous utilisons NVD comme la base de données de vulnérabilité, les vulnérabilités référencées par NVD sont contenues dans un ensemble de fichiers JSON respectant tous le même schéma, présenté en annexe F. Comme illustré en figure 5.8 nous extrayons, via divers *adapters*, des informations permettant de créer des *roleInstances* conformes au *rôle Vulnérabilité*.

Nous définissons des *adapters* entre le concept de fichier JSON et les *rôles* de *Vulnérabilité*, d'*Impact*, d'*Actif Vulnérable* et de *Prérequis*. Ces *adapters* permettent d'identifier le nom, les prérequis, les impacts et les actifs vulnérables d'une vulnérabilité.

Un actif vulnérable est décrit sous forme d'un identifiant CPE tel que : "*cpe :2.3 :a :wp-symposiumpro :wp_symposium :14.11 :* :* :* :* :wordpress :* :**". Dans cet exemple il s'agit

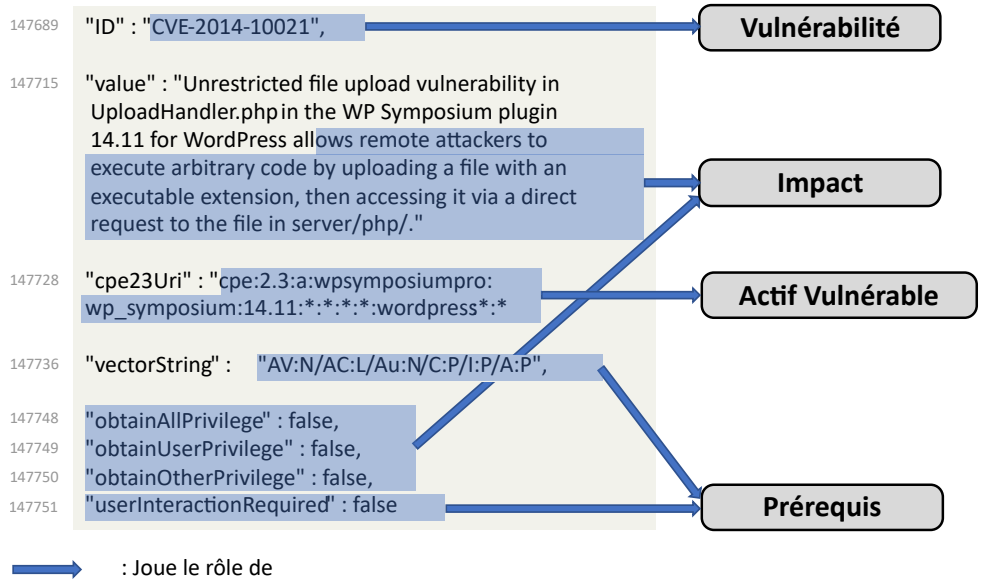


FIGURE 5.8 – Lien de "Joue le rôle de" entre la description JSON d'une vulnérabilité et les *rôles* du PVR vulnérabilité

d'une extension de *wordpress* nommé *wp_symposium* en version *14.11* et vendu par *wpsymposiumpro*. Un *adapter* permet de récupérer le nom et la version de l'application afin de construire une instance du *rôle Actif Vulnérable*. Pour définir les impacts d'une vulnérabilité, un *adapter* analyse la "*value*" de la "*description*" d'une vulnérabilité afin d'identifier des mots clefs tels que "execute arbitrary code". Cela permet de définir la *catégorie* de l'impact, dans notre exemple la chaine de caractère "execute arbitrary code" est caractéristique d'un impact de *catégorie Exécution de code*. De même, d'autre types d'analyse textuelle définissent les prérequis et le nom de la vulnérabilité.

5.2.1.3 Regroupement et fédération de données pour les différents PVR

Le PVR vulnérabilité est construit grâce à un unique *player* regroupant l'ensemble des informations utiles à la construction du modèle de *rôle*. Dans ces conditions, chaque *roleInstance* contient des informations ne provenant que d'un unique *player*. Les solutions de fédération de sources de données hétérogènes présentées dans la section 4.2.2 ne sont donc pas nécessaires pour la construction de ce point de vue.

Au contraire, le PVR système est construit avec le concours de quatre *players* contenant des informations complémentaires et redondantes. Comme présenté dans le chapitre 4, nous utilisons la relation de contenance entre *rôles* afin de définir un *containerRole* regroupant l'ensemble des informations complémentaires et responsable de la fédération des données redondantes.

Le modèle de *rôle* représentant le PVR système est présenté en figure 5.9. Sur cet exemple, une instance du *rôle Composant* regroupe des informations provenant de trois sources distinctes, à savoir un modèle Pimca et deux tableaux Excel distincts.

Nous illustrons en figure 5.10 la construction d'un *composant* modélisé comme un *containerRole* contenant quatre *containedRoles* numérotés de 1 à 4. *ContainedRole_1* et *containedRole_2* référencent le nom du *composant*, à savoir "Serveur Web" dans deux *playerInstances* distincts, de même *containedRole_3* et *containedRole_4* référencent l'adresse IP du *composant* dans différentes sources. Conformément au métamodèle du PVR Système présenté dans la section 3.3.2, le *roleInstance composant_ServeurWeb* possède un nom et une description, en l'occurrence "Serveur Web" et "IP = 172.16.0.10". Cet exemple illustre deux

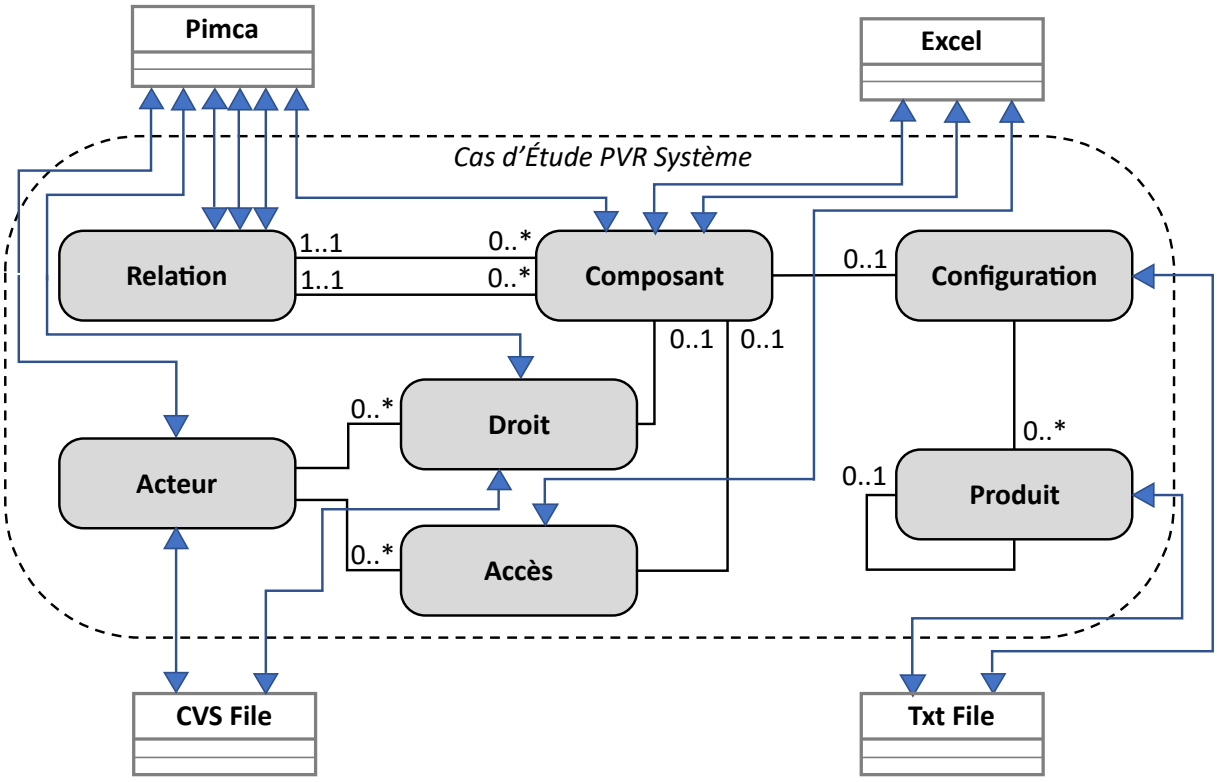


FIGURE 5.9 – Modèle de *rôle* du PVR système

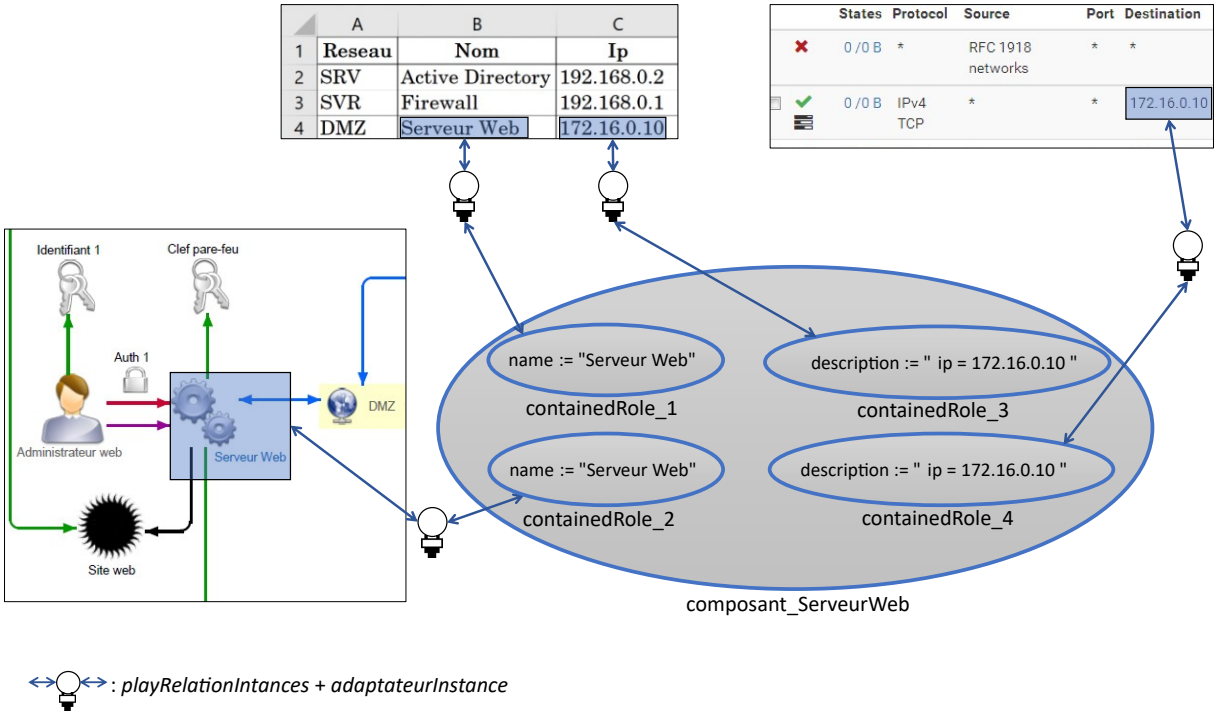


FIGURE 5.10 – Construction d'une instance du *rôle Composant*

des apports de la relation de contenance entre des instances de *rôles*, le regroupement et la fédération d'informations. En effet, *composant_ServeurWeb* regroupe des informations provenant de plusieurs *playerInstances* et permet de fédérer des données redondantes présentes dans différents modèles sources. Comme présenté dans la section 4.1.1, lorsqu'un

message est envoyé à *composant_ServeurWeb*, tel qu'une modification de l'adresse IP, ce dernier est transmis à l'ensemble des *containedRoles*. Les *roleInstances containedRole_1* et *containedRole_2* ne pouvant traiter le message ignorent ce dernier alors que *containedRole_3* et *containedRole_4* traitent le message et mettent à jour la valeur de l'adresse IP contenue dans leurs *playerInstances* respectives. Une fois les modifications effectuées, les *containedRoles* notifient *composant_ServeurWeb* du traitement du message, ce dernier peut alors confirmer à l'émetteur originel du message que le changement d'adresse IP est effectif.

5.2.2 Description de l'attaquant et création de son PVS

Dans cette section, nous présentons un type d'attaquant et modélisons son PVS sur le système présenté précédemment.

5.2.2.1 Description de l'attaquant

Nous étudions le point de vue d'un attaquant opportuniste nommé "Dimitri" cherchant à nuire le plus efficacement possible au système. L'attaquant n'a aucune connaissance à priori du système, mise a part l'existence du serveur web. Il dispose de peu de ressources et des capacités techniques limitées mais possède des compétences d'ingénierie sociale.

5.2.2.2 Construction du PVS

Pour cet exemple, nous utiliserons le même PVS que pour l'exemple fil rouge présenté en section 3.4.1.2. Le PVS de Dimitri est donc composé des mêmes *rôle* que celui de Mallory mais les deux attaquants n'ayant pas les même capacités et objectifs, le *contexte* associé au modèle de *rôle* est différent. La vue spécifique de Dimitri sur le système, conforme au PVS, est représenté en figure 5.11. L'attaquant est représenté par une *machine contrôlée* et le

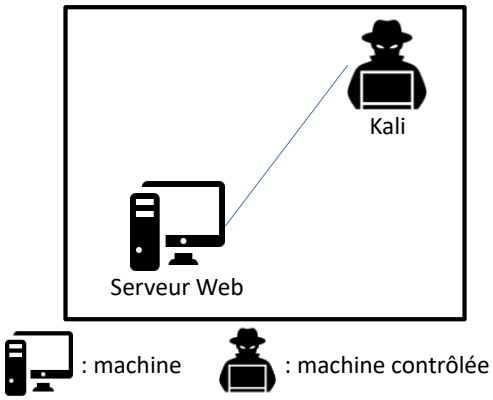


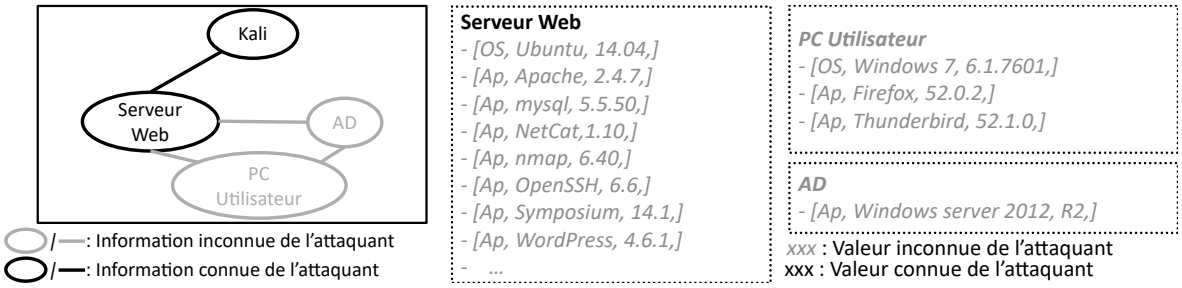
FIGURE 5.11 – Vue spécifique de Dimitri avant l'attaque

serveur web comme une *machine*. Le reste du système n'étant, à cette étape, pas connu de l'attaquant, il n'apparaît pas dans cette vue. De même, les commutateurs utilisés dans le système réel pour connecter l'attaquant au serveur web ne sont pas représentés ici car le PVS utilisé dans cet exemple se focalise sur les éléments manipulables par l'attaquant.

5.2.2.3 Adaptation des PVR pour le PVS de l'attaquant

Le PVS utilisé dans cet exemple étant le même que celui présenté en section 3.4.1.2, les *adapters* utilisés pour traduire les propriétés des PVR en propriétés du PVS sont les mêmes.

L'attaquant ne connaît, du moins au début de l'attaque, qu'une faction du système, cette connaissance est modélisée dans le PVR attaquant et permet de déterminer quels éléments sont connus ou non de l'attaquant. La figure 5.12 schématise ce que Dimitri connaît du système avant l'attaque. Au cours de son attaque Dimitri va découvrir peu à peu la topologie,



(a) Topologie connue de Dimitri

(b) Configurations connues de Dimitri

	Dimitri			Admin Web			Sylvain			Administrateur		
	D	AR	AP	D	AR	AP	D	AR	AP	D	AR	AP
Kali	●	●	●	○	○	○	○	○	○	○	○	○
Serveur Web	○	◐	○	●	●	●	○	●	◐	●	●	◐
PC Utilisateur	○	○	○	○	●	◐	◐	●	●	●	●	◐
AD	○	○	○	○	●	◐	○	●	◐	●	●	●

○ : Aucun
◐ : Utilisateur / Limité / Partiel
● : Administrateur / Illimité / Total

D : Droits | AR : Accès Réseau | AP : Accès Physique
Nom / ○ ◐ ● : Information inconnue de l'attaquant
Nom / ○ ◐ ● : Information connue de l'attaquant

(c) Acteurs, droits et accès connues de Dimitri

FIGURE 5.12 – Connaissances de Dimitri avant l'attaque

la configuration et les différents acteurs du système. Cette découverte est présentée dans la section 5.3.

Le PVS de Dimitri est composé de deux types d'éléments des *machines* et des *logiciels* spécialisés en *machine contrôlée*, *machine vulnérable* et *logiciel vulnérable*. Les instances du *rôle Machine* sont construites à partir des composants du système, des relations entre ces composants et des connaissances de l'attaquant. Une *machine* regroupe alors des informations provenant de différents *players*, ici les *rôles ElementConnu* et *Composant*. Dans Role4All, une *machine* est modélisée comme un *roleInstance* contenant deux *containedRoles*, référençant les *players* identifiés. Une *machine* est alors construite grâce à deux *adapters* dont le code est présenté ci-dessous :

```
Adapter subclass: #AdapterRoleElementConnuToRoleMachine
instanceVariableNames: ''
classVariableNames: ''

  getMachineName: obj1
    |name|
    name := obj1 nom.
    ↑name.
```

```
9
10          setMachineName: newName player: obj1
11            obj1 nom := newName.
12            ↑obj1 nom.
13
14
15        Adapter subclass: #AdapterRoleComposantToRoleMachine
16        instanceVariableNames: ''
17        classVariableNames: ''
18
19          getMachineName: obj1
20            |name|
21            name := obj1 nom.
22            ↑name.
23
24          setComposantName: newName player: obj1
25            obj1 nom: newName.
26            ↑obj1 nom.
27
28          getMachineRelations: obj1
29            |relations|
30            relations := (self getMachineRelationsEntrantes: obj1) +
31              (self getMachineRelationsSortantes: obj1).
32            ↑relations.
33
34          getMachineRelationsEntrantes: obj1
35            |relationsIn|
36            relationsIn := OrderedCollection new.
37            obj1 in: [rel: relationsIn addLast: (rel name)]
38            ↑relationsIn.
39
40          getMachineRelationsSortantes: obj1
41            |relationsOut|
42            relationsOut := OrderedCollection new.
43            obj1 out: [rel: relationsIn addLast: (rel name)]
44            ↑relationsOut.
45
46          addMachineRelationsEntrante: aRealation player: obj1
47            obj1 in add: aRealation.
48
49          ...
```

Par construction du PVS de Dimitri, les *rôles MachineVulnérable* et *MachineContrôlée* héritent du *rôle Machine*. Les *adapters* permettant de définir des instances de ces *rôles* sont construits comme des spécialisations de *AdapterRoleElementConnuToRoleMachine* et de *AdapterRoleComposantToRoleMachine*. L'utilisation de la relation d'héritage entre *adapters* permet de simplifier la création de nouvelles adaptations via la réutilisation de code existant.

5.3 Interprétation du point de vue de l'attaquant

Dans cette section, nous détaillons le comportement nominal du système et le comportement malveillant de Dimitri. Par la suite, nous adaptons le PVS de Dimitri afin de l'interpréter dans un point de vue dédié. L'objectif de cette interprétation est d'identifier un scénario d'attaque permettant de nuire au système. Pour finir, nous validons le scénario d'attaque identifié en exécutant ce dernier sur le système réel.

5.3.1 Description des comportements et adaptation du PVS pour le point de vue d'interprétation

Dans une première sous-section, nous décrivons le comportement nominal du système puis nous détaillons les objectifs de l'attaquant, et pour finir, nous interprétons le comportement de l'attaquant.

5.3.1.1 Description du comportement nominal du système

Le comportement nominal du système peut être défini via différents outils et être adapté pour le point de vue d'interprétation. Pour notre cas d'étude, nous définissons directement le comportement nominal du système via les *rôles* dédiés à l'interprétation. Nous définissons un ensemble d'*actions* et de variables d'environnement traduisant les activités réalisables par les différents acteurs du système. En plus de l'attaquant, nous modélisons trois acteurs capables d'agir sur le système : l'administrateur web, l'administrateur système et l'utilisateur Sylvain. Chaque acteur peut allumer ou éteindre sa machine sans prérequis particulier. Ce comportement est modélisé par une *action* consistant en la modification d'une variable d'*environnement* modélisant l'état de la machine. De manière analogue, un acteur peut exécuter un programme sur sa propre machine. Ce comportement peut être détourné par un attaquant, par exemple en envoyant un fichier vérolé à un acteur du système. L'envoi du fichier est alors modélisé via une *action* de type malveillant entraînant la création d'une nouvelle *action* de type nominal potentiellement nuisible au système (rançongiciel, etc.). En plus de ce comportement général, chaque acteur peut avoir un comportement spécifique, l'administrateur système peut modifier les droits des différents acteurs du système ou peut modifier les règles de routage du pare-feu, par exemple en bannissant une adresse IP. L'administrateur web peut quant à lui modifier le site web et potentiellement mettre à jour certains des logiciels présents sur le serveur.

Dans notre cas d'étude, nous définissons 37 *behaviorBlocks* de type nominal, à savoir :

— Deux *behaviorBlocks*, pour chaque machine du système, permettant de gérer l'état (allumé et éteint) de la machine.

— 27 *behaviorBlocks* permettant à l'administrateur système de retirer les droits d'un *acteur* ou de lui en attribuer de nouveaux et ce sur chaque *machine* du système.

— La possibilité de définir des *behaviorBlocks* permettant à l'administrateur web de modifier la version de chaque *logiciel* présent sur le serveur. Le nombre de *logiciel* et de versions étant important, nous ne définissons pour notre cas d'étude que 2 *behaviorBlocks* permettant respectivement de mettre à jour la version de Wordpress et celle de Symposium. Cette limitation permet de simplifier l'exécution manuelle de l'interpréteur.

Ces différents *behaviorBlocks* sont générés via l'exécution d'un script créant pour chaque *machine* et *acteur* du système les *behaviorBlocks* associés. Le pseudo code de ce script est donné ci-dessous :

```
1 For each instance i of the role named Machine :
2         Create a behaviorBlock named Start_i
3         Create a behaviorBlock named Shutdown_i
4
5 For each instance i of the role named Acteur :
6         For each instances j of the role named Machine :
7                 Create a behaviorBlock named RemoveRigthOf_i_On_j
8                 Create a behaviorBlock named Add_i_AsUserOf_j
9                 Create a behaviorBlock named Add_i_AsAdminOf_j
```

```
10
11 For each instances i of the role named Machine :
12         If i.nom == "Serveur Web" :
13                 For each instances j of the role named Logiciel
14                         If j.nom == "Wordpress" or "Symposium"
15                                 Create a behaviorBlock named
                                       UpdateLogiciel_jOf_i
```

5.3.1.2 Objectif de l'attaquant et comportement malveillant

L'objectif de l'attaquant est de nuire au système par tous les moyens à sa disposition. Pour notre cas d'étude, nous considérerons que l'attaquant a réussi son attaque si il arrive à rendre le site web indisponible, à corrompre le PC utilisateur ou à prendre la main sur l'AD.

Nous considérons dans notre exemple cinq types d'action possibles pour l'attaquant : découverte du système, analyse d'une machine, recherche de vulnérabilités, exécution d'une vulnérabilité et envoi d'un logiciel malveillant.

La découverte du système est modélisée par l'*action* '*SysDiscovery*' permettant de découvrir l'ensemble des *machines* en relation directe avec une *machine contrôlée*. L'analyse d'une machine est modélisée par l'*action* '*Scan*' permettant d'identifier une partie des *logiciels* présents sur une *machine*. La recherche de vulnérabilité est modélisée par l'*action* '*SearchVuln*' permettant de déterminer, pour un *logiciel* donné, l'ensemble des vulnérabilités exploitables par l'attaquant. L'exécution d'une vulnérabilité est modélisée par l'*action* '*ExecuteVuln*' permettant d'apporter des modifications au système conformément aux *effets* définis dans la *vulnérabilité*. L'envoi d'un logiciel malveillant est modélisé par l'*action* '*SendMalware*' permettant de générer une nouvelle *action* nommée '*ExecuteMalware*'. Les effets de cette dernière sont déterminés par l'attaquant mais l'*action* doit être déclenchée par un acteur légitime du système. Par exemple, l'envoi d'un email contenant un rançongiciel 'X' à l'acteur 'Y' est modélisé par le *behaviorBlock* '*SendMalware_X_to_Y*' et l'exécution de ce rançongiciel est modélisé par le *behaviorBlock* '*ExecuteMalware_X_on_Y*'.

Dans notre cas d'étude, nous associons une *action* '*SysDiscovery*' par *machine contrôlée*, une *action* '*Scan*' par *machine*, une *action* '*SearchVuln*' par *logiciel* et une *action* '*ExecuteVuln*' par *vulnérabilité*. Au cours d'une attaque, la vue de l'attaquant sur le système s'enrichit, par conséquent, de nouveaux *behaviorBlocks* peuvent être définis à chaque étape de l'attaque. Par exemple, si un attaquant découvre au cours d'une attaque la *machine* 'PC Utilisateur', alors, nous construisons automatiquement un nouveau *behaviorBlock* nommé '*Scan_PCUtilisateur*' permettant à l'attaquant d'exécuter une action de type 'Scan' sur le PC Utilisateur.

5.3.1.3 Interprétation du comportement de l'attaquant

Le point de vue d'interprétation permet d'associer à chaque élément du PVS de l'attaquant un ensemble de *behaviorBlocks*. Dans Role4All les *behaviorBlocks* sont des *rôles* et les éléments du PVS de l'attaquant sont considérés comme des *players*, la relation entre ces deux éléments est alors réalisée à l'aide de *playRelations* et d'*adapters*.

Pour notre cas d'étude, nous définissons six actions nommées *sysDiscovery*, *scan*, *searchVuln*, *executeVuln*, *start* et *shutdown*. Chaque action est construite comme une fonction, à laquelle est associée un *type* catégorisant l'action comme nominale ou malveillante, par exemple le pseudo-code de l'action malveillante *searchVuln* est donné ci-dessous :

```
1 searchVuln (logiciel)
```

```
2         If logiciel is vulnerable:
3                 Assign the role LogicielVulnerable to logiciel.
4                 Assign the role MachineVulnerable to the logiciel's
                      machine.
5                 vulns := List of all the vulnerabilities of
                     logiciel.
6                 For each vuln in vulns:
7                         Create a behaviorBlock named
                             ExecuteVuln_vuln.
```

Ce pseudo-code illustre trois comportements possibles lors de l'exécution d'une action : l'interaction avec un *player*, l'affectation d'un nouveau *rôle* et la création de *behaviorBlocks*.

— Interactions avec un *player* : à la ligne 5, nous listons l'ensemble des vulnérabilités présentes sur un *logiciel*. Pour ce faire, nous comparons le nom et la version de ce *logiciel* aux noms et versions de l'ensemble des actifs vulnérables des *vulnérabilités* modélisées. Le code relatif à cette comparaison utilise un accesseur afin de récupérer l'ensemble des actifs vulnérables connus.

— Affectation de nouveaux *rôles* : nous affectons un nouveau *rôle* à un *logiciel* aux lignes 3 et 4. Ce comportement permet de changer la représentation que l'attaquant a du *logiciel* observé. Il en est de même pour la *machine* contenant ce *logiciel*.

— Création de *behaviorBlocks* : aux lignes 6 et 7 nous créons un nouveau *behaviorBlock* pour chaque *vulnérabilité* identifiée. Les *guards* et *actions* de ces *behaviorBlocks* sont construites en fonction des *prérequis* et *impacts* de la *vulnérabilité*.

En plus de ces trois comportements, une action peut également modifier une variable d'environnement ou générer un événement.

Après avoir codé les différentes actions possibles pour l'attaquant, nous construisons le *program* correspondant à notre interpréteur :

```
interpretUseCase
|useCase behaviors environement|

"Create Environement"
  environement := Environement new
    envVars: ['attackInProgress', 'ServeurWebStatus',
        'PcUtilisateurStatus', 'AdStatus'];
    envEvents: [].

"Create behaviors"
  behaviors:= OrderedCollection new.
  behaviors add: (BehaviorBlock
    name: 'SysDiscovery_Kali';
    type: malveillant;
    guard: (Guard
      value: attackInProgress & ServeurWebStatus.
    ).
    action: (SysDiscovery: playRelation getMachinery).
  ).


  "For each machine X controled by the attacker we define
      the following behaviorBlocks : SysDiscovery_X."
  "For each machine X we define
      the following behaviorBlocks : Start_X, Shutdown_X."
  "For each machine X discovered by the attacker we define
      the following behaviorBlocks : Scan_X."
  "For each logiciel X discovered by the attacker we define
```

```
      the following behaviorBlocks : SearchVuln_X."
  "For each logicielVulnerable X discovered by the attacker we define
      the following behaviorBlocks : ExecuteVuln_X."
  "Create the behaviorBlocks LaunchAttack and StopAttack."


"Create Program"
  useCase:= (Program
    name: 'Use Case';
    environement: environement;
    behaviorBlocks: (behaviors asArray).
  ).
  useCase environement initEnv: [false,true,true,true].

"Execute program"
  useCase executeInteractiv.
```

5.3.2 Interprétation du comportement de l'attaquant

Une fois les *actions* et le *program* définis, il est possible d'interpréter les comportements modélisés. Dans notre cas d'étude, nous effectuons une interprétation interactive afin d'observer pas à pas les modifications apportées au point de vue de l'attaquant.

La figure 5.13 schématise les premiers pas d'interprétation de notre cas d'étude. Tout

***BehaviorBlocks* déclenchables**	**Point de vue de l'attaquant**
Comportement nominal ➢ Shutdown_AD ➢ Shutdown_PcUtilisateur ➢ Shutdown_ServeurWeb ➢ RemoveRightOf_Sylvain_On_PcUtilisateur ➢ Add_Sylvain_AsUserOf_ServeurWeb ➢ Add_Sylvain_AsAdminOf_ServeurWeb ➢ ... Comportement malveillant ➢ **LaunchAttack**	 Kali
Comportement nominal ➢ Shutdown_AD ➢ ... Comportement malveillant ➢ **SysDiscovery_Mallory** ➢ Scan_Mallory ➢ StopAttack	 Kali
Comportement nominal ➢ Shutdown_AD ➢ ... Comportement malveillant ➢ **Scan_ServeurWeb** ➢ SysDiscovery_Mallory ➢ Scan_Mallory ➢ StopAttack	 Kali  ServeurWeb
...	...

FIGURE 5.13 – Interprétation du comportement de l'attaquant

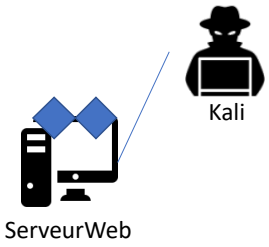
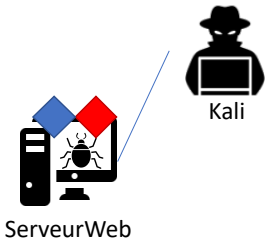
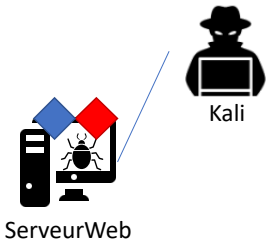
***BehaviorBlocks* déclenchables**	**Point de vue de l'attaquant**
Comportement nominal ➢ Shutdown_AD ➢ ... Comportement malveillant ➢ SearchVuln_Wordpress ➢ **SearchVuln_Symposium** ➢ Scan_ServeurWeb ➢ ...	
Comportement nominal ➢ Shutdown_AD ➢ ... Comportement malveillant ➢ **ExecuteVuln_CVE-2014-10021** ➢ ExecuteVuln_CVE-2014-8810 ➢ ExecuteVuln_CVE-2015-3325 ➢ ...	
Comportement nominal ➢ Shutdown_AD ➢ ... Comportement malveillant ➢ SearchVuln_Wordpress ➢ SearchVuln_Symposium ➢ **Scan_ServeurWeb** ➢ ...	
...	...

FIGURE 5.13 – Interprétation du comportement de l'attaquant

au long de l'interprétation, l'ensemble des *behaviorBlocks* relatifs au comportement nominal du système sont déclenchables, et ce, indépendamment des connaissances de l'attaquant. À l'opposé, le point de vue d'interprétation permet de déclencher un comportement malveillant sur un élément uniquement si ce dernier est connu de l'attaquant.

Le premier *behaviorBlock* déclenché est '*LaunchAttack*', l'action associée fixe la valeur de la variable d'environnement *attackInProgress* à *true*. Cette modification initialise le début de l'attaque et rend possible le déclenchement de *behaviorBlocks* de type malveillant. Par la suite, l'attaquant découvre, via le *behaviorBlock SysDiscovery_Mallory*, la *machine* représentant le serveur web. L'attaquant analyse alors cette *machine* via le *behaviorBlock Scan_ServeurWeb* et découvre deux *logiciels* : Wordpress et Symposium. La recherche de vulnérabilités sur le *logiciel* Symposium, via le *behaviorBlock SearchVuln_Symposium*, est concluante. A cette étape, nous savons que le serveur web contient des vulnérabilités exploitables, nous modifions alors la vue de l'attaquant afin de représenter le serveur web comme une *machine vulnérable* et Symposium comme un *logiciel vulnérable*. L'exploitation d'une des vulnérabilités découvertes sur Symposium, représentée par le *behaviorBlock ExecuteVuln_CVE-2014-10021*, accorde à l'attaquant des droits utilisateur. Ces nouveaux droits permettent à l'attaquant d'obtenir de nouvelles informations lors de du déclenchement du *behaviorBlock Scan_ServeurWeb*, il découvre alors le système d'exploitation utilisé par le serveur web. L'attaquant recherche alors les vulnérabilités relatives à ce système d'exploitation via le *behaviorBlock SearchVuln_Ubuntu*. L'exploitation d'une des vulnérabilités découvertes, représentée par le *behaviorBlock ExecuteVuln_CVE-2016-5195*, permet à l'attaquant d'obtenir des droits administrateur sur le serveur web. Le serveur web est dé-

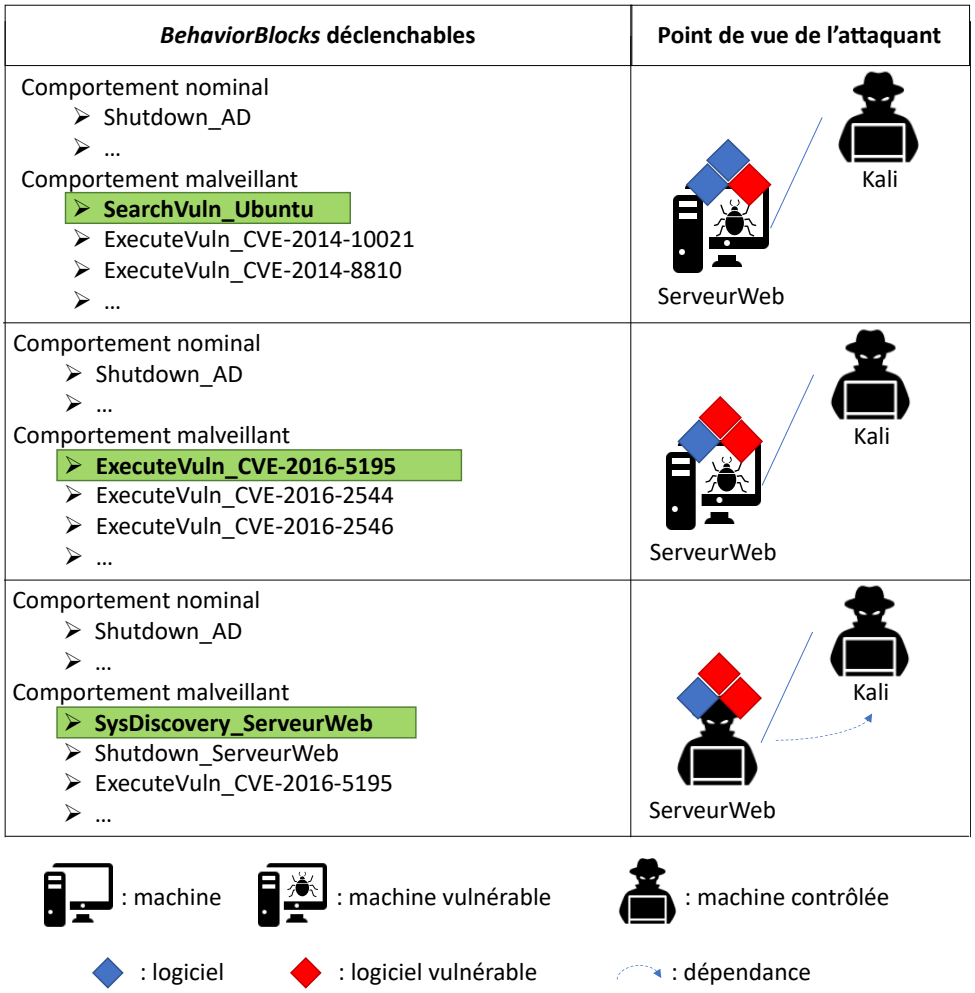


FIGURE 5.13 – Interprétation du comportement de l'attaquant

sormais contrôlé par l'attaquant, nous modifions alors la vue afin de représenter le serveur web comme une *machine contrôlée*. Il est alors possible pour l'attaquant de détourner le comportement nominal du système, par exemple en exécutant l'action *shutdown* sur le serveur web. La prise de contrôle du serveur web permet également à l'attaquant d'initier une nouvelle phase de découverte du système.

Le scénario représenté ici dépend entièrement du choix et de l'ordre des *behaviorBlocks* exécutés. Nous nous focalisons ici sur le déclenchement de *behaviorBlocks* malveillants, on peut cependant souligner qu'un scénario d'attaque peut également dépendre du comportement nominal du système. Par exemple, l'ouverture d'un email piégé est un comportement nominal du système car il est déclenché par un acteur légitime, cependant cette *action* peut permettre à un attaquant d'infecter une *machine* et de progresser dans son scénario d'attaque.

L'enchaînement de *behaviorBlocks* présenté sur la figure 5.13 constitue un scénario d'attaque amenant à la prise de contrôle du serveur web. L'attaquant peut alors utiliser le serveur web comme point de départ afin de corrompre le PC utilisateur ou de prendre le contrôle de l'AD. Dans la section 5.3.4, nous validons ce scénario d'attaque en l'exécutant directement sur le système réel.

Dans la section 5.3.3, nous utilisons le scénario d'attaque présenté dans cette section afin de générer des modèles de sécurité.

5.3.3 Génération de données pour des outils d'analyse formelle

Le point de vue d'interprétation permet de déclencher une suite de comportements pouvant résulter en un scénario d'attaque. Il offre également la possibilité, via une spécialisation, de construire des arbres et des graphes d'attaque à l'aide d'outils d'analyse formelle.

5.3.3.1 Génération d'arbres d'attaque

Nous soulignons dans la section 2.1.3 que la généricité apportée par les arbres d'attaque permet une grande diversité des analyses et une réutilisabilité des résultats. Ils sont, avec les graphes d'attaques, l'un des modèles de sécurité les plus répandus. Notre approche de modélisation de point de vue par les rôles permet, via une spécialisation du point de vue d'interprétation, la génération des arbres.

Comme présenté dans la section 2.1.3, un arbre d'attaque est un diagramme conceptuel des menaces d'un système et des attaques possibles pour atteindre ces menaces. Il est utilisé pour identifier différents chemins conduisant à un objectif prédéfini et pour construire une arborescence qui décrit comment une vulnérabilité aide un attaquant à atteindre son objectif. Il peut être enrichi par diverses métriques de sécurité représentant un coût, un profit, une chance de détection, une chance de réussite, une complexité, etc.

Afin de générer un arbre d'attaque, il est nécessaire de prendre en compte un certain nombre de métriques de sécurité, par exemple la chance de détection d'une action réalisée par l'attaquant. Pour ce faire, Role4All offre la possibilité de construire une spécialisation du point de vue d'interprétation, incluant la ou les métriques désirées, par exemple en ajoutant à chaque *behaviorBlock* une variable "*détection*" modélisant la probabilité pour un attaquant d'être détecté.

Dans notre cas d'étude, l'attaquant se connecte légitimement au serveur web, cette connexion semble inoffensive pour le système. Cependant, en prenant en compte le temps de réponse du serveur, on peut observer qu'un grand nombre de connexions rapprochées surchargent le serveur qui ne peut alors pas satisfaire toutes les demandes de connexion. Ce type de comportement est caractéristique d'une attaque par déni de service, les utilisateurs légitimes du système ne pouvant plus l'utiliser normalement. Cet exemple illustre qu'en plus de permettre la construction d'arbres d'attaque, la spécialisation du point de vue d'interprétation permet de détecter d'autres types d'attaque grâce à la prise en compte de nouvelles métriques.

L'objectif de Role4All est ici de produire des modèles de sécurité à partir de la fédération d'outils de modélisation de système tels que Pimca, de standards de cybersécurité tels que NVD, d'outils de gestions tels que Excel et de données non structurées. Les modèles graphiques ainsi générés peuvent alors servir de bases pour des analyses de sécurité plus complexes telles que des arbres d'attaque.

5.3.3.2 Analyse formelle et graphe d'attaque

Comme présenté dans la section 2.1.3, un graphe d'attaque représente une vue détaillée de la sécurité d'un système, il permet de déterminer si un attaquant peut atteindre des états finaux à partir d'un état initial. Il est composé de nœuds et de relations, les nœuds représentent des états et les relations font référence à des transitions entre différents états.

Afin de générer un graphe d'attaque, il est nécessaire de connaître tous les chemins permettant d'atteindre un état final à partir d'un état initial. L'analyse formelle, en particulier la vérification de modèles, est l'une des solutions permettant d'explorer l'ensemble des chemins d'exécution possible. Pour ce faire, Role4All propose deux solutions : l'analyse

via un outil externe ou la spécialisation du point de vue d'interprétation pour permettre la vérification de modèles au sein de Role4All.

Il existe de nombreux outils de vérification de modèles, tels que OBP2 [21, 22], permettant d'exhiber l'ensemble des chemins allant d'un état initial donné à un état final attendu. Role4All permet de fédérer des outils hétérogènes afin de construire le point de vue d'un attaquant, la même approche permet également de construire un point de vue dédié à un outil d'analyse formelle. Dans notre cas d'étude, il existe un scénario d'attaque permettant à un attaquant qui ne possède initialement aucun droit sur le serveur web de prendre le contrôle de ce dernier. Dans cet exemple, un outil de vérification de modèle permettrait de générer l'ensemble des suites d'actions possibles permettant à l'attaquant de mener à bien son attaque. Cependant, le nombre d'états possibles lors d'une attaque évolue exponentiellement, par exemple, pour un système de 10 *machines*, si chaque *machine* contient 10 *logiciels* et que chaque logiciel contient 10 *vulnérabilités* alors, il existe 1 000 transitions possibles à partir d'un état donné. Dans notre cas d'étude, l'attaquant prend le contrôle du serveur web en 8 étapes, pour un système de 10 machines, il existerait alors 10^{11} états à parcourir afin de découvrir tous les scénarii d'attaque. La vérification de modèle est donc une solution possible pour générer des graphes d'attaque mais, pour permettre un passage à l'échelle sur des systèmes complexes, des travaux supplémentaires, tels qu'une meilleure description des vulnérabilités, seraient nécessaires.

Role4All permet également, via une spécialisation du point de vue d'interprétation, d'effectuer de la vérification de modèle par les rôles. Cette approche nécessite la construction d'un point de vue complexe mais permet d'éviter la mise en place d'une couche d'adaptation supplémentaire.

L'implémentation actuelle de Role4All ne permet pas encore la génération de graphes d'attaque via des outils d'analyse formelle. Cependant, la construction de point de vue via des modèles de rôles et la spécialisation de point de vue sont deux solutions offertes par l'approche permettant la mise en place d'une telle solution.

5.3.4 Validation de l'attaque sur le système réel

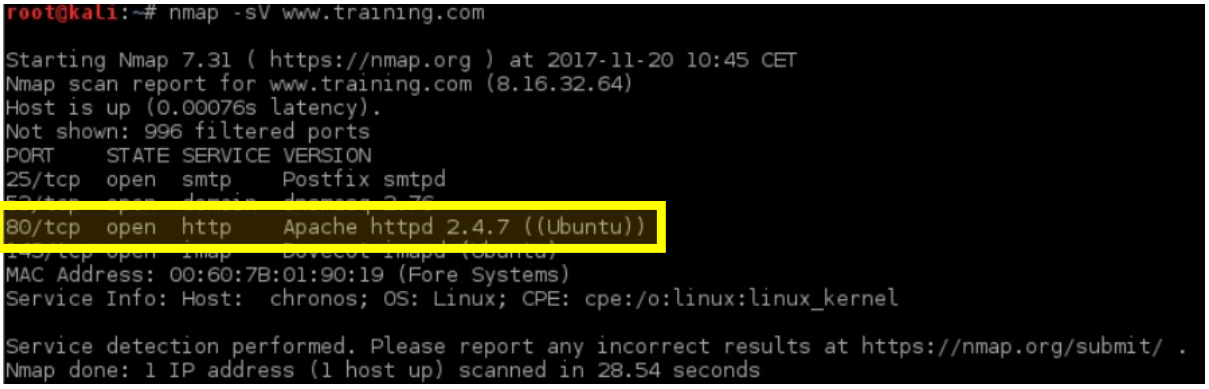
Nous présentons dans la section 5.3.2, un scénario d'attaque permettant à Dimitri de prendre le contrôle du serveur web. Dans cette section nous démontrons que cette attaque est réalisable sur le système réel.

Nous effectuons la phase de découverte du système à l'aide de NMAP, un outil permettant de réaliser une analyse du réseau. Le résultat de l'analyse est présentée en figure 5.14. Une première analyse 5.14(a) permet de découvrir le serveur web avec son adresse IP, ses ports ouverts, etc. On peut noter que le serveur possède, sur le port 80, un service http dispensé par le logiciel Apache. Une analyse approfondie 5.14(b) permet d'obtenir la version de WordPress utilisée sur le serveur.

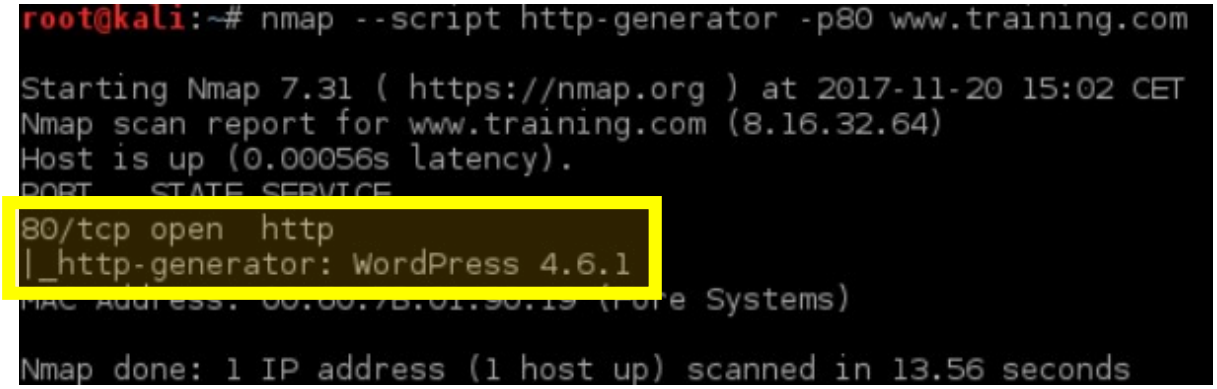
Afin d'obtenir plus d'informations sur le serveur web, nous utilisons WPScan, un outil d'analyse de sécurité dédié à WordPress. Le résultat de l'analyse est présenté en figure 5.15. La version de WordPress utilisée présente de nombreuses vulnérabilités, en particulier une de type "*Unauthenticated Shell Upload*" permettant l'ouverture d'une fenêtre de commande avec des droits utilisateur sur le serveur web.

Afin d'exploiter cette vulnérabilité, nous utilisons Metasploit, un outil de développement et d'exécution d'exploits contre une machine distante. Le mise en pratique de l'exploit est présentée en figure 5.16. L'exécution de la commande "*use exploit/ unix/ webapp/ wp_symposium shell upload*" entraîne l'ouverture d'une fenêtre de commande avec des droits utilisateur sur le serveur web, utilisable à distance par un attaquant.

A cette étape, nous avons découvert la configuration du serveur web, obtenu un accès réseau sur ce dernier et possédons des droits utilisateur sur cette machine. Comme



(a)



(b)

FIGURE 5.14 – Analyse du système via l'outil Nmap

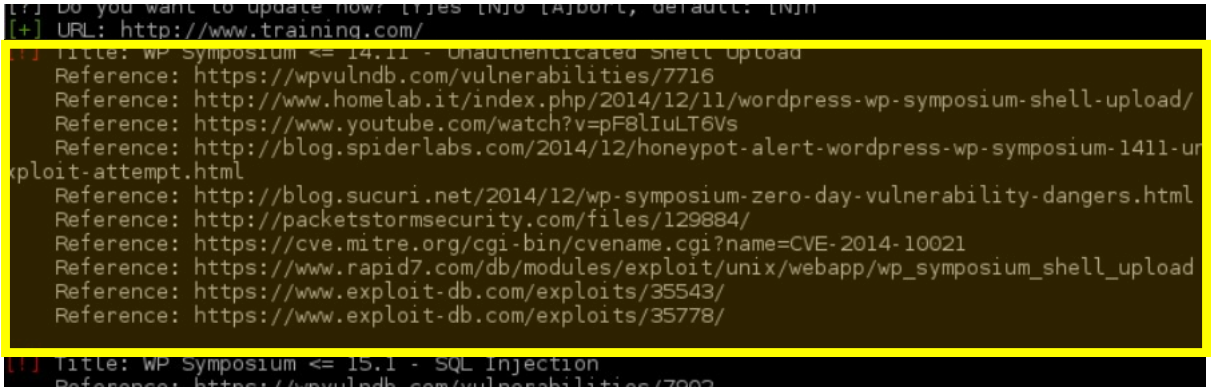


FIGURE 5.15 – Analyse du logiciel WordPress via l'outil WPScan

illustré en figure 5.17, nous utilisons les accès et droits obtenus afin de télécharger et exécuter *Cowroot*, une vulnérabilité d'élévation de privilège. L'exploitation de *Cowroot* via la commande "./*cowroot*" nous permet d'obtenir des droits administrateur sur la machine. L'objectif de notre attaque est atteint, à savoir prendre le contrôle du serveur web. Il est alors possible de continuer l'attaque en utilisant le serveur web comme porte d'entrée sur le système afin d'infecter le PC utilisateur ou de prendre le contrôle de l'AD.

L'utilisation d'outils tels que Nmap, WPScan ou Metasploit facilite le déroulement d'une attaque mais le scénario mis en exergue dans la section 5.3.2 est effectivement réalisable sur le système réel.

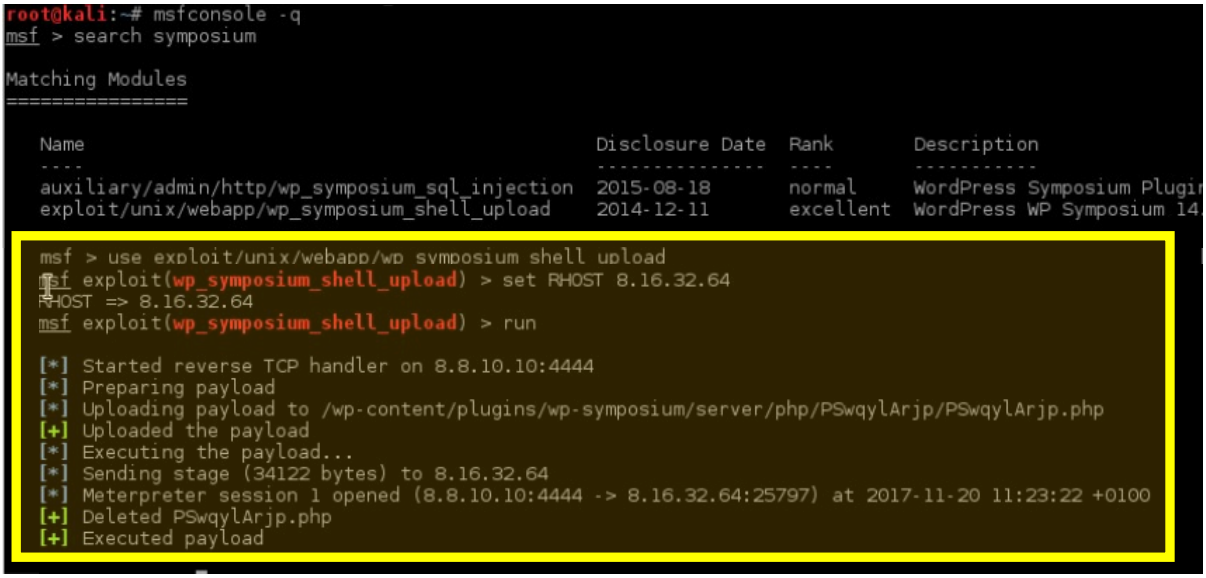


FIGURE 5.16 – Exécution d'un exploit via Metasploit

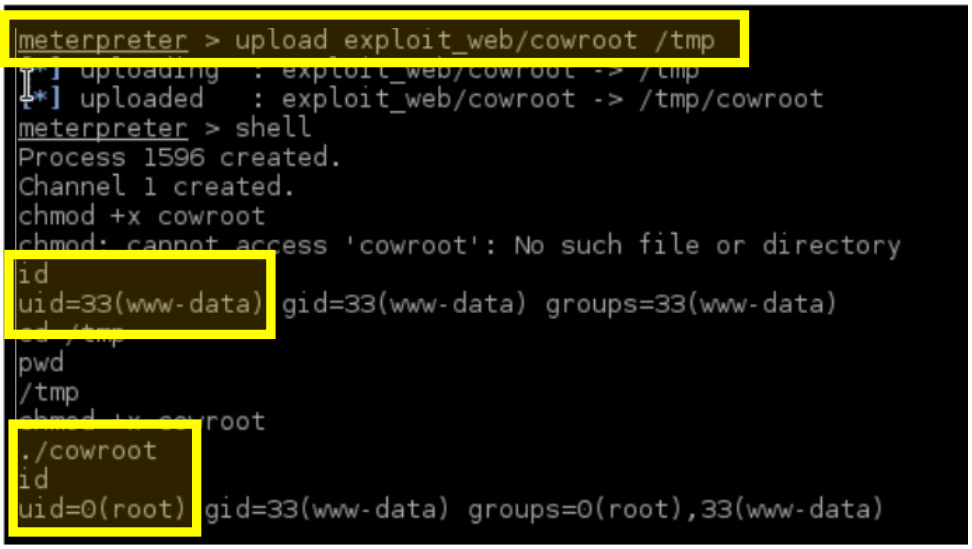


FIGURE 5.17 – Élévation de privilège

5.4 Conclusion

Nous mettons en avant dans le chapitre 2 que la modélisation du point de vue d'un attaquant requiert la mise en commun d'informations standardisées et provenant d'outils spécifiques. Dans les chapitres 3 et 4, nous construisons le point de vue d'un attaquant à l'aide de PVR et de PVS, de plus nous analysons ce dernier, via un point de vue d'interprétation.

Dans ce chapitre, nous validons notre approche de modélisation et d'analyse sur un cas d'études. Nous illustrons, en figure 5.18, l'ensemble des étapes permettant, à partir d'un système réel, de construire le point de vue d'un attaquant, d'interpréter ce dernier et de générer des modèles de sécurité.

En section 5.1, nous présentons et modélisons un cas d'étude à l'aide de différents modèles source. Nous utilisons des outils spécifiques tels que Pimca, des outils généralistes

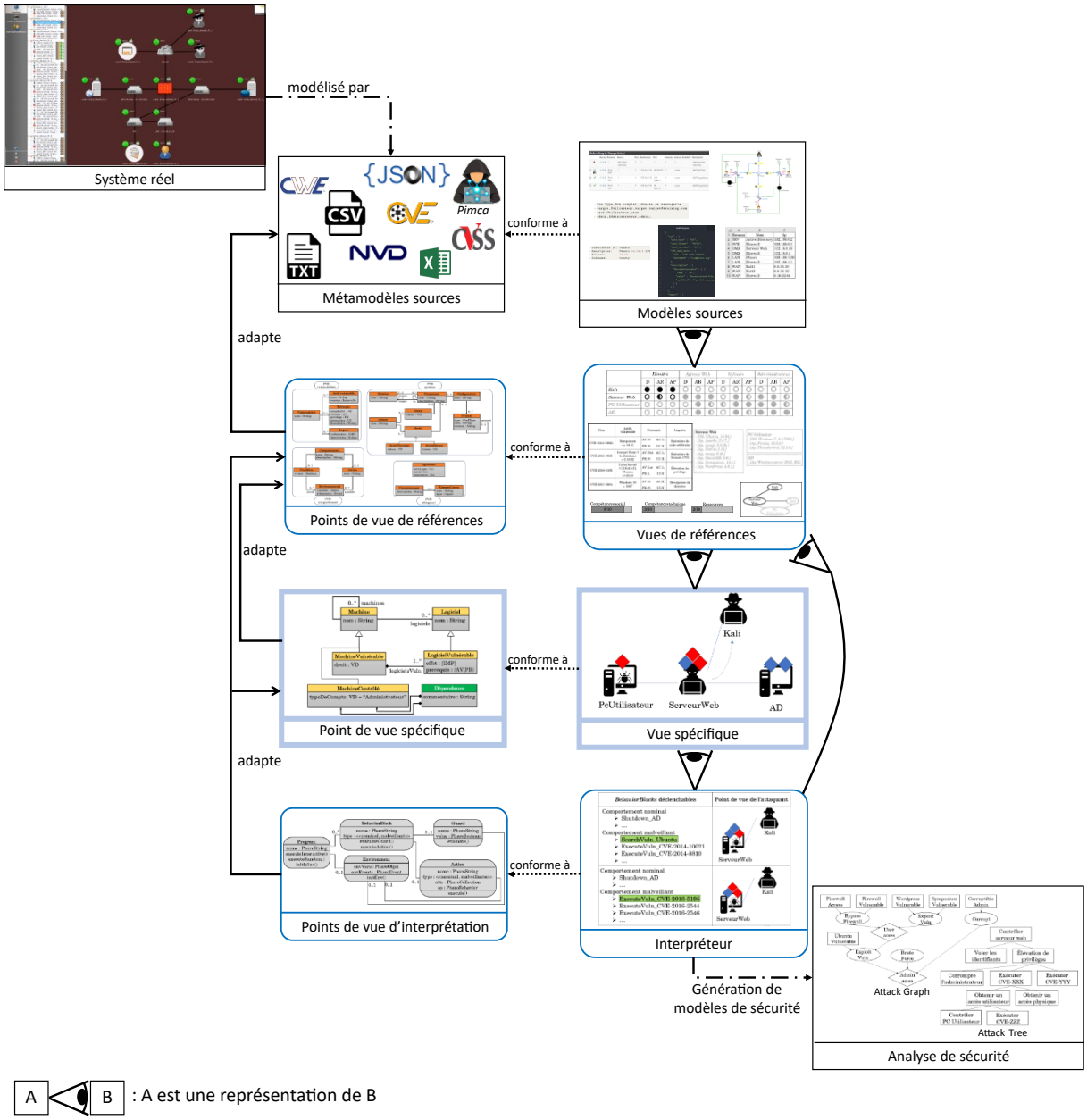


FIGURE 5.18 – Étapes permettant de construire et d'interpréter le point de vue d'un attaquant

tels que Excel et différents standards de sécurité tels que CVSS ou CVE afin de décrire un système et ses vulnérabilités.

En section 5.2.1, nous présentons comment l'ensemble des modèles source sont adaptés, afin de construire des vues de référence conformes aux PVR. La construction des vues de référence nécessite la mise en commun et la fédération d'informations contenues dans les modèles sources, pour ce faire, nous utilisons la relation de contenance implémentée dans Role4All.

En section 5.2.2, nous présentons les étapes de construction d'un PVS. Dans notre cas d'étude, nous réutilisons le modèle de *rôles* présenté dans la section 4.2.3 et définissons un nouveau *contexte* correspondant à la description type d'attaquant sélectionné. Le PVS proposé ne modélise que les éléments du système connu de l'attaquant, pour ce faire, nous

construisons des *roleInstances* regroupant des informations provenant de différents PVR. La construction de ces instances est réalisée à l'aide de *containerRoles* et de *containedRoles*.

En section 5.3, nous interprétons le point de vue de l'attaquant via un point de vue dédié. Nous exécutons une interprétation interactive permettant de découvrir un scénario d'attaque. Nous démontrons l'exécutabilité de ce scénario sur le système réel en section 5.3.4. Le point de vue d'interprétation permet également de générer des modèles de sécurité tel que des arbres et des graphes d'attaque. Les modèles générés peuvent être utilisés afin de mener un grand nombre d'analyses de sécurité complexes.

L'approche proposée dans ce manuscrit repose sur l'utilisation de Role4All, qui nous permet de construire un point de vue d'attaquant adaptable et interprétable, à partir d'un ensemble d'outils hétérogènes. Role4All offre la possibilité de représenter, à chaque étape d'une attaque, ce qu'un attaquant perçoit d'un système. Il permet également de produire, en fonction des métriques utilisées, divers types de données destinées aux analyses de sécurité.

Chapitre 6

Conclusion générale et perspectives

La modélisation et l'analyse de la menace est une approche de cybersécurité pro-active dirigée par des modèles, qui permet d'identifier et d'évaluer l'impact de différentes failles de sécurité. Les modèles utilisés représentent des systèmes, des menaces, des attaquants et des comportements. Dans ce manuscrit, nous nous focalisons sur les premières phases de cette approche en adoptant le point de vue de l'attaquant. Bien sûr, ce parti pris se place dans le cadre d'une approche de sécurité globale où nous cherchons à prendre la place de l'attaquant afin de mieux anticiper la défense. L'objectif est alors de représenter ce qu'un attaquant perçoit du système et de produire des modèles de sécurité dépendant des connaissances et des capacités de ce dernier.

6.1 Objectifs et problématiques du manuscrit

Nous avons déterminé, à partir de la littérature (§2.1), qu'il n'existe pas de consensus de la notion d'attaquant commune à l'ensemble des domaines de la cybersécurité. Le premier objectif de ce manuscrit est donc de définir et de modéliser la notion de point de vue d'attaquant, dans un contexte de modélisation et d'analyse de la menace. Cette définition ne cherche pas à unifier toutes les définitions ou encore à proposer la *meilleure* définition possible mais plutôt à proposer un cadre adaptable et évolutif. La solution proposée repose sur l'utilisation de deux types de points de vue, représentant respectivement les préoccupations communes à tous types d'attaquant et les préoccupations spécifiques à un contexte d'attaque. Le second objectif de ce manuscrit consiste en la construction et l'interprétation de ces points de vue.

Au cours de notre état de l'art, présenté dans le chapitre 2, nous identifions différentes problématiques liées à notre second objectif, à savoir :

1. La création d'un point de vue flexible offrant des relations bidirectionnelles avec les modèles sources couvrant les définitions courantes.
2. La prise en compte de l'hétérogénéité des sources d'information et de leur interopérabilité, reposant sur une mise en relation dynamique, non intrusive et adaptative.
3. La génération de modèles de sécurité prenant en compte le comportement de l'attaquant mais également celui du système attaqué.

Dans ce manuscrit, nous proposons de s'appuyer sur Role4All, notre solution de modélisation et d'interprétation de points de vue, reposant sur la notion de rôle. Role4All permet de fédérer des modèles hétérogènes et offre la possibilité d'interpréter différents comportements. Les informations produites par Role4All peuvent être utilisées pour faciliter la

compréhension des problèmes de sécurité par un humain ou pour produire des données exploitables par des outils d'analyse de cybersécurité.

Nous récapitulons, dans la suite de cette conclusion, les contributions permettant de répondre à nos objectifs tout en proposant des solutions aux problématiques identifiées.

Les travaux exposés dans ce manuscrit ont fait l'objet de trois publications internationales, dont deux [48, 49] focalisées sur notre contribution majeure, à savoir, la construction et l'interprétation du point de vue d'un attaquant :

— [48] Bastien Drouot and Joël Champeau. Model federation based on role modeling. In MODELSWARD, pages 72–83, 2019.

— [49] Bastien Drouot, Fahad R Golra, and Joël Champeau. A role modeling based approach for cyber threat analysis. In International Conference on Model-Driven Engineering and Software Development, pages 76–100. Springer, 2019.

— [157] Tithnara Sun, Bastien Drouot, Fahad Golra, Joël Champeau, Sylvain Guerin, Luka Le Roux, Raúl Mazo, Ciprian Teodorov, Lionel Aertryck, and Bernard L'Hostis. A domain-specific modeling framework for attack surface modeling. In International Conference on Information Systems Security and Privacy, 2020.

6.2 Contributions du manuscrit

La définition et la modélisation de la notion de point de vue d'attaquant constituent notre première contribution. Présentées dans le chapitre 3, elles reposent sur la définition de la notion d'attaquant dans le domaine de la modélisation et l'analyse de la menace. Nous proposons cette définition, construite à partir des standards de cybersécurité : *un individu, un groupe ou un état doté d'aptitudes à exploiter des vulnérabilités et qui met en œuvre ses connaissances sur les composants, les relations et les acteurs d'un système d'information, afin de nuire à la disponibilité, la confidentialité et/ou l'intégrité d'actifs de ce système*.

À partir de cette définition, propre à notre contexte d'étude, nous identifions les préoccupations communes à tous types d'attaquants. Elles portent sur la structure et les vulnérabilités du système attaqué, le comportement des parties prenantes et les connaissances, aptitudes et objectifs de l'attaquant. Modéliser un point de vue d'attaquant de manière générique et partagée est illusoire dans le cadre de notre travail. La somme des expertises et expériences est humainement hors d'atteinte et en perpétuelle évolution. C'est pourquoi, nous avons chercher à identifier une proposition basée sur la flexibilité et l'adaptabilité à différents contextes. Cette proposition repose sur un cadre de modélisation constitué de deux types de point de vue pouvant s'adapter dynamiquement. Premièrement, nous construisons quatre points de vue de référence (PVR) décrivant les préoccupations identifiées précédemment. Ils permettent de modéliser l'ensemble des informations nécessaires pour représenter la perception d'un attaquant. Par la suite, nous construisons des points de vue spécifiques (PVS). Un PVS est construit à partir des PVR et de spécificités propres au contexte de l'attaque. Un PVS représente alors les informations contenues dans les PVR, ces informations étant elles-mêmes une représentation des données provenant des modèles sources. Nous concluons le chapitre 3 en identifiant 20 caractéristiques issues des travaux de Bruneliere et al. [27] nécessaires pour la création du point de vue d'un attaquant. Ce cadre de modélisation étant posé, nous couvrons la première problématique identifiée de définition et de modélisation de la notion de point de vue d'attaquant.

Notre seconde contribution, présentée dans le chapitre 4, répond à la problématique de construction et d'interprétation du point de vue d'un attaquant. Elle repose sur le concept de rôle pour la fédération de modèles via notre langage Role4All. Notre approche a été construite afin de couvrir les caractéristiques de construction de points de vue identifiées

précédemment et en cherchant à répondre aux 3 problématiques majeures (§6.1) issues du cadre de modélisation. Afin de positionner notre approche par rapport à l'état de l'art, nous décrivons formellement notre utilisation des rôles dans la section 4.1.2 et nous justifions, dans la section 4.2, le choix de couvrir 19 des 27 caractéristiques de rôle proposées par Kühn [100]. De plus, nous présentons en section 4.1.3 une implémentation de Role4All en tant que DSL interne à Pharo.

Le métamodèle de Role4All, présenté en section 4.1.1 repose sur l'utilisation de cinq concepts principaux : *rôle*, *player*, *playRelation*, *adapter* et *context*. Dans Role4All, les concepts d'un métamodèle source sont considérés comme des *players* et un point de vue est représenté par un modèle de *rôles*. L'association entre un *player* et un *rôle* dépend d'un *context*. De plus, un *player* est lié à un *rôle* par une *playRelation* et un *adapter*. L'*adapter* traduit les propriétés d'un *player* en propriétés d'un *rôle* et inversement. Il fournit alors la flexibilité et l'adaptation dynamique des modèles sources, ce qui répond à la première problématique identifiée.

Pour rappel (§4.2), notre fédération de modèles repose essentiellement sur l'utilisation de *containerRole* et de *containedRoles* qui nous permettent de fédérer des modèles hétérogènes de manière non-intrusive. Dans notre approche, les informations contenues dans les *rôles* sont une représentation des données provenant des modèles sources, il n'y a donc pas de duplication d'information. Cette caractéristique permet de capturer directement, dans une vue, toute modification apportée à un modèle source. De plus, Role4All permet d'associer ou de dissocier un *rôle* d'un *player* à tout moment. Cette caractéristique permet de modifier le point de vue d'un attaquant en cours d'exécution. Finalement, Role4All met en place une fédération de modèles hétérogènes non-intrusive, qui prend en compte l'évolution des sources d'information et qui peut s'adapter aux modifications du point de vue d'un attaquant, ce qui répond à la deuxième problématique identifiée.

La troisième problématique identifiée est couverte via un point de vue d'interprétation présenté en section 4.3. Ce dernier est une représentation du point de vue de l'attaquant dédié à l'interprétation. Il permet d'interpréter le comportement de l'attaquant et le comportement du système via des *rôles* spécifiques. Ainsi, le point de vue d'interprétation offert par Role4All permet de générer des modèles de sécurité tels que des arbres d'attaque.

Par ailleurs, le chapitre 5 nous permet de valider notre approche sur un cas d'étude. Nous illustrons dans ce chapitre les relations entre nos différentes contributions. Tout d'abord, les informations des modèles sources sont mises en forme dans des PVR. Puis, les informations des PVR sont composées et adaptées pour un PVS représentant le point de vue d'un attaquant spécifique. Finalement, ce PVS est analysé via un point de vue d'interprétation afin de produire des modèles de sécurité.

6.3 Perspectives

Les travaux présentés dans ce manuscrit répondent aux objectifs identifiés lors de notre état de l'art (§2). Toutefois, plusieurs perspectives d'évolution sont envisageables à court et moyen terme. Elles sont principalement axées sur la caractérisation du domaine d'études et l'évolution de Role4All.

La figure 6.1 présente une schématisation de notre approche de modélisation de point de vue d'attaquant incluant les perceptives d'évolution identifiées, à savoir :

① Les PVR proposés dans ce manuscrit sont modélisés pour décrire les préoccupations d'un attaquant dans un contexte particulier, celui de la modélisation et l'analyse de la menace. La prise en compte de nouveaux contextes d'étude est envisageable.

De nouveaux PVR décrivant les nouvelles préoccupations prises en compte pourraient être définis. Par exemple, pour représenter le point de vue d'un défenseur, il faudrait

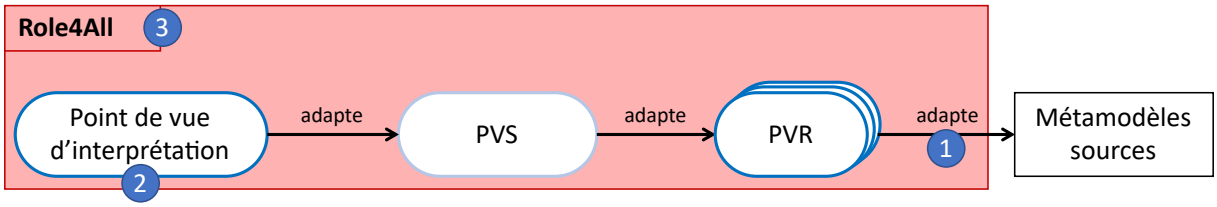


FIGURE 6.1 – Perspective d'évolution de l'approche

prendre en compte de nouvelles préoccupations telles que des coûts de sécurisation, les frontières du système ou encore les besoins des utilisateurs.

La prise en compte de nouvelles préoccupations permettrait de représenter et d'interagir avec un même système, simultanément, selon différents points de vue. Il serait alors possible de simuler les interactions entre un attaquant et un défenseur lors d'une attaque.

② Le développement actuel de Role4All offre la possibilité de générer des arbres d'attaque via l'utilisation d'un point de vue d'interprétation. Il serait envisageable d'étendre les outils d'analyse proposés par Role4All afin d'y inclure des approches de vérification formelle de modèles. Une telle approche permettrait de générer des graphes d'attaque et d'identifier de nouveaux scénarii d'attaque violant des propriétés de sécurité.

Une possibilité repose sur la fédération d'un outil d'analyse formelle, tel que OBP2 [21, 22]. La fédération ne permet pas dupliquer d'information mais requiert la construction de nouveaux modèles de *rôles* et de nouveaux *adapter*. Cette solution permettrait d'élargir les capacités d'analyse de Role4All et de rechercher automatiquement des scénarii d'attaque.

③ Dans ce manuscrit, nous implémentons Role4All comme un DSL interne à Pharo. Ce choix nous a permis un prototypage rapide du langage. Cependant, pour optimiser Role4All, nous envisageons d'implémenter notre approche dans un autre langage ou un framework de modélisation existant.

Pour ce faire, nous privilégions actuellement OpenFlexo qui est un environnement de modélisation dédié à la fédération de modèle. Nous évoquons dans la section 3.5 que cette solution couvre un grand nombre des caractéristiques nécessaires à la création de points de vue d'attaquant. Surtout, l'utilisation d'OpenFlexo faciliterait l'association avec des outils externes (via la concept de *Model Slot*) et offrirait un support de création d'interface graphique. De plus, l'outil Pimca, utilisé dans notre cas d'étude (§5), est implémenté dans OpenFlexo. L'emploi d'OpenFlexo nous permettrait de rapprocher l'outil de représentation et d'analyse de système de notre approche de modélisation et d'analyse de points de vue d'attaquants.

Les travaux présentés dans cette thèse offrent potentiellement d'autres perspectives. Cependant, la perpétuelle évolution propre à de nombreux domaines, telle que la cybersécurité, requiert d'incessantes adaptations conceptuelles et techniques. C'est pourquoi, il nous semble important d'augmenter la diffusion des approches de fédération de modèles, telle que notre approche basée sur les rôles.

Liste de publications

Publication internationales

Publications comme premier auteur

- [48] Bastien Drouot and Joël Champeau. Model federation based on role modeling. In MODELSWARD, pages 72–83, 2019.
- [49] Bastien Drouot, Fahad R Golra, and Joël Champeau. A role modeling based approach for cyber threat analysis. In International Conference on Model-Driven Engineering and Software Development, pages 76–100. Springer, 2019.

Autres publications

- [157] Tithnara Sun, Bastien Drouot, Fahad Golra, Joël Champeau, Sylvain Guerin, Luka Le Roux, Raúl Mazo, Ciprian Teodorov, Lionel Aertryck, and Bernard L'Hostis. A domain-specific modeling framework for attack surface modeling. In International Conference on Information Systems Security and Privacy, 2020.

Communications par poster

- Bastien Drouot and Joël Champeau, *Modeling and simulating the attacker's point of view*, Séminaire MOCS, 2018.
- Bastien Drouot, Vincent Leilde, Jean-Philippe Schneider, Jean-Christophe Le Lann and Joël Champeau *Toolkit Dedicated to System Discovery*, Séminaire MOCS, 2017.

Glossaire

AD :	*Active Directory*
CR :	Concept(s) de Référence(s)
CS :	Concept(s) Spécifique(s)
CTA :	*Cyber Threat Analysis*
CVE :	*Common Vulnerabilities and Exposures*
CVSS :	*Common Vulnerability Scoring System*
CWE :	*Common Weakness Enumeration*
DSML :	*Domain Specific Modeling Language*
IDM :	Ingénierie Dirigée par les Modèles
IR :	Instance(s) de Référence(s)
IS :	Instance(s) Spécifique(s)
NVD :	*National Vulnerability Database*
PVR :	Point(s) de Vue de Référence(s)
PVS :	Point(s) de Vue Spécifique(s)
SSH :	*Secure SHell*
VR :	Vue(s) de Référence(s)
VS :	Vue(s) Spécifique(s)

Bibliographie

[1] Common vulnerabilities and exposures. https://cve.mitre.org/, .

[2] Cve vulnerabilities by type. https://www.cvedetails.com/vulnerabilities-by-types.php, .

[3] Common vulnerability scoring system, v3.1. https://www.first.org/cvss/calculator/3.1. Accédé : 2020-05-06.

[4] Common weakness enumeration. https://cwe.mitre.org/.

[5] Hynesim hybrid network simulation. https://www.hynesim.org/.

[6] National vulnerability database. https://nvd.nist.gov/.

[7] Structured threat information expression. https://oasis-open.github.io/cti-documentation/stix/intro.

[8] military terms alphabet list. https://www.militaryfactory.com/dictionary/military-terms-alphabet-list.asp.

[9] Technologies de l'information – techniques de sécurité – systèmes de management de la sécurité de l'information – vue d'ensemble et vocabulaire. *ISO/IEC 27000 :2018*, pages 1–38, February 2018.

[10] Sarfraz Alam, Mohammad MR Chowdhury, and Josef Noll. Interoperability of security-enabled internet of things. *Wireless Personal Communications*, 61(3) :567–586, 2011.

[11] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224, 2002.

[12] Egil P Andersen. Conceptual modeling of objects–a role modeling approach. 1997.

[13] Egil P Andersen and Trygve Reenskaug. System design by composing structures of interacting objects. In *European Conference on Object-Oriented Programming*, pages 133–152. Springer, 1992.

[14] Colin Atkinson and Thomas Kühne. Rearchitecting the uml infrastructure. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 12(4) :290–321, 2002.

[15] Charles W Bachman and Manilal Daya. The role concept in data models. In *Proceedings of the third international conference on Very large data bases-Volume 3*, pages 464–476, 1977.

[16] Matteo Baldoni, Guido Boella, and Leon van der Torre. Roles as a coordination construct : Introducing powerjava. *Electronic Notes in Theoretical Computer Science*, 150(1) :9–29, 2006.

[17] François Bancilhon and Nicolas Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems (TODS)*, 6(4) :557–575, 1981.

[18] FSRBM Barbosa and Ademar Aguiar. Modeling and programming with roles : introducing javastage. In *11th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMeT12)*, 2012.

[19] Daniel Bardou. Roles, subjects and aspects : How do they relate ? In *European Conference on Object-Oriented Programming*, pages 418–419. Springer, 1998.

[20] Charles H Bennett. Logical reversibility of computation. *IBM journal of Research and Development*, 17(6) :525–532, 1973.

[21] Valentin Besnard, Matthias Brun, Frédéric Jouault, Ciprian Teodorov, and Philippe Dhaussy. Unified ltl verification and embedded execution of uml models. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 112–122, 2018.

[22] Valentin Besnard, Ciprian Teodorov, Frédéric Jouault, Matthias Brun, and Philippe Dhaussy. Verifying and monitoring uml models with observer automata : A transformation-free approach. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 161–171. IEEE, 2019.

[23] Frédéric Besson, Nataliia Bielova, and Thomas Jensen. Hybrid monitoring of attacker knowledge. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 225–238. IEEE, 2016.

[24] Aaron Bohannon, J Nathan Foster, Benjamin C Pierce, Alexandre Pilkiewicz, and Alan Schmitt. Boomerang : resourceful lenses for string data. In *Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 407–419, 2008.

[25] Hugo Bruneliere, Jokin Garcia, Philippe Desfray, Djamel Eddine Khelladi, Regina Hebig, Reda Bendraou, and Jordi Cabot. On lightweight metamodel extension to support modeling tools agility. In *European Conference on Modelling Foundations and Applications*, pages 62–74. Springer, 2015.

[26] Hugo Bruneliere, Jokin Garcia Perez, Manuel Wimmer, and Jordi Cabot. Emf views : A view mechanism for integrating heterogeneous models. In *International Conference on Conceptual Modeling*, pages 317–325. Springer, 2015.

[27] Hugo Bruneliere, Erik Burger, Jordi Cabot, and Manuel Wimmer. A feature-based survey of model view approaches. *Software & Systems Modeling*, 18(3) :1931–1952, 2019.

[28] Hugo Bruneliere, Florent Marchand de Kerchove, Gwendal Daniel, Sina Madani, Dimitris Kolovos, and Jordi Cabot. Scalable model views over heterogeneous modeling technologies and resources. *Software and Systems Modeling*, 2020.

[29] Antonio Bucchiarone, Jordi Cabot, Richard F Paige, and Alfonso Pierantonio. Grand challenges in model-driven engineering : an analysis of the state of the research. *Software and Systems Modeling*, pages 1–9, 2020.

[30] Sabine Buckl, Sascha Krell, and Christian M Schweda. A formal approach to architectural descriptions–refining the iso standard 42010. In *International Workshop on Cooperation and Interoperability, Architecture and Ontology*, pages 77–91. Springer, 2010.

[31] Loli Burgueño, Jordi Cabot, and Sébastien Gérard. The future of model transformation languages : An open community. *Journal of Object Technology*, 18(3), 2019.

[32] Jordi Cabot and Ruth Raventós. Roles as entity types : A conceptual modelling pattern. In *International Conference on Conceptual Modeling*, pages 69–82. Springer, 2004.

[33] Jordi Cabot and Ruth Raventós. Conceptual modelling patterns for roles. In *Journal on Data Semantics V*, pages 158–184. Springer, 2006.

[34] Joël Champeau, Vincent Leildé, and Papa Issa Diallo. Model federation in toolchains. 2013.

[35] David Chen, Nicolas Daclin, et al. Framework for enterprise interoperability. In *Proc. of IFAC Workshop EI2N*, pages 77–88. Bordeaux, 2006.

[36] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM transactions on database systems (TODS)*, 1(1) :9–36, 1976.

[37] Zhongqiang Chen, Yuan Zhang, and Zhongrong Chen. A categorization framework for common computer vulnerabilities and exposures. *The Computer Journal*, 53(5) : 551–580, 2010.

[38] Daniel Chernuchin and Gisbert Dittrich. Role types and their dependencies as components of natural types. In *2005 AAAI Fall Symposium : Roles, an interdisciplinary perspective*, 2005.

[39] Krzysztof Czarnecki and Simon Helsen. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, volume 45, pages 1–17. USA, 2003.

[40] Krzysztof Czarnecki, J Nathan Foster, Zhenjiang Hu, Ralf Lämmel, Andy Schürr, and James F Terwilliger. Bidirectional transformations : A cross-discipline perspective. In *International Conference on Theory and Practice of Model Transformations*, pages 260–283. Springer, 2009.

[41] Mohamed Dahchour, Alain Pirotte, and Esteban Zimányi. A generic role model for dynamic objects. In *International Conference on Advanced Information Systems Engineering*, pages 643–658. Springer, 2002.

[42] Umeshwar Dayal and Philip A Bernstein. On the correct translation of update operations on relational views. *ACM Transactions on Database Systems (TODS)*, 7(3) : 381–416, 1982.

[43] Csaba Debreceni, Ákos Horváth, Ábel Hegedüs, Zoltán Ujhelyi, István Ráth, and Dániel Varró. Query-driven incremental synchronization of view models. In *Proceedings of the 2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling*, pages 31–38, 2014.

[44] Rinku Dewri. Location privacy and attacker knowledge : who are we fighting against? In *International Conference on Security and Privacy in Communication Systems*, pages 96–115. Springer, 2011.

[45] Edsger W Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8) :453–457, 1975.

[46] Zinovy Diskin, Yingfei Xiong, and Krzysztof Czarnecki. Specifying overlaps of heterogeneous models for global consistency checking. In *Proceedings of the First International Workshop on Model-Driven Interoperability*, pages 42–51, 2010.

[47] Josep Domingo-Ferrer, Sara Ricci, and Jordi Soria-Comas. Disclosure risk assessment via record linkage by a maximum-knowledge attacker. In *2015 13th Annual Conference on Privacy, Security and Trust (PST)*, pages 28–35. IEEE, 2015.

[48] Bastien Drouot and Joël Champeau. Model federation based on role modeling. In *MODELSWARD*, pages 72–83, 2019.

[49] Bastien Drouot, Fahad R Golra, and Joël Champeau. A role modeling based approach for cyber threat analysis. In *International Conference on Model-Driven Engineering and Software Development*, pages 76–100. Springer, 2019.

[50] Seventh Edition. The authoritative dictionary of ieee standards terms. In *IEEE Std. 100-2000*, pages 1–1362. 2000.

[51] Bouchra El Asri, Mahmoud Nassar, Abdelaziz Kriouile, and Bernard Coulette. Views, subjects, roles and aspects : A comparison along software lifecycle. In *ICEIS (3)*, pages 139–146, 2004.

[52] Matthew Emerson and Janos Sztipanovits. Techniques for metamodel composition. In *OOPSLA–6th Workshop on Domain Specific Modeling*, pages 123–139, 2006.

[53] Alexander M Ernst et al. Enterprise architecture management pattern catalog (version 1.0. In *Chair for Informatics 19 (sebis), Technische Universit%■t*. Citeseer, 2008.

[54] Shamal Faily and Ivan Fléchais. Barry is not the weakest link : Eliciting secure system requirements with personas. 2010.

[55] Stefan Feldmann, Manuel Wimmer, Konstantin Kernschmidt, and Birgit Vogel-Heuser. A comprehensive approach for managing inter-model inconsistencies in automated production systems engineering. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1120–1127. IEEE, 2016.

[56] David Ferraiolo, Janet Cugini, and D Richard Kuhn. Role-based access control (rbac) : Features and motivations. In *Proceedings of 11th annual computer security application conference*, pages 241–48, 1995.

[57] Anthony Finkelstein, Jeff Kramer, Bashar Nuseibeh, Ludwik Finkelstein, and Michael Goedicke. Viewpoints : A framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, 2(01) :31–57, 1992.

[58] J Nathan Foster, Michael B Greenwald, Jonathan T Moore, Benjamin C Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations : A linguistic approach to the view-update problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(3) :17–es, 2007.

[59] Daniel Fraunholz, Simon Duque Anton, and Hans Dieter Schotten. Introducing gamfis : A generic attacker model for information security. In *2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6. IEEE, 2017.

[60] Shivi Garg, RK Singh, and AK Mohapatra. Analysis of software vulnerability classification based on different technical parameters. *Information Security Journal : A Global Perspective*, 28(1-2) :1–19, 2019.

[61] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[62] Nirnay Ghosh and SK Ghosh. An intelligent technique for generating minimal attack graph. In *First Workshop on Intelligent Security (Security and Artificial Intelligence)(SecArt'09)*, 2009.

[63] Thomas Goldschmidt, Steffen Becker, and Erik Burger. Towards a tool-oriented taxonomy of view-based modelling. *Modellierung 2012*, 2012.

[64] Fahad R Golra, Antoine Beugnard, Fabien Dagnat, Sylvain Guerin, and Christophe Guychard. Addressing modularity for heterogeneous multi-model systems using model federation. In *Companion Proceedings of the 15th International Conference on Modularity*, pages 206–211, 2016.

[65] Georg Gottlob, Michael Schrefl, and Brigitte Röck. Extending object-oriented systems with roles. *ACM Transactions on Information Systems (TOIS)*, 14(3) :268–296, 1996.

[66] Kasper B Graversen and Kasper Østerbye. Aspect modelling as role modelling. In *OOPSLA 2002 Workshop on TS4AOSD*, 2002.

[67] Kasper Bilsted Graversen. The nature of roles. *A Taxonomic Analysis Ofroles As A Language Construct*, 2006.

[68] Giancarlo Guizzardi and Gerd Wagner. Conceptual simulation modeling with ontouml advanced tutorial. In *Proceedings of the 2012 Winter Simulation Conference (WSC)*, pages 1–15. IEEE, 2012.

[69] Giancarlo Guizzardi, Gerd Wagner, Nicola Guarino, and Marten van Sinderen. An ontologically well-founded profile for uml conceptual models. In *International Conference on Advanced Information Systems Engineering*, pages 112–126. Springer, 2004.

[70] Terry Halpin. Orm 2. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 676–687. Springer, 2005.

[71] Stefan Hanenberg and Rainer Unland. Roles and aspects : Similarities, differences, and synergetic potential. In *International Conference on Object-Oriented Information Systems*, pages 507–520. Springer, 2002.

[72] Simon Hansman and Ray Hunt. A taxonomy of network and computer attacks. *Computers & Security*, 24(1) :31–43, 2005.

[73] Simon Luke Hansman. A taxonomy of network and computer attack methodologies. 2003.

[74] Ke-Qing He, Jian Wang, and Peng Liang. Semantic interoperability aggregation in service requirements refinement. *Journal of computer science and technology*, 25(6) : 1103–1117, 2010.

[75] R Heckman. Attacker classification to aid targeting critical systems for threat modelling and security review (2005), 2015.

[76] Rolf Hennicker and Annabelle Klarl. Foundations for ensemble modeling–the helena approach. In *Specification, Algebra, and Software*, pages 359–381. Springer, 2014.

[77] Stephan Herrmann. Programming with roles in objectteams/java. In *proc. AAAI Fall Symposium*, 2005.

[78] Soichiro Hidaka, Massimo Tisi, Jordi Cabot, and Zhenjiang Hu. Feature-based classification of bidirectional transformation approaches. *Software & Systems Modeling*, 15(3) :907–928, 2016.

[79] Hannes Holm, Khurram Shahzad, Markus Buschle, and Mathias Ekstedt. P^2cysemol : Predictive, probabilistic cyber security modeling language. *IEEE Transactions on Dependable and Secure Computing*, 12(6) :626–639, 2014.

[80] Thomas J Holt and Max Kilger. Know your enemy : The social dynamics of hacking. *The Honeynet Project*, pages 1–17, 2012.

[81] Jin B Hong, Dong Seong Kim, Chun-Jen Chung, and Dijiang Huang. A survey on the usability and practical applications of graphical security models. *Computer Science Review*, 26 :1–16, 2017.

[82] Zhenjiang Hu, Shin-Cheng Mu, and Masato Takeichi. A programmable editor for developing structured documents based on bidirectional transformations. *Higher-Order and Symbolic Computation*, 21(1-2) :89–118, 2008.

[83] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of mde in industry. In *Proceedings of the 33rd international conference on software engineering*, pages 471–480, 2011.

[84] Vinay M Igure and Ronald D Williams. Taxonomies of attacks and vulnerabilities in computer systems. *IEEE Communications Surveys & Tutorials*, 10(1) :6–19, 2008.

[85] ISO ISO. 14258 : Industrial automation systems - concepts and rules for enterprise models. *International Organization for Standardization, Geneva, Switzerland*, 2000.

[86] ISO ISO. Iec/ieee systems and software engineering : Architecture description. *ISO/IEC/IEEE 42010 : 2011 (E)(Revision of ISO/IEC 42010 : 2007 and IEEE Std 1471-2000)*, 2011.

[87] Sushil Jajodia, Steven Noel, and Brian O'berry. Topological analysis of network attack vulnerability. In *Managing cyber threats*, pages 247–266. Springer, 2005.

[88] Tobias Jäkel, Thomas Kühn, Hannes Voigt, and Wolfgang Lehner. Rsql-a query language for dynamic data types. In *Proceedings of the 18th International Database Engineering & Applications Symposium*, pages 185–194, 2014.

[89] Tobias Jäkel, Thomas Kühn, Stefan Hinkel, Hannes Voigt, and Wolfgang Lehner. Relationships for dynamic data types in rsql. *Datenbanksysteme für Business, Technologie und Web (BTW 2015)*, 2015.

[90] Tobias Jäkel, Thomas Kühn, Hannes Voigt, and Wolfgang Lehner. Towards a role-based contextual database. In *East European Conference on Advances in Databases and Information Systems*, pages 89–103. Springer, 2016.

[91] Chanchala Joshi, Umesh Kumar Singh, and Kapil Tarey. A review on taxonomies of attacks and vulnerability in computer and network system. *International Journal*, 5(1), 2015.

[92] Nafiseh Kahani, Mojtaba Bagherzadeh, James R Cordy, Juergen Dingel, and Daniel Varró. Survey and classification of model transformation tools. *Software & Systems Modeling*, 18(4) :2361–2397, 2019.

[93] Apu Kapadia, Jalal Al-Muhtadi, Roy H Campbell, and Dennis Mickunas. Ir ac 2000 : Secure interoperability using dynamic role translation. university of illinois, dekalb, ill. 2000.

[94] Gerti Kappel, Werner Retschitzegger, and Wieland Schwinger. A comparison of role mechanisms in object-oriented modeling. In *Modellierung*, volume 98, pages 105–109. Citeseer, 1998.

[95] Günter Kniesel. Objects don't migrate ! perspectives on objects with roles. 1996.

[96] EVROPSKA KOMISIJA. Joint communication to the european parliament, the council, the european economic and social committee and the committee of the regions : Cybersecurity strategy of the europian union : An open, safe and secure cyberspace, 2013.

[97] Barbara Kordy, Ludovic Piètre-Cambacédès, and Patrick Schweitzer. Dag-based attack and defense modeling : Don't miss the forest for the attack trees. *Computer science review*, 13 :1–38, 2014.

[98] Bent Bruun Kristensen. Object-oriented modeling with roles. In *OOIS'95*, pages 57–71. Springer, 1996.

[99] Bent Bruun Kristensen and Kasper Østerbye. Roles : conceptual abstraction theory and practical language issues. *Theory and Practice of Object Systems*, 2(3) :143–160, 1996.

[100] Thomas Kühn. *A Family of Role-Based Languages*. PhD thesis, Technische Universität Dresden, 2017.

[101] Thomas Kühn, Max Leuthäuser, Sebastian Götz, Christoph Seidl, and Uwe Aßmann. A metamodel family for role-based modeling and programming languages. In *International Conference on Software Language Engineering*, pages 141–160. Springer, 2014.

[102] Thomas Kühn, Stephan Böhme, Ing Sebastian Götz, et al. A combined formal model for relational context-dependent roles (extended). 2015.

[103] Thomas Kühn, Stephan Böhme, Sebastian Götz, and Uwe Aßmann. A combined formal model for relational context-dependent roles. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, pages 113–124, 2015.

[104] Thomas Kühn, Kay Bierzynski, Sebastian Richly, and Uwe Aßmann. Framed : full-fledge role modeling editor (tool demo). In *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*, pages 132–136, 2016.

[105] Thomas Kühn, Kevin Ivo Kassin, Walter Cazzola, and Uwe Aßmann. Modular feature-oriented graphical editor product lines. In *Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1*, pages 76–86, 2018.

[106] Thomas Kühn, Christopher Werner, Hendrik Schön, Zhao Zhenxi, and Uwe Aßmann. Contextual and relational role-based modeling framework. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 442–449. IEEE, 2019.

[107] Harjinder Singh Lallie, Kurt Debattista, and Jay Bal. An empirical evaluation of the effectiveness of attack graphs and fault trees in cyber-attack perception. *IEEE Transactions on Information Forensics and Security*, 13(5) :1110–1122, 2017.

[108] Benoît Langlois, Daniel Exertier, and Boubekeur Zendagui. Development of modelling frameworks and viewpoints with kitalpha. In *Proceedings of the 14th Workshop on Domain-Specific Modeling*, pages 19–22, 2014.

[109] Elizabeth LeMay, Michael D Ford, Ken Keefe, William H Sanders, and Carol Muehrcke. Model-based security metrics using adversary view security evaluation (advise). In *2011 Eighth International Conference on Quantitative Evaluation of SysTems*, pages 191–200. IEEE, 2011.

[110] Aleksandr Lenin, Jan Willemson, and Dyan Permata Sari. Attacker profiling in quantitative security assessment based on attack trees. In *Nordic Conference on Secure IT Systems*, pages 199–212. Springer, 2014.

[111] David John Leversage and Eric James Byres. Comparing electronic battlefields : Using mean time-to-compromise as a comparative security metric. In *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 213–227. Springer, 2007.

[112] Mengchi Liu and Jie Hu. Information networking model. In *International Conference on Conceptual Modeling*, pages 131–144. Springer, 2009.

[113] Francis Lodwick. *A Common Writing*. Longman Publishing Group, The Works of Francis Lodwick : A study of his writings in the intellectual context of the seventeenth century (1972). Vivian Salmon, 1647.

[114] Frank Loebe. Abstract versus social roles—a refined top-level ontological analysis. 2005.

[115] Peri Loucopoulos and Roberto Zicari. *Conceptual modeling, databases, and CASE : an integrated view of information systems development*. John Wiley & Sons, Inc., 1992.

[116] Giridhar Manepalli. Federation of metadata registries. *Corporation for National Research Initiatives, Virginia, United States of America, gmanepalli@ cnri. reston. va. us, Retrieved September*, 12, 2016.

[117] Didonet Del Fabro Marcos, Bézivin Jean, Jouault Frédéric, Breton Erwan, and Gueltas Guillaume. Amw : A generic model weaver. *Proc. of the 1eres Journées sur l'Ingénierie Dirigée par les Modeles*, 200, 2005.

[118] David J Martin, Daniel Kifer, Ashwin Machanavajjhala, Johannes Gehrke, and Joseph Y Halpern. Worst-case background knowledge for privacy-preserving data publishing. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 126–135. IEEE, 2007.

[119] Kazutaka Matsuda, Zhenjiang Hu, Keisuke Nakano, Makoto Hamana, and Masato Takeichi. Bidirectionalization transformation based on automatic derivation of view complement functions. *ACM SIGPLAN Notices*, 42(9) :47–58, 2007.

[120] Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In *International Conference on Information Security and Cryptology*, pages 186–198. Springer, 2005.

[121] Lambert Meertens. Designing constraint maintainers for user interaction, 1998.

[122] Johannes Meier, Christopher Werner, Heiko Klare, Christian Tunjic, Uwe Aßmann, Colin Atkinson, Erik Burger, Ralf Reussner, and Andreas Winter. Classifying approaches for constructing single underlying models. In *International Conference on Model-Driven Engineering and Software Development*, pages 350–375. Springer, 2019.

[123] Hafedh Mili, Joumana Dargham, and Ali Mili. Views : A framework for feature-based development and distribution of oo applications. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 11–pp. IEEE, 2000.

[124] Pierre-Alain Muller and Nathalie Gaertner. *Modélisation objet avec UML*, volume 514. Eyrolles Paris, 2000.

[125] I Nai-Fovino, R Neisse, J. L Hernandez-Ramos, N Polemi, G Ruzzante, M Figwer, A Lazari, European Commission, and Joint Research Centre. *A proposal for a European cybersecurity taxonomy.* 2019. ISBN 9789276116035. doi : 10.2760/106002. URL https://op.europa.eu/publication/manifestation_identifier/PUB_KJNA29868ENN. OCLC : 1140141718.

[126] Mahmoud Nassar. *Analyse/conception par points de vue : le profil VUML*. PhD thesis, 2005.

[127] Mahmoud Nassar, Bernard Coulette, Xavier Crégut, Sophie Ebersold, and Abdelaziz Kriouile. Towards a view based unified modeling language. *ICEIS (3)*, 2003 :257–265, 2003.

[128] Jörg Niemöller, Leonid Mokrushin, Konstantinos Vandikas, Stefan Avesand, and Lars Angelin. Model federation and probabilistic analysis for advanced oss and bss. In *2013 Seventh International Conference on Next Generation Mobile Apps, Services and Technologies*, pages 122–129. IEEE, 2013.

[129] Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. Viewpoints : meaningful relationships are difficult ! In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 676–681. IEEE, 2003.

[130] Janis Osis and Erika Asnina. *Model-driven domain analysis and software development : architectures and functions*. Information Science Reference, 2011.

[131] Harold Ossher, Matthew Kaplan, William Harrison, Alexander Katz, and Vincent Kruskal. Subject-oriented composition rules. In *Proceedings of the tenth annual*

conference on Object-oriented programming systems, languages, and applications, pages 235–250, 1995.

[132] Xinming Ou, Wayne F Boyer, and Miles A McQueen. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345, 2006.

[133] Mike P Papazoglou and Bernd J Krämer. Modeling object dynamics., 2000.

[134] N Paulauskas and E Garsva. Attacker skill level distribution estimation in the system mean time-to-compromise. In *2008 1st International Conference on Information Technology*, pages 1–4. IEEE, 2008.

[135] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79, 1998.

[136] Trygve Reenskaug and James O Coplien. The dci architecture : A new vision of object-oriented programming. *An article starting a new blog :(14pp) http :/ /www. artima. com/articles/dci_vision. html*, 2009.

[137] Trygve Reenskaug, Per Wold, Odd Arild Lehne, et al. *Working with objects : the OOram software engineering method*. Manning Greenwich, 1996.

[138] Joel Richardson and Peter Schwarz. Aspects : Extending objects to support multiple, independent roles. In *Proceedings of the 1991 ACM SIGMOD international conference on Management of data*, pages 298–307, 1991.

[139] Dirk Riehle. *Framework design : A role modeling approach*. PhD thesis, ETH Zurich, 2000.

[140] Colette Rolland, Naveen Prakash, and Adolphe Benjamen. A multi-model view of process modelling. *Requirements engineering*, 4(4) :169–187, 1999.

[141] Andrew P Sage and Christopher D Cuppan. On the systems engineering and management of systems of systems and federations of systems. *Information knowledge systems management*, 2(4) :325–345, 2001.

[142] Jean-Philippe Schneider, Joël Champeau, Ciprian Teodorov, Eric Senn, and Loïc Lagadec. A role language to interpret multi-formalism system of systems models. In *2015 Annual IEEE Systems Conference (SysCon) Proceedings*, pages 200–205. IEEE, 2015.

[143] Mirko Seifert, Christian Wende, and Uwe Aßmann. Anticipating unanticipated tool interoperability using role models. In *Proceedings of the First International Workshop on Model-Driven Interoperability*, pages 52–60, 2010.

[144] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 273–284. IEEE, 2002.

[145] JJ Shiling and Peter F Sweeney. Three steps to views : extending the object-oriented paradigm. *ACM SIGPLAN Notices*, 24(10) :353–361, 1989.

[146] Adam Shostack. *Threat modeling : Designing for security*. John Wiley & Sons, 2014.

[147] Freddy Kamdem Simo. *Model-based federation of systems of modelling*. PhD thesis, 2017.

[148] Umesh Kumar Singh and Chanchala Joshi. Quantifying security risk by critical network vulnerabilities assessment. *International Journal of Computer Applications*, 156(13) :26–33, 2016.

[149] Umesh Kumar Singh, Chanchala Joshi, and Neha Gaud. Information security assessment by quantifying risk level of network vulnerabilities. *International Journal of Computer Applications*, 156(2) :37–44, 2016.

[150] Lance Spitzner. Know your enemy : The tools and methodologies of the script kiddie. *The Honeynet Project*, 2000.

[151] Maarten WA Steen, David H Akehurst, Hugo WL ter Doest, and Marc M Lankhorst. Supporting viewpoint-oriented enterprise architecture. In *Proceedings. Eighth IEEE International Enterprise Distributed Object Computing Conference, 2004. EDOC 2004.*, pages 201–211. IEEE, 2004.

[152] Friedrich Steimann. Formale modellierung mit rollen. *Habilitation, Faculty of Electrical Engineering and Information Technology, University of Hannover (in German)*, page 74, 2000.

[153] Friedrich Steimann. A radical revision of uml's role concept. In *International Conference on the Unified Modeling Language*, pages 194–209. Springer, 2000.

[154] Friedrich Steimann. On the representation of roles in object-oriented and conceptual modelling. *Data & Knowledge Engineering*, 35(1) :83–106, 2000.

[155] Friedrich Steimann. The role data model revisited. *Applied Ontology*, 2(2) :89–103, 2007.

[156] Perdita Stevens. Bidirectional model transformations in qvt : semantic issues and open questions. *Software & Systems Modeling*, 9(1) :7, 2010.

[157] Tithnara Sun, Bastien Drouot, Fahad Golra, Joël Champeau, Sylvain Guerin, Luka Le Roux, Raúl Mazo, Ciprian Teodorov, Lionel Aertryck, and Bernard Hostis. A domain-specific modeling framework for attack surface modeling. In *International Conference on Information Systems Security and Privacy*, 2020.

[158] Tetsuo Tamai, Naoyasu Ubayashi, and Ryoichi Ichiyama. An adaptive object model with dynamic role binding. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pages 166–175. IEEE, 2005.

[159] Anshu Tripathi and Umesh Kumar Singh. Towards standardization of vulnerability taxonomy. In *2010 2nd International Conference on Computer Technology and Development*, pages 379–384. IEEE, 2010.

[160] Tony UcedaVelez and Marco M Morana. *Risk centric threat modeling*. Wiley Online Library, 2015.

[161] Zoltán Ujhelyi, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, Benedek Izsó, István Ráth, Zoltán Szatmári, and Dániel Varró. Emf-incquery : An integrated development environment for live model queries. *Science of Computer Programming*, 98 :80–99, 2015.

[162] Hoang Van Tran, Hiep Xuan Huynh, Vinh Cong Phan, and Bernard Pottier. A federation of simulations based on cellular automata in cyber-physical systems. *EAI Endorsed Trans. Context-aware Syst. & Appl.*, 3(7) :e3, 2016.

[163] Hein S Venter, Jan HP Eloff, and YL Li. Standardising vulnerability categories. *Computers & Security*, 27(3-4) :71–83, 2008.

[164] Hironori Washizaki, Tian Xia, Natsumi Kamata, Yoshiaki Fukazawa, Hideyuki Kanuka, Dan Yamaoto, Masayuki Yoshino, Takao Okubo, Shinpei Ogata, Haruhiko Kaiya, et al. Taxonomy and literature survey of security pattern research. In *2018 IEEE Conference on Application, Information and Network Security (AINS)*, pages 87–92. IEEE, 2018.

[165] Christopher Werner and Uwe Aßmann. Model synchronization with the role-oriented single underlying model. In *MODELS Workshops*, pages 62–71, 2018.

[166] Christopher Werner, Hendrik Schön, Thomas Kühn, Sebastian Götz, and Uwe Aßmann. Role-based runtime model synchronization. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 306–313. IEEE, 2018.

[167] Christopher Werner, Manuel Wimmer, and Uwe Aßmann. A generic language for query and viewtype generation by-example. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 379–386. IEEE, 2019.

[168] Zhang Yongzheng and Yun Xiaochun. A new vulnerability taxonomy based on privilege escalation. In *Proceedings of the 6th International Conference on Enterprise Information Systems*, 2004.

[169] Weiqing Zhang and Birger Møller-Pedersen. Tool integration models. In *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, volume 1, pages 485–494. IEEE, 2013.

[170] Weiqing Zhang and Birger Møller-Pedersen. Modeling of tool integration resources with oslc support. In *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 99–110. IEEE, 2014.

[171] Weiqing Zhang, Vincent Leilde, Birger Møller-Pedersen, Joel Champeau, and Christophe Guychard. Towards tool integration through artifacts and roles. In *2012 19th Asia-Pacific Software Engineering Conference*, volume 1, pages 603–613. IEEE, 2012.

Annexe A

Impacts des types de vulnérabilité CVE

Dans la section 2.1.2, nous présentons les 13 types de vulnérabilité utilisés par CVE [1] pour catégoriser les vulnérabilités, à savoir :

— dénis de service (*DoS*).
— exécution de code (*Code Execution*).
— débordement de tampon (*Overflow*).
— corruption de mémoire (*Memory Corruption*).
— injection SQL (*Sql Injection*).
— script intersite (*XSS*).
— traversement de répertoires (*Directory Traversal*).
— fractionnement d'une réponse HTTP (*Http Response Splitting*).
— contournement (*Bypass something*).
— obtention d'informations (*Gain Information*).
— obtention de privilèges (*Gain Privileges*).
— contrefaçon de demande intersite (*CSRF*).
— inclusion de fichier (*File Inclusion*) .

Dans la section 3.3.2, nous présentons 8 types d'impacts utilisés pour modéliser les préoccupations d'un attaquant liées à l'exploitation d'une vulnérabilité, à savoir :

— Gain d'accès
— Contournement
— Dénis de service
— Élévation de privilège
— Détournement de session
— Exécution de code
— Vol d'informations
— Altération

Dans le tableau A.1, nous identifions pour chaque type de vulnérabilité de CVE les impacts les plus courants. Notre catégorisation en impacts étant inspirée de celle en types, il existe pour certains types une correspondance parfaite avec un impact, par exemple le type CVE *DoS* correspond à l'impact "Dénis de service". De même, à certains types CVE correspondent différents impacts, tels que le type CVE *XSS*.

	Gain d'accès	Contournement	Dénis de service	Élévation de privilège	Détournement de session	Exécution de code	Vol d'informations	Altération
DoS			x					
Code Execution						x		
Overflow	x			x				x
Memory Corruption			x			x		x
Sql Injection				x		x		x
XSS	x			x	x	x	x	
Directory Traversal							x	x
Http Response Splitting	x					x		x
Bypass something		x						
Gain Information							x	
Gain Privileges				x				
CSRF					x			x
File Inclusion								x

Tableau A.1 – Impacts des différents types de vulnérabilité recensés par CVE dans [2]

Annexe B

Métamodèle complet de Role4All

N	Caractéritique (Niveau conceptuel)	Role4All
1	Roles have its own properties and behaviors (M0, M1)	●
2	Roles depend on relationships (M0, M1)	○
3	Objects may play different roles simultaneously (M0 M1)	●
4	Objects may play the same role several times, simultaneously (M0)	●
5	Objects may acquire and abandon roles dynamically (M0)	●
6	The sequence of role acquisition and removal may be restricted (M0, M1)	◐
7	Objects of unrelated types can play the same role (M1)	●
8	Roles can play roles (M0, M1)	●
9	A role can be transferred from one object to another (M0)	●
10	The state of an object can be role-specific (M0)	●
11	Features of an object can be role-specific (M1)	●
12	Roles restrict access (M0)	●
13	Different roles may share structure and behavior (M1)	●
14	An object and its roles share identity (M0)	○
15	An object and its roles have different identities (M0)	●
16	Relationships between roles can be constrained (M1)	◐
17	There may be constraints between relationships (M1)	○
18	Roles can be grouped and constrained together (M1)	○
19	Roles depend on compartments (M0, M1)	◐
20	Compartments have properties and behaviors (M0, M1)	●
21	A role can be part of several compartments (M0, M1)	●
22	Compartments may play roles like objects (M0, M1)	○
23	Compartments may play roles which are part of themselves (M0, M1)	○
24	Compartments can contain other compartments (M0, M1)	○
25	Different compartments may share structure and behavior (M1)	●
26	Compartments have their own identity (M0)	●
27	The number of roles occurring in a compartment can be constrained (M1)	○

○ : Non couverte ◐ : Partiellement couverte ● : Couverte
M0 : Relative aux types (rôle, *player*, etc.)
M1 : Relative aux instances (*roleInstance*, *playerInstance*, etc.)

Tableau B.1 – Ensemble des caractéristiques couvertes par Role4All

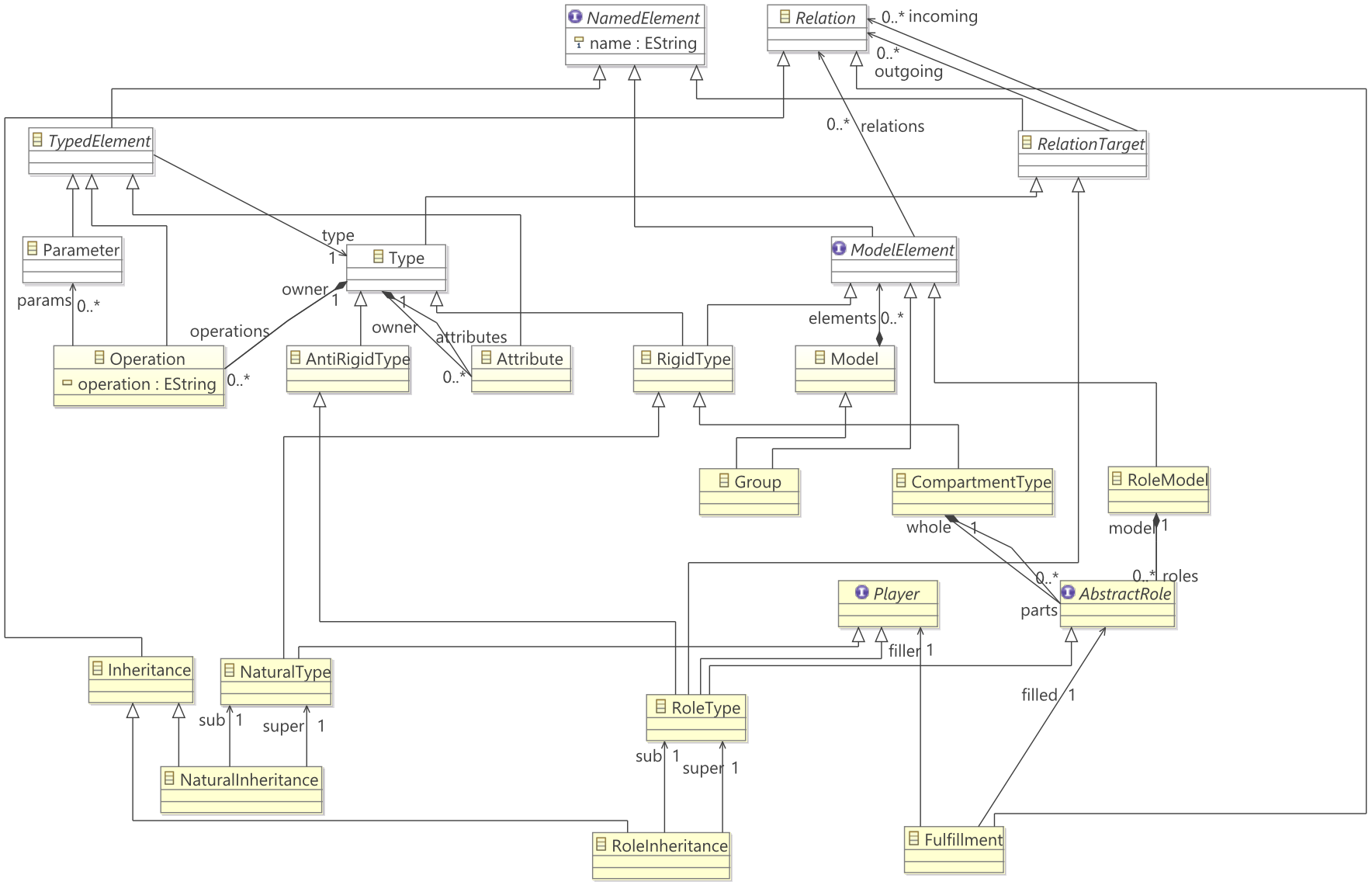


FIGURE B.1 – Métamodèle de Role4All généré par *FRaMED* [104]

Annexe C

Calcul de nombre de combinaisons entre *roleInstance* et *playerInstance*

Dans le chapitre 4, nous affirmons que pour x *roleInstances* et y *playerInstances* il existe :

— Cas 1 : $x * (2^y - 1)$ adaptations différentes dans le cas où un *roleInstance* peut être lié à différents *playerInstance* à la fois

— Cas 2 : $x * y$ dans le cas où un *roleInstance* n'est lié qu'à un seul *playerInstance*

Par la suite nous noterons $R = (r_1, r_2, ..., r_x)$ l'ensemble des *roleInstances* et $P = (p_1, p_2, ..., p_y)$ l'ensemble des *playerInstances*. On a alors $size(R) = x$ et $size(P) = y$.

C.1 Explication du cas 1

Dans ce cas, on a un *roleInstance* qui peut être lié à un ou plusieurs *playerInstances*. Les combinaisons possibles sont donc composées d'un *roleInstance* et d'au moins un *playerInstance*. De plus, dans notre cas, il n'existe pas de relation d'ordre c'est-à-dire $(p1, p2) = (p2, p1)$. Finalement, un *roleInstance* peut être lié à tout sous-ensemble non vide de l'ensemble des *playerIntances*. En notant $\mathbb{P}(\mathcal{Y})$ l'ensemble des parties de l'ensemble $\mathcal{Y}$ on a pour chaque *roleInstance* $size(\mathbb{P}(P)) - 1$ combinaisons possibles soit $2^{size(P)} - 1 = 2^y - 1$ adaptations différentes pour chaque *roleInstance*.

Pour l'ensemble des *roleInstances* on a donc $size(R) * (2^y - 1) = x * (2^y - 1)$ adaptations différentes possibles.

C.2 Explication du cas 2

Dans ce cas, un *roleInstance* est lié à un seul *playerInstance*. Clairement, le nombre de combinaisons possibles est $size(R) * size(P)$, on a donc $x * y$ adaptations différentes possibles.

Annexe D

Conséquences du support des caractéristiques de vue et de point de vue d'attaquants par Role4All

Caractéristiques de vue et point de vue	**Conséquences**
① Multiple Metamodel Arity	Support des caractéristiques de rôle 1 et 7.
② Viewtype Manifestation Materialized	Support de la caractéristique de rôle 1.
③ Viewtype language Opération Add, Modify and Filter	Support de la caractéristique de rôle 13. Besoin du concept d'*Organasier*
④ Viewtype language Paradigm Extentional	Besoin d'*adaptateur*
⑤ Intrisiveness Non-Intrusive	Support des caractéristiques de rôle 5 et 15.
⑥ Runetime Consistency Direction Model to View and View to Model	Support des caractéristiques de rôle 10 et 11. Besoin d'*adaptateur*
⑦ Multiple Model Arity	Besoin de la relation de contenance
⑧ Closedness Augmenting and Query Language Opération Selection, Projection, Join, Aggregate and Rename	Support des caractéristiques de rôle 10, 11 et 12.
⑨ Query language Paradigm Intensional Explicit	Support de la caractéristique de rôle 16 Besoin des concepts d'*AssociationRule* et de *Contexte*
⑩ View Manifestation Virtual	Besoin de *playRelation* et d'*adaptateur*
⑪ Runetime Consistency Recomputation Incremental and Frequency Immediat	Supporter car support de ⑩

Tableau D.1 – Conséquence sur Role4All du support des caractéristiques de vue et de point de vue d'attaquants

Annexe E

Scénario d'attaque du système fil rouge

Nous présentons dans la section 3.2, l'exemple fil rouge et nous détaillons les différents éléments le constituant au cours du chapitre 3. Nous présentons dans cette annexe, en figure E.1, le scénario d'attaque utilisé par Mallory afin de voler les informations détenues par Alice. Sur cette figure, les flèches pleines représentent l'exécution de *behaviorBlocks* et celles en pointillés, l'association d'un nouveau rôle ou la modification d'un player. Les blocs gris clair symbolisent des *machines* ou des *logiciels*, ceux gris foncé, des *machines vulnérables* ou des *logiciels vulnérables* et les noirs des *machines contrôlées*.

Comme présenté dans la section 3.2, le scénario d'attaque est découpé en quatre étapes :

e0 | Situation initiale : Nous initions l'attaque par l'exécution du *behaviorBlock* 'Start Attack', nous représentons alors Mallory par une *machine controlée*. Alice et Bob sont allumés (*aliceIsTurnOn* := *true* et *bobIsTurnOn* := *true*) et Carol éteinte (*caroleIsTurnOn* := *false*).

e1 | Connexion au système :

- e1.1 | Découverte du système : Mallory exécute le *behaviorBlock* 'SysDiscovery'. Dans le système fil rouge, Mallory est connecté à Alice, Bob et Carol. Cependant Carol n'étant pas allumée, l'attaque ne découvre que les *machines* représentant Alice et Bob.
- e1.2 | Exploration du système : Mallory exécute le *behaviorBlock* 'Scan' sur Alice et sur Bob. L'attaquant découvre alors, grâce à ses accès et droits sur les différentes *machines*, une partie des applications utilisées par Alice et Bob. Nous représentons chaque application découverte via une instance du rôle *Logiciel*. Par souci de concision, seul le *logiciel* 'Symposium' est représenté ici.

e2 | Prise de contrôle :

- e2.1 | Entrée dans le système : Mallory exécute le *behaviorBlock* 'ScanVuln' sur Symposium. Ce *logiciel* possède une vulnérabilité exploitable par Mallory, nous lui associons alors le rôle de *Logiciel Vulnérable* et, par effet de bord, la *machine* représentant Bob devient une *machine vulnérable*. Mallory exécute ensuite le *behaviorBlock* 'ExecuteVuln' sur Symposium, l'exploitation de la vulnérabilité présente sur ce *logiciel vulnérable* permet à Mallory d'obtenir des droits utilisateur sur Bob.
- e2.2 | Élévation de privilège : Mallory exécute de nouveau le *behaviorBlock* 'Scan'. Grâce aux nouveaux droits obtenus, l'attaquant découvre le système d'exploitation utilisé par Bob représenté par le *logiciel* 'Ubuntu'. Mallory exécute ensuite *behaviorBlock* 'ScanVuln' sur Ubuntu. Ce *logiciel* possède une vulnérabilité

exploitable par Mallory, nous lui associons alors le rôle de *Logiciel Vulnérable*. Mallory exécute enfin le *behaviorBlock* 'ExecuteVuln' sur le *logiciel vulnérable*, Ubuntu lui permettant d'obtenir des droits administrateur sur Bob. Par conséquent, la *machine vulnérable* représentant Bob devient une *machine contrôlée*.

e3| Récupération d'informations :

e3.1| Pivotage et découverte du réseau : Mallory via la *machine contrôlée* Bob exécute le *behaviorBlock* 'SysDiscovery'. Il découvre alors que Bob est connecté à la *machine* Alice, Carol étant toujours éteinte, l'attaquant ne découvre pas son existence.

e3.2| Exploration du réseau : Mallory via Bob exécute le *behaviorBlock* 'Scan' sur Alice. Il découvre le système d'exploitation utilisé par cette dernière, nous le représentons ici par le *logiciel* 'Windows 10'.

e3.3| Collecte de données : Mallory via Bob exécute le *behaviorBlock* 'ScanVuln' sur le *logiciel* d'Alice. Ce dernier contient une vulnérabilité exploitable par l'attaquant, pour représenter cette découverte, nous associons le rôle de *Logiciel Vulnérable* à Windows 10 et Alice celui de *Machine Vulnérable* à Alice. Par la suite, Mallory, via Bob, exécute le *behaviorBlock* 'ExecuteVuln' sur le *logiciel vulnérable* d'Alice. L'exploitation de la vulnérabilité de Windows 10 permet le vol des informations contenues par Alice. Mallory a alors obtenu ce qu'il cherchait, nous stoppons alors l'attaque avec l'exécution du *behaviorBlock* 'Stop Attack'.

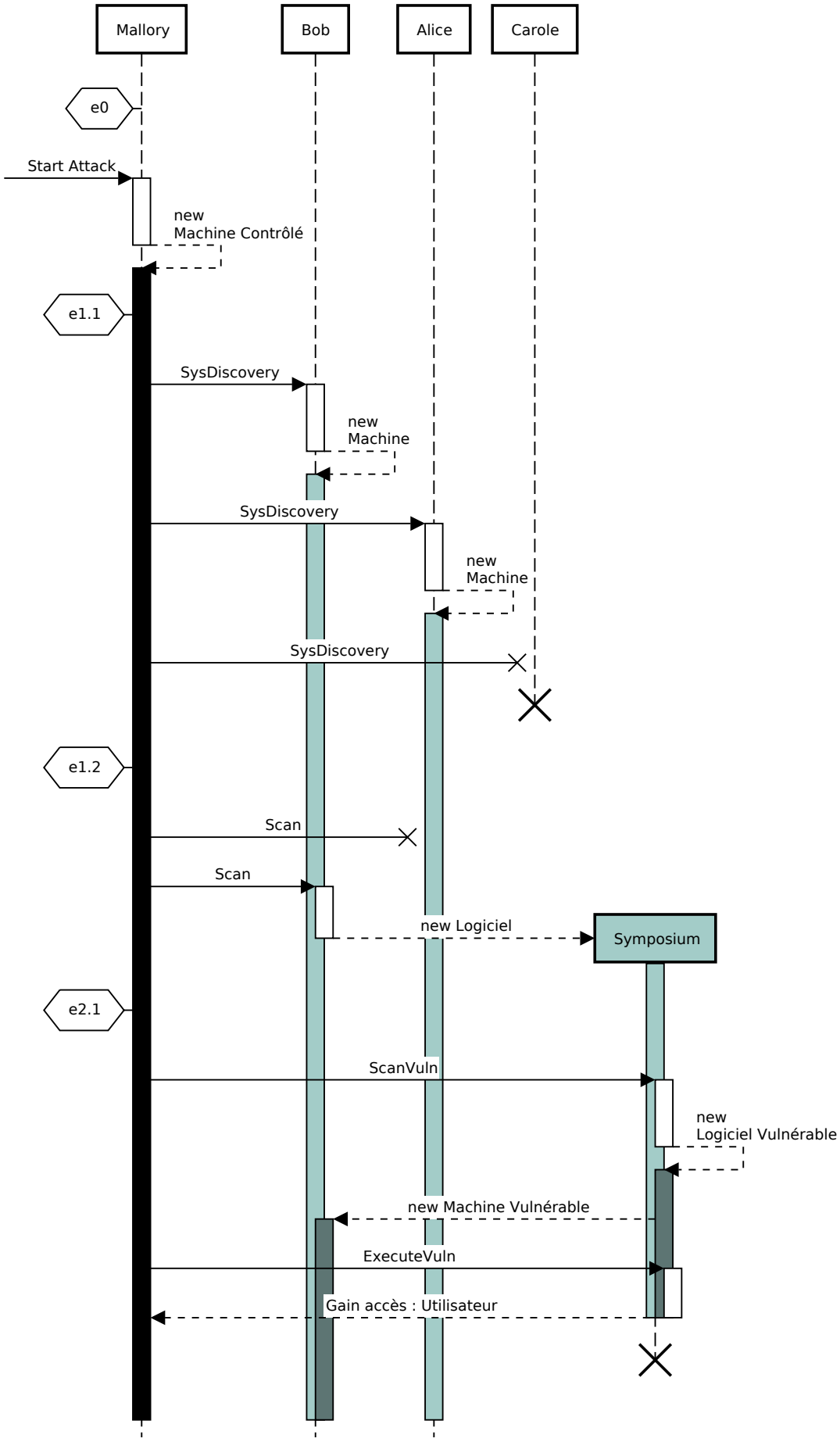


FIGURE E.1 – Diagramme de séquence représentant l'attaque de Mallory sur le système fil rouge

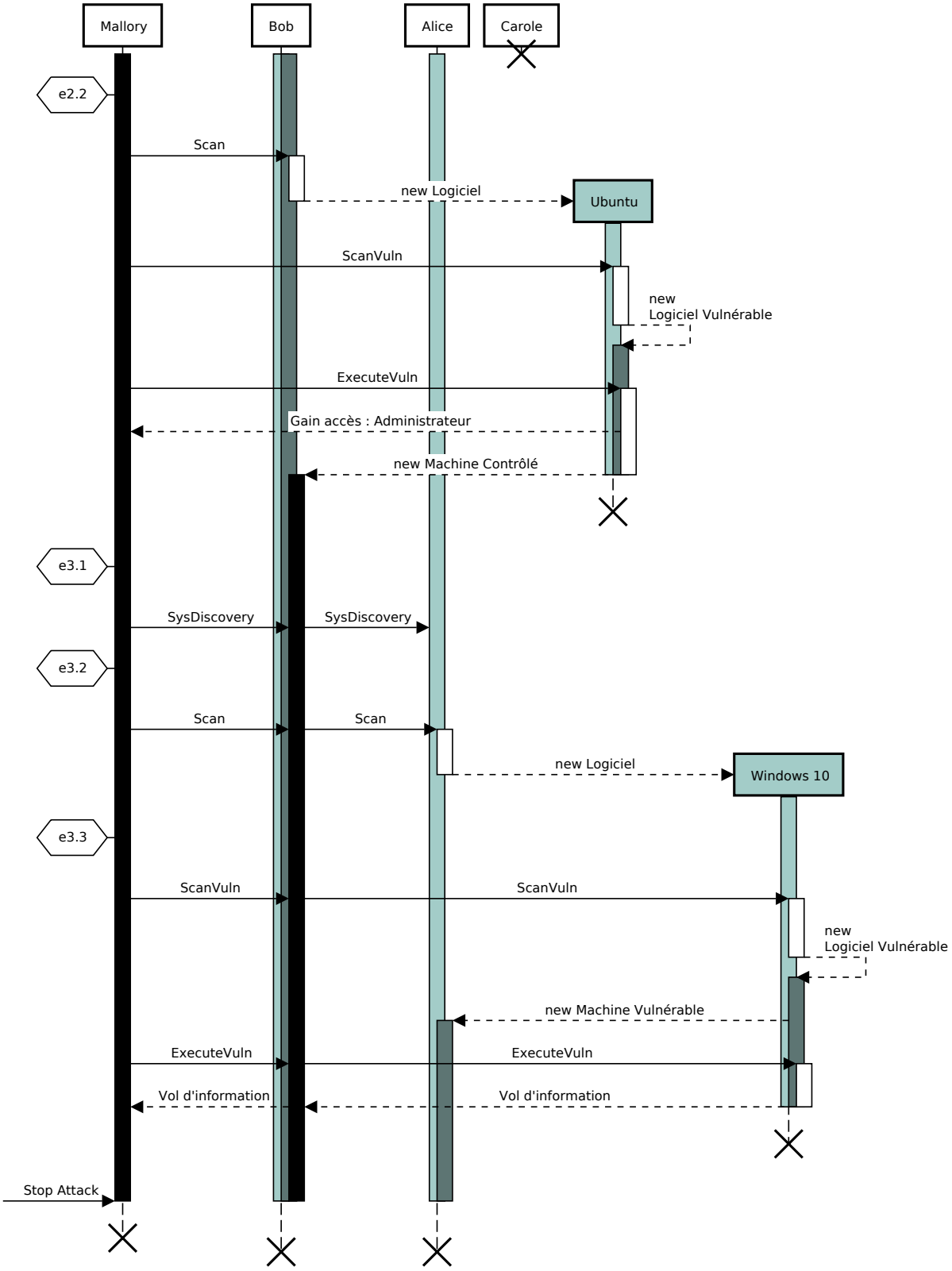


FIGURE E.1 – Diagramme de séquence représentant l'attaque de Mallory sur le système fil rouge

Annexe F

Schéma JSON utilisé pour représenter les vulnérabilités dans NVD

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "JSON Schema for NVD Vulnerability Data Feed version 1.1
     ",
  "id": "https://scap.nist.gov/schema/nvd/feed/1.1/
     nvd_cve_feed_json_1.1.schema",
  "definitions": {
    "def_cpe_name": {
      "description": "CPE name",
      "type": "object",
      "properties": {
        "cpe22Uri": {
          "type": "string"
        },
        "cpe23Uri": {
          "type": "string"
        },
        "lastModifiedDate": {
          "type": "string"
        }
      },
      "required": [
      "cpe23Uri"
      ]
    },
    "def_cpe_match": {
      "description": "CPE match string or range",
      "type": "object",
      "properties": {
        "vulnerable": {
          "type": "boolean"
        },
```

```
        "cpe22Uri": {
          "type": "string"
        },
        "cpe23Uri": {
          "type": "string"
        },
        "versionStartExcluding": {
          "type": "string"
        },
        "versionStartIncluding": {
          "type": "string"
        },
        "versionEndExcluding": {
          "type": "string"
        },
        "versionEndIncluding": {
          "type": "string"
        },
        "cpe_name": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/def_cpe_name"
          }
        }
      },
      "required": [
      "vulnerable",
      "cpe23Uri"
      ]
    },
    "def_node": {
      "description": "Defines a node or sub-node in an NVD
         applicability statement.",
      "properties": {
        "operator": {"type": "string"},
        "negate": {"type": "boolean"},
        "children": {
          "type": "array",
          "items": {"$ref": "#/definitions/def_node"}
        },
        "cpe_match": {
          "type": "array",
          "items": {"$ref": "#/definitions/def_cpe_match"}
        }
      }
    },
    "def_configurations": {
      "description": "Defines the set of product configurations for
          a NVD applicability statement.",
      "properties": {
```

```
        "CVE_data_version": {"type": "string"},
        "nodes": {
          "type": "array",
          "items": {"$ref": "#/definitions/def_node"}
        }
      },
      "required": [
      "CVE_data_version"
      ]
    },
    "def_subscore": {
      "description": "CVSS subscore.",
      "type": "number",
      "minimum": 0,
      "maximum": 10
    },
    "def_impact": {
      "description": "Impact scores for a vulnerability as found on
          NVD.",
      "type": "object",
      "properties": {
        "baseMetricV3": {
          "description": "CVSS V3.x score.",
          "type": "object",
          "properties": {
            "cvssV3": {"$ref": "cvss-v3.x.json"},
            "exploitabilityScore": {"$ref": "#/definitions/
               def_subscore"},
            "impactScore": {"$ref": "#/definitions/def_subscore"}
          }
        },
        "baseMetricV2": {
          "description": "CVSS V2.0 score.",
          "type": "object",
          "properties": {
            "cvssV2": {"$ref": "cvss-v2.0.json"},
            "severity": {"type": "string"},
            "exploitabilityScore": {"$ref": "#/definitions/
               def_subscore"},
            "impactScore": {"$ref": "#/definitions/def_subscore"},
            "acInsufInfo": {"type": "boolean"},
            "obtainAllPrivilege": {"type": "boolean"},
            "obtainUserPrivilege": {"type": "boolean"},
            "obtainOtherPrivilege": {"type": "boolean"},
            "userInteractionRequired": {"type": "boolean"}
          }
        }
      }
    },
    "def_cve_item": {
```

```
      "description": "Defines a vulnerability in the NVD data feed
         .",
      "properties": {
        "cve": {"$ref": "CVE_JSON_4.0_min_1.1.schema"},
        "configurations": {"$ref": "#/definitions/
           def_configurations"},
        "impact": {"$ref": "#/definitions/def_impact"},
        "publishedDate": {"type": "string"},
        "lastModifiedDate": {"type": "string"}
      },
      "required": ["cve"]
    }
  },
  "type": "object",
  "properties": {
    "CVE_data_type": {"type": "string"},
    "CVE_data_format": {"type": "string"},
    "CVE_data_version": {"type": "string"},
    "CVE_data_numberOfCVEs": {
      "description": "NVD adds number of CVE in this feed",
      "type": "string"
    },
    "CVE_data_timestamp": {
      "description": "NVD adds feed date timestamp",
      "type": "string"
    },
    "CVE_Items": {
      "description": "NVD feed array of CVE",
      "type": "array",
      "items": {"$ref": "#/definitions/def_cve_item"}
    }
  },
  "required": [
  "CVE_data_type",
  "CVE_data_format",
  "CVE_data_version",
  "CVE_Items"
  ]
}
```

Title: Models' federation for cybersecurity analysis from an attacker's viewpoint.

Keywords: Cybersecurity, Modeling, Viewpoint, Model Federation

Abstract:

Cybersecurity encompasses multiple problem areas. One of those, "Cyber Threat Analysis" (CTA), intersects various domains, system modeling, attacker modeling and threat description, where each one evolves independently. CTA is covered by multiple approaches, such as penetration testing or fault injection testing. To consider a-priori analysis methods, models are required. In this manuscript, we focus on the attacker viewpoint modeling approachs.
Abundance of cybersecurity's standards must be taken into account in our work. Indeed, using various Domain Specific Modeling Languages (DSMLs), the CTA's experts describe their models in a language relevant to their problem area.

In this modeling context, we must consider several open research questions. Firstly, how to address the interoperability problem between heterogeneous DSMLs. Secondly, how to handle, in our modeling context, the evolving nature of CTA's methods and knowledge due to the effervescence of the research domain.

To tackle these issues, in this manuscript we propose a role-based federation approach to create dynamic and adaptive attacker's viewpoints linked to existing source models.

First, we define and model the attacker's viewpoint in a CTA context. Based on cybersecurity standards as source models, our definition covers all attacker's concerns in this scope. To take into account the necessity to adapt and to evolve this definition, we propose a modeling framework based on several viewpoints representing different concerns. Additionaly, we construct viewpoints on viewpoints in order to represent specific concerns or interpret specific attacker behavior.

Then, we present our modeling language to express viewpoints, named Role4All. The concept of role is central to our approach, allowing us to represent and to federate various DSMLs in order to generate viewpoints. Interoperability between the domain models is crucial to apply dedicated algorithms and interpretations on the attacker's viewpoints. In our work, dedicated viewpoints are used by security analysis tools to simulate attacks on a system.

A case study on development and on interpretation of attacker's viewpoint validates our approach. Our case study is based on the federation of multiple domain models describing, as an example, vulnerability definition, architectures description or configuration management. In particular, we highlight the use of roles to dynamically adapt attacker's perception on the attacked system.