Spécification et analyse formelles des politiques de sécurité dans un processus de courtage de l'informatique en nuage

Asma Guesmi

▶ **To cite this version:**

Asma Guesmi. Spécification et analyse formelles des politiques de sécurité dans un processus de courtage de l'informatique en nuage. Cryptographie et sécurité [cs.CR]. Université d'Orléans, 2016. Français. NNT : 2016ORLE2010 . tel-01431183

HAL Id: tel-01431183
https://theses.hal.science/tel-01431183

Submitted on 10 Jan 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ D'ORLÉANS

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

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLÉANS

THÈSE présentée par :

Asma GUESMI

soutenue le : **01 Juillet 2016**

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

Discipline/ Spécialité : **Informatique**

Spécification et analyse formelles des politiques de sécurité dans un processus de courtage de l'informatique en nuage

THÈSE dirigée par :

Pascal BERTHOMÉ	Professeur des universités, INSA Centre Val de Loire
Frédéric LOULERGUE	Professeur des universités, Université d'Orléans
Patrice CLEMENTE	Maître de conférences, INSA Centre Val de Loire

RAPPORTEURS :

Hervé GUYENNET	Professeur des universités, Université de Franche-Comté
Fabrice MOURLIN	Maître de conférences HDR, Université Paris-Est Créteil

---

JURY :

Maryline LAURENT	Professeur des universités, Institut Mines-Télécom,
Hervé GUYENNET	Professeur des universités, Université de Franche-Comté
Fabrice MOURLIN	Maître de conférences HDR, Université Paris-Est Créteil
Pascal BERTHOMÉ	Professeur des universités, INSA Centre Val de Loire
Frédéric LOULERGUE	Professeur des universités, Université d'Orléans
Patrice CLEMENTE	Maître de conférences, INSA Centre Val de Loire

Remerciements

En premier lieu, je remercie cordialement M. Hervé Guyennet et M. Fabrice Mourlin d'avoir accepté d'être rapporteurs de cette thèse ainsi que Mme. Maryline Laurent d'avoir fait l'honneur d'examiner cette thèse.

Je tiens à exprimer mes remerciements à mes directeurs et encadrant de thèse : M. Pascal Berthomé, M. Frédéric Loulergue et M. Patrice Clemente, pour m'avoir donné l'opportunité d'effectuer cette thèse et pour leur disponibilité et leurs conseils.

J'adresse mes plus profondes reconnaissances à tous les membres de l'équipe SDS du LIFO pour leur accueil et leur sympathie pendant ces trois ans de thèse, merci particulièrement à Aline, Ahmad et Jérémy.

Enfin, je tiens à remercier mes amis et surtout ma famille pour leur soutien et leurs encouragements.

Table des matières

Table des matières V

Liste des figures VIII

1 Introduction générale 1
- 1.1 Contexte . . . 1
- 1.2 Contributions . . . 3
- 1.3 Organisation du manuscrit . . . 6

2 Préliminaires 9
- 2.1 Introduction . . . 9
- 2.2 Alloy . . . 10
  - 2.2.1 Modèle Alloy . . . 10
  - 2.2.2 Langage Alloy . . . 10
  - 2.2.3 Analyseur Alloy . . . 10
  - 2.2.4 Structure de Alloy . . . 11
  - 2.2.5 Opérateurs . . . 14
  - 2.2.6 Exemple . . . 15
- 2.3 Conclusion . . . 18

3 État de l'art 21
- 3.1 Introduction . . . 21
- 3.2 Informatique en nuage . . . 22
  - 3.2.1 Définition générale . . . 22
  - 3.2.2 Caractéristiques essentielles . . . 22
  - 3.2.3 Modèles de service . . . 23
  - 3.2.4 Modèles de déploiement . . . 24
- 3.3 Courtage nuagique . . . 24
  - 3.3.1 Définition . . . 25
  - 3.3.2 Quelques solutions existantes . . . 25
  - 3.3.3 Synthèse . . . 27
- 3.4 Sécurité dans l'informatique en nuage . . . 27
  - 3.4.1 Les menaces de la sécurité de l'informatique en nuage . . . 28
  - 3.4.2 Solutions de sécurité dans l'informatique en nuage . . . 30

3.4.3 Cloud Security Alliance . . . 36
3.4.4 Courtage nuagique basé sur la sécurité . . . 38
3.5 Assurance de la sécurité au moyen des méthodes formelles . . . 41
3.6 Analyse et synthèse . . . 42
3.7 Conclusion . . . 43

4 Spécification des besoins du client et offres des fournisseurs 45
4.1 Introduction . . . 45
4.2 Offres des fournisseurs . . . 46
4.3 Besoins du client . . . 48
4.3.1 Besoins fonctionnels . . . 48
4.3.2 Besoins non-fonctionnels . . . 48
4.4 Spécifications en langage Alloy . . . 50
4.4.1 Offres des fournisseurs . . . 52
4.4.2 Besoins du client . . . 56
4.5 Détection des inconsistances coté client . . . 63
4.5.1 Conflits des spécifications fonctionnelles . . . 64
4.5.2 Conflits des spécifications non-fonctionnelles . . . 65
4.6 Conclusion . . . 74

5 Placement 75
5.1 Introduction . . . 75
5.2 Mise en correspondance des besoins et des offres . . . 76
5.3 Mise en correspondance des besoins fonctionnels . . . 78
5.3.1 Recherche des correspondances . . . 79
5.3.2 Affectation des MV aux nœuds physiques . . . 80
5.4 Mise en correspondance des besoins non-fonctionnels . . . 86
5.4.1 Vérification de la validité des contraintes de placement du client . . . 89
5.4.2 Vérification de la validité des contraintes d'interconnexion du client . 92
5.4.3 Exemple - Le courtier vérifie la satisfiabilité des besoins non-fonctionnels 94
5.5 Conclusion . . . 99

6 Contrôle d'accès et propriétés de sécurité 101
6.1 Introduction . . . 101
6.2 Modèle de base du ACSP-RSM . . . 102
6.2.1 Cas d'usage : système hospitalier . . . 104
6.2.2 Les besoins de contrôle d'accès . . . 106
6.2.3 Les besoins de propriétés de sécurité . . . 107
6.2.4 Conflits et redondances des besoins de CA et de PS . . . 109
6.3 Spécification et analyse formelles des politiques ACSP . . . 110
6.3.1 Spécification du modèle de base de ACSP-RSM . . . 111
6.3.2 Spécification des règles d'autorisation en ABAC . . . 113

6.3.3 Spécification des relations entre les objets et entre les sujets 115
6.3.4 Spécification des propriétés de sécurité requises 116
6.3.5 Analyse formelle des politiques ACSP 117
6.4 Conclusion 124

7 Conclusion et perspectives 127
7.1 Conclusion 127
7.2 Perspectives 128

Bibliographie 131

Liste des figures

1.1 L'architecture générale de l'application de courtage. 4

2.1 Exemple de modèle Alloy. 16
2.2 Contre-exemple d'un prédicat Alloy. 17
2.3 Instance d'exécution d'une fonction Alloy. 17
2.4 Contre-exemple d'une assertion Alloy. 18

4.1 Exemple des offres des fournisseurs. 56
4.2 Interface graphique pour la spécification des besoins du client. 61
4.3 Instance des besoins du client. 63
4.4 Contre-exemple : relation de flux non satisfiable. 68
4.5 Contre-exemple : propriété d'isolation non satisfiable. 69
4.6 Contre-exemple : propriété de concurrence non satisfiable. 72

5.1 L'algorithme de placement. 77
5.2 Exemple : les besoins fonctionnels du client. 82
5.3 Exemple : les offres fonctionnelles des fournisseurs. 83
5.4 Exemple : une des MV ne peut être allouée sur aucun nœud physique. 83
5.5 Exemple : les nouveaux besoins fonctionnels du client. 85
5.6 Exemple : l'ensemble firstSelection proposé par le courtier. 85
5.7 Exemple : les offres fonctionnelles et non-fonctionnelles des fournisseurs. . . . 88
5.8 Exemple : les besoins non-fonctionnels. 89
5.9 Exemple : placement des MV sur les grappes et contraintes non-fonctionnelles. 95
5.10 Exemple : contre-exemple Alloy prouve la non-satisfiabilité de la propriété de concurrence. 97
5.11 Exemple : placement qui satisfait tous les besoins fonctionnels et non-fonctionnels. 98

6.1 Architecture du SI d'un « système hospitalier » 106
6.2 Exemple : contre-exemple, propriété de séparation de privilèges non respectée. 122
6.3 Exemple : contre-exemple, conflit de règle de partage et règle d'accès. 124

Acronymes

- ABAC : *Attribute Based Access Control*
- ACSP-RSL : *Access Control and Security Properties Requirements Specification Language*
- ACSP-RSM : *Access Control and Security Properties Requirements Specification Model*
- CA : Contrôle d'Accès
- CPU : *Central Processing Unit*
- CSA : *Cloud Security Alliance*
- ENISA : *The European Network and Information Security Agency*
- MV : Machine Virtuelle
- NIST : *National Institute of Standards and Technology*
- PS : Propriété de Sécurité
- QdS : Qualité de Service
- RAM : *Random Access Memory*
- RBAC : *Role Based Access Control*
- SAT : *boolean SATisfiability problem*
- SecLA : *Security Level Agreement*
- SLA : *Service Level Agreement*

Introduction générale

1

Sommaire

1.1 Contexte 1
1.2 Contributions 3
1.3 Organisation du manuscrit 6

1.1 Contexte

L'informatique en nuage (*Cloud Computing*) est un concept qui propose des services (stockage, calcul) à distance, accessible via un réseau comme internet. L'informatique en nuage est de plus en plus adoptée pour des raisons essentiellement financières. En effet, il est moins cher d'héberger des données et des services dans l'informatique en nuage qu'acheter du matériel, des logiciels, des licences, de l'énergie et d'assurer leur entretien. Cela facilite l'accès à l'information, le déploiement des grandes applications et plateformes. Les deux principaux avantages de l'informatique en nuage sont l'élasticité et la flexibilité qui signifient que les ressources et capacités de calcul sont allouées et libérées en temps réel pour répondre précisément aux besoins des clients. L'informatique en nuage se caractérise par la disponibilité des ressources et offre des capacités de stockage et des performances de calcul énormes. Finalement elle permet d'augmenter la productivité dans des délais brefs.

Cependant, selon une enquête [50] du CSA (*Cloud Security Alliance*), les projets de sécurité de l'informatique en nuage ont été en tête des projets informatiques en 2014. 75% des entreprises considèrent les projets de sécurité comme importants ou très importants. Une autre étude [54] effectuée par ENISA (*The European Network and Information Security Agency*) montre que 77% des entreprises ont des exigences de sécurité élevées ou très élevées. L'étude montre aussi que le facteur majeur pour lequel les entreprises n'adoptent pas l'informatique en nuage est la sécurité. Les risques sont très variés :

- La perte de données qui peut être due aux attaques ou aux erreurs de configuration et de traitement.
- L'utilisation malicieuse des performances de calcul pour effectuer des attaques.
- La vulnérabilité des technologies partagées et le manque d'isolation dans l'informatique en nuage permettent de réaliser facilement des attaques.

- L'interruption et l'indisponibilité des services de l'informatique en nuage suite aux attaques par déni de service, qui forcent les clients à consommer des quantités anormales de ressources.
- La perte de maîtrise et de gouvernance de ses propres données est due à l'externalisation de l'informatique traditionnelle sur des serveurs distants. En effet, le système d'information, la sécurité, le contrôle et la localisation des données sont gérés par le fournisseur ou par un tiers.
- L'enfermement dans une solution est dû au manque de standard dans l'informatique en nuage. Cela complique la portabilité des données entre les fournisseurs.

Différents efforts ont eu lieu afin d'expliquer aux clients de l'informatique en nuage les solutions de sécurité utilisées par leurs fournisseurs et de leur proposer certaines garanties. La *Cloud Security Alliance (CSA)* est une organisation ayant pour mission de promouvoir l'utilisation des bonnes pratiques en terme de sécurité dans l'informatique en nuage. La CSA a proposé d'intégrer les critères de sécurité dans le contrat *Service Level Agreement (SLA)* signé entre les fournisseurs, les clients et les intermédiaires (une autorité de certification, un tiers de confiance ou un courtier). La CSA a donc proposé plusieurs initiatives, parmi lesquelles, le « *Consensus Assessments Initiative Questionnaire* ». Celui-ci sert à évaluer la sécurité des fournisseurs et la rendre transparente aux clients. Ce dernier est un questionnaire composé de plusieurs questions réparties dans différents domaines tels que : la conformité (CO), la sécurité de l'information (IS), le domaine juridique (LG) et la sécurité de l'architecture (SA).

Ces questions que le client, le courtier ou l'auditeur de l'informatique en nuage peuvent poser, concernent les offres de sécurité, les méthodes et les standards que les fournisseurs adoptent afin de protéger les données des clients. Par exemple, une des questions du domaine IS est : Est ce que vous proposez un chiffrement de données des clients au repos ? Les fournisseurs répondent à toutes les questions tout en expliquant comment ce critère de sécurité est assuré et quels normes et standards sont utilisés (comme HIPAA, ISO27001, ...). Ce consensus a vite été adopté par les grands fournisseurs : *Amazon AWS*, *Microsoft Windows Azure*, *Red Hat OpenShift*, *HP Enterprise*, et d'autres. L'avantage de ce consensus est qu'il fournit des propriétés de sécurité de haut niveau, mais c'est aussi son inconvénient. Ces propriétés sont trop génériques et n'apportent rien d'un point de vue opérationnel.

Actuellement les clients peuvent exprimer des besoins métiers basiques tels que : déployer des applications, héberger des machines virtuelles (MV), avec des contraintes de capacité de stockage, de mémoire et de vitesse des CPU. Cependant, les architectures des systèmes d'information deviennent de plus en plus complexes. Les clients de l'informatique en nuage doivent être capables de personnaliser des modèles de services et d'exiger des besoins métiers plus complexes. De plus, les clients s'intéressent beaucoup aux critères de sécurité. Ils souhaitent exprimer des besoins de sécurité fins pour protéger leurs systèmes d'information, leurs applications et plateformes qui ont tendance à être complexes et très personnalisés. Déployer des plateformes, des sys-

tèmes de santé ou d'éducation et stocker des données de gouvernement ou de défense, nécessitent des critères de sécurité très pointus. Par exemple :

- Chiffrer les données par des méthodes variées et des clés différentes pour un même client, en utilisant des algorithmes que le client propose.
- Isoler les données et les services.
- Gérer des contraintes de partage de données et de communication entre les différentes composantes d'une application déployée sur plusieurs fournisseurs.
- Forcer des modèles de contrôle d'accès inconnus aux fournisseurs.
- Forcer des contraintes de séparation des privilèges sur un système déployé dans l'informatique en nuage.

Dans ce genre de situation, le client ne peut pas se contenter du questionnaire de la CSA. Il doit être en mesure d'exprimer, lui même, ses besoins de sécurité et de s'assurer que les fournisseurs sont capables de respecter ses choix, de satisfaire ses besoins et de protéger ses données et services.

1.2 Contributions

Nous nous plaçons dans le contexte de courtage de l'informatique en nuage qui consiste en un processus intermédiaire entre les fournisseurs des services de l'informatique en nuage et leurs clients. Un courtier négocie les besoins des clients et les offres des fournisseurs dans le but de placer les services/données des clients chez les fournisseurs les plus adéquats qui satisfont tous les besoins.

La figure 1.1 décrit l'architecture générale et les différents composants du courtier que nous proposons. Elle représente aussi le processus de courtage que nous décrivons dans cette thèse.

Étape 1 : Le client décrit le système qu'il souhaite migrer dans l'informatique en nuage ou le système (ressources et services) qu'il souhaite se procurer. Le système contient d'abord la description du modèle (données, applications, machines virtuelles (MV), services, utilisateurs, ...), les invariants du système tels que : les relations et les conflits entre les composants du système ainsi que des contraintes quantitatives comme le nombre et les performances des MV (besoins fonctionnels). Le client décrit ses besoins qualitatifs comme la politique de contrôle d'accès et les propriétés de sécurité (besoins non-fonctionnels).

Les fournisseurs de l'informatique en nuage, de leur côté, décrivent leurs systèmes et offres en tenant compte à la fois des critères physiques du matériel, de performances et de disponibilité, ainsi que des critères qualitatifs tels que les méthodes et standards de sécurité utilisés afin de protéger les données des clients.

Le courtier a deux tâches. La première consiste à vérifier la consistance de la description du système du client et la validité des propriétés de sécurité exigées, car une

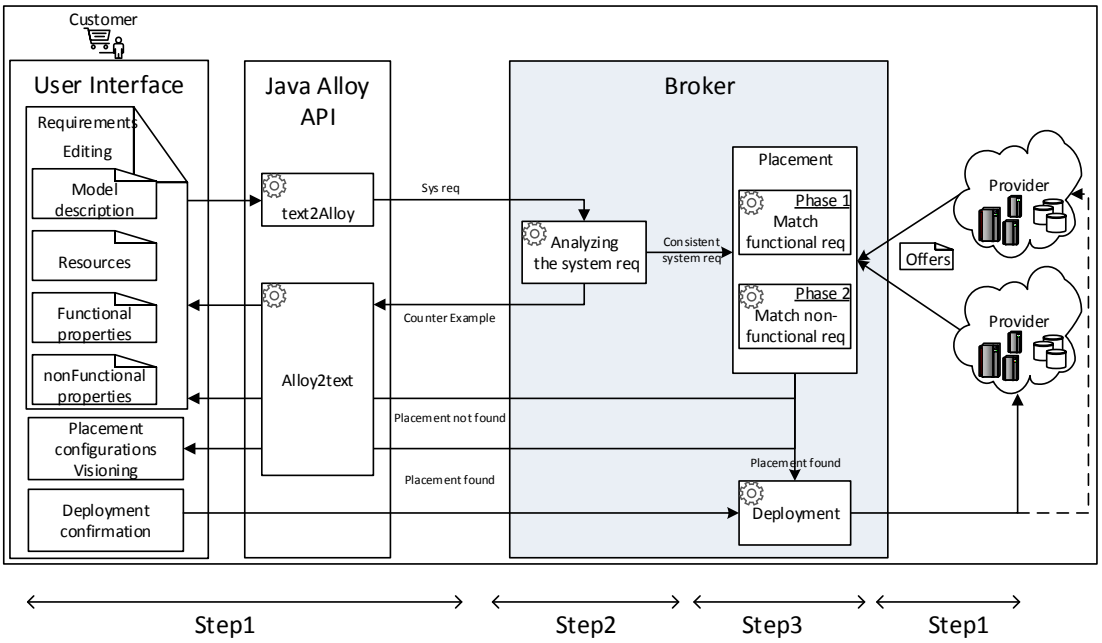

FIGURE 1.1 – *L'architecture générale de l'application de courtage.*

politique de sécurité mal implémentée mène à des vulnérabilités. La deuxième tâche est la mise en correspondance des besoins du client avec les offres des fournisseurs afin d'en sélectionner les plus adéquates. Le courtier devrait effectuer le placement des données/services du client chez les fournisseurs sélectionnés ainsi qu'établir un contrat entre toutes les parties.

Pour la vérification de la cohérence des systèmes décrits par le client, nous avons opté pour les méthodes formelles. Les méthodes formelles fournissent des spécifications décrites à un niveau d'abstraction spécifique. Ces spécifications peuvent être analysées et vérifiées dès les premières étapes du cycle de développement d'un système. Ainsi, il est possible de détecter les conflits ou erreurs de définition avant la mise en œuvre. Ce qui justifie notre choix : nous voulons détecter les conflits des spécifications du client avant de déployer ses services et données chez les fournisseurs. De manière générale, nous souhaitons détecter au plus tôt les points bloquants à chaque étape. Ainsi, on évite des risques de sécurité auxquels est exposé le nuage informatique.
Nous avons décidé d'utiliser le langage de spécification Alloy qui est supporté par l'analyseur Alloy [1]. Ce dernier est un outil de modélisation basé sur la logique du premier ordre. Alloy permet de spécifier des contraintes, des relations et des comportements qui décrivent un ensemble de structures, par exemple : toutes les configurations de sécurité possibles d'une application web. Alloy est un langage déclaratif et permet de faire des descriptions modulaires et des configurations complexes pour les systèmes à vérifier. L'analyseur Alloy permet une analyse syntaxique et sémantique.

1. http://alloy.mit.edu/

Étape 2 : Une fois le client décrit son système, nous vérifions sa consistance en s'assurant qu'il ne contient pas de conflit de spécification. Le courtier analyse les spécifications et retourne les contradictions au client. Le client en prenant connaissance des conflits, peut corriger la description de ses besoins. Une fois que le courtier a reçu une description cohérente des besoins, il procède au placement. Les offres des fournisseurs peuvent être analysées de la même manière, mais nous n'avons pas considéré cela dans nos travaux.

Étape 3 : Nous utilisons encore des méthodes formelles pour placer les services/-données du client chez les fournisseurs adéquats en tenant compte de toutes les contraintes client/fournisseur. D'abord, le courtier met en correspondance les besoins du client avec les offres des fournisseurs. Il commence par sélectionner les fournisseurs qui peuvent assurer les critères quantitatifs comme la vitesse de CPU et la quantité de mémoire. Ensuite, le courtier considère la configuration réseau qui relie les éléments (serveurs physiques) des fournisseurs sélectionnés et vérifie si elle peut assurer des propriétés de sécurité exigées par le client. Cette étape est réalisée en plaçant les données/services du client chez des fournisseurs, ensuite, en vérifiant que ce placement satisfait toutes les contraintes et besoins du client.

À la fin de cette étape, le courtier peut trouver une configuration de placement qui satisfait le client, mais il est aussi possible qu'aucune configuration ne satisfasse ses besoins. Dans tous les cas, le courtier retourne le résultat au client en lui expliquant ce qui ne peut pas être assuré, dans le cas échéant.

Comme décrit dans la figure 1.1, nous avons utilisé l'*API Java Alloy*. Elle facilite l'utilisation de Alloy tout en bénéficiant des avantages de JAVA. En effet, nous avons développé une interface graphique qui permet aux clients de saisir facilement leurs besoins grâce à un assistant de saisie. Les besoins que le client a saisis dans l'interface sont automatiquement traduits en spécifications Alloy. Ces spécifications sont analysées en utilisant Alloy et les résultats sont récupérés via l'*API Java Alloy*.

Nos contributions majeures sont la définition de propriétés de sécurité fines et de politiques de contrôle d'accès comme critères de courtage. Nous utilisons des méthodes formelles pour analyser des systèmes de l'informatique en nuage et vérifier la consistance des spécifications des clients. Nous utilisons Alloy pour trouver des correspondances entre les besoins et les offres. Nous vérifions que les propriétés de sécurité et contraintes exigées par le client sont satisfaites par les offres des fournisseurs sélectionnés et la configuration de placement générée par le courtier. Ainsi, nous proposons un processus de courtage complètement formel qui commence par la description des besoins clients et des offres fournisseurs jusqu'au placement des services/données des clients chez les fournisseurs adéquats. Nos travaux présentés dans cette thèse ont fait l'objet de publications dans des conférences internationales :

[71] Asma Guesmi et Patrice Clemente. « Access Control and Security Properties Requirements Specification for Clouds' SecLAs ». In : IEEE 5th International Conference on Cloud Computing Technology and Science, CloudCom 2013, Bristol, United Kingdom, 2013, Volume 1, pages 723–729.

[72] Asma Guesmi, Patrice Clemente, Frédéric Loulergue et Pascal Berthomé. « Cloud resources placement based on functional and non-functional requirements ». In SECRYPT 2015 - Proceedings of the 12th International Conference on Security and Cryptography, Colmar, Alsace, France, 2015, pages 335–342.

1.3 Organisation du manuscrit

Ce manuscrit est organisé comme suit:

Dans le chapitre 2, nous présentons le langage et l'analyseur Alloy, ses composants et ses caractéristiques.

Dans le chapitre 3, nous présentons l'informatique en nuage, ses caractéristiques, ses modèles de déploiement et de service. L'explosion des offres de l'informatique en nuage a engendré le développement des courtiers. Ces intermédiaires entre les fournisseurs et les clients de l'informatique en nuage sont présentés dans ce même chapitre. Nous présentons quelques projets de recherche et des solutions de courtage informatique qui ont été proposés jusqu'à ce jour. Une section de ce chapitre est consacrée aux menaces et risques de sécurité dans l'informatique en nuage. Nous présentons les travaux de la littérature qui proposent des solutions afin d'améliorer la sécurité dans l'informatique en nuage, notamment les modèles de contrôle d'accès. Nous consacrons une section de ce chapitre pour analyser les initiatives proposées par la CSA ayant pour objectif d'évaluer les offres de sécurité des fournisseurs du nuage informatique. Nous mettons l'accent sur les travaux de la littérature concernant le courtage et qui incluent les critères de sécurité dans le processus de négociation.

Dans les chapitres 3, 4 et 5, nous proposons un processus de courtage des services de l'informatique en nuage, auquel nous ajoutons les critères de sécurité.

Nous utilisons dans le chapitre 4 le langage formel Alloy pour la description des systèmes de l'informatique en nuage, les contraintes et les besoins en termes de propriétés de sécurité. En utilisant l'analyseur Alloy, nous vérifions que les systèmes décrits sont consistants et que les propriétés de sécurité que le client exige ne contiennent pas de conflit. Les inconsistances et conflits détectés sont signalés au client qui à son tour modifie son système afin d'éliminer les inconsistances.

Lorsque les systèmes décrits sont consistants, les besoins des clients et les offres des fournisseurs sont mis en correspondance. Le courtier sélectionne les offres qui répondent aux besoins du client et vérifie ensuite que toutes les contraintes du client sont satisfaites. Ce placement est basé sur plusieurs contraintes. Il est également effectué en utilisant Alloy. Ceci fait l'objet du chapitre 5.

Dans le chapitre 6, nous proposons un langage semi-formel pour la description des besoins fins de sécurité et du contrôle d'accès basé sur les attributs (ABAC) qu'un client pourrait exiger quand il veut adopter l'informatique en nuage. Ces besoins sont traduits en Alloy afin de les analyser et de détecter les conflits qu'ils contiennent. Chaque conflit de spécification de la politique de contrôle d'accès du client lui est notifié. L'objectif est de mettre en place une politique de sécurité cohérente afin d'éviter toute violation ou perte de données ou de contrôle, une fois la politique appliquée.

Nous présentons le bilan de nos travaux et un ensemble de perspectives dans le chapitre 7.

Préliminaires 2

Sommaire

2.1 Introduction 9
2.2 Alloy 10
2.2.1 Modèle Alloy 10
2.2.2 Langage Alloy 10
2.2.3 Analyseur Alloy 10
2.2.4 Structure de Alloy 11
2.2.5 Opérateurs 14
2.2.6 Exemple 15
2.3 Conclusion 18

2.1 Introduction

Dans le chapitre précédent, nous avons présenté nos choix afin de réaliser ce travail. Parmi ces choix, le langage et l'analyseur Alloy nous sont utiles afin de décrire les systèmes de l'informatique en nuage et les contraintes et besoins des clients. Ils nous permettent aussi d'analyser ces systèmes et de vérifier leur cohérence.

Alloy est un langage structurel à syntaxe facile. Il permet de décrire des systèmes afin de les simuler, les analyser et de vérifier des propriétés des systèmes. Il permet une description abstraite des systèmes en se concentrant sur les fonctionnalités importantes. En ignorant les détails d'implémentation, ce langage permet de se concentrer seulement sur les contraintes et les relations entre les composants d'un système et ses comportements. Dans notre travail, nous nous focalisons sur les relations qui peuvent avoir un impact sur la sécurité des systèmes clients dans le nuage informatique.
Dans ce chapitre, nous présentons le langage et l'analyseur Alloy. Nous détaillons la syntaxe et la structure de ce langage. Nous présentons les différents moyens d'analyse d'un système. Ce chapitre contient les informations nécessaires à la compréhension de ce manuscrit mais la présentation de Alloy n'est pas exhaustive, pour plus d'information il faut se référer à l'ouvrage de Jackson [85].

2.2 Alloy

Alloy [1] est un langage de modélisation de structures et un outil pour les analyser. Alloy a été développé au MIT (*Massachusetts Institute of Technology*) par le groupe de recherche *Software Design Group* dirigé par Daniel Jackson [84, 85].

2.2.1 Modèle Alloy

Un modèle Alloy est une collection de contraintes qui décrivent un ensemble de structures, par exemple : les topologies possibles d'un réseau d'interconnexion.
Dans cette thèse, nous analysons les configurations d'interconnexion entre des nœuds dans le nuage informatique et vérifions à l'aide de Alloy que ces configurations assurent des propriétés de sécurité des systèmes des clients.

2.2.2 Langage Alloy

Ce langage a une syntaxe simple qui est basée sur le langage Z [151]. Il est basé sur la logique du premier ordre. Il permet de modéliser des structures en utilisant des relations. Il est déclaratif, il définit des relations, des contraintes et des propriétés pour décrire les systèmes.
Le modèle peut être simulé et les propriétés vérifiées en utilisant l'analyseur Alloy.

2.2.3 Analyseur Alloy

Alloy est soutenu par l'outil « l'analyseur Alloy » qui offre une analyse sémantique automatique des modèles et offre aussi des représentations textuelles et visuelles.
Pour analyser un modèle, autrement dit, pour vérifier sa cohérence et la validité des propriétés le concernant, Alloy utilise une des implémentations de la théorie SAT (*Boolean Satisfiability Problem*) [117].
L'analyseur Alloy offre deux types d'analyse [85]:

- **Simulation :** est réalisée en utilisant la commande **run**. Elle permet d'exécuter un **prédicat** afin d'avoir une instance du modèle qui satisfait toutes les contraintes ainsi que la propriété exprimée dans ce prédicat. Cette commande permet aussi de vérifier la consistance du modèle spécifié, c'est-à-dire elle permet de vérifier que les spécifications ne contiennent pas de contradictions. En effet, si en exécutant un prédicat, l'analyseur ne trouve aucune instance, cela veut dire soit que le prédicat est toujours faux soit qu'il y a des contradictions dans les spécifications et les contraintes du modèle (le modèle est sur-contraint).
- **Vérification :** est réalisée en utilisant la commande **check**. Elle permet de vérifier qu'une propriété exprimée dans une **assertion**, est satisfiable par toutes les instances du modèle. L'analyseur retourne un contre-exemple dans le cas où le

1. `http://alloy.mit.edu/`

modèle ne satisfait pas la propriété. Le contre-exemple consiste en une instance du modèle qui viole la propriété de l'assertion.

Ces deux modes d'analyse sont effectués dans une portée définie par l'utilisateur (appelée **scope**). La portée limite la taille des instances du modèle. Elle limite le nombre d'éléments que l'analyseur génère pour toutes les signatures des instances [85].
Il se peut que l'analyseur ne trouve pas de solution dans la portée définie. Cela pourrait signifier qu'il existe une solution dans une portée plus vaste.
Si l'analyseur trouve un contre-exemple pour une assertion dans une portée donnée alors l'assertion est fausse. Cependant, si aucun contre-exemple n'est trouvé alors l'assertion est satisfiable dans la portée définie mais cela n'implique pas sa satisfiabilité dans une portée supérieure.
L'utilisateur peut augmenter la portée. Cependant il peut atteindre le point où l'analyseur n'est plus en mesure d'épuiser l'espace de possibilités dans un délai raisonnable et sans avoir consommé toute la mémoire de la machine. Cette limite de Alloy dépend du nombre d'atomes, de la portée et surtout de la complexité du modèle à analyser. Même si la portée est petite pour un modèle complexe, Alloy risque d'être à court de mémoire et l'analyse et arrêtée.
De manière générale, une petite portée est suffisante pour trouver un contre-exemple [135]. Il s'agit de l'hypothèse de petite portée (*Small Scope*). Il est possible qu'il y ait un contre-exemple dans une plus grande portée. Mais, dans la pratique, lorsque on augmente la portée, la chance d'avoir un contre-exemple baisse. Ainsi, on obtient une certaine assurance, mais pas dans un sens absolu [83, 85].
Si la portée n'est pas définie par l'utilisateur, alors elle est par défaut limitée à trois éléments pour toutes les signatures [85].

2.2.4 Structure de Alloy

Un modèle Alloy peut être constitué de plusieurs modules. Un module peut être réutilisé et il contient un ensemble de spécifications. Les spécifications peuvent représenter des déclarations de types, de contraintes et de propriétés ou comportements. Dans un module (ou un modèle) Alloy on retrouve des spécifications appelées : signature, fait, prédicat, fonction et assertion.

Dans Alloy, tout est relation. Tous les types de données y compris les scalaires, les ensembles et les tuples sont considérés comme des relations. Un ensemble est une relation unaire. Un scalaire est un ensemble singleton. Un tuple est une relation singleton.

2.2.4.1 Signature

Une signature déclare des ensembles d'atomes et leurs relations les uns avec les autres. La signature est le seul moyen qui permet de représenter les types de base d'un modèle Alloy. Elle est analogue à la notion de classe dans l'orienté objet. Dans

Alloy, chaque classe est représentée par une signature et les attributs de classe sont les champs (atomes) de la signature qui sont décrits de la même manière que dans les langages orientés objet.
Par exemple, on définit une machine caractérisée par deux attributs (cpu et ram) comme suit :

```
abstract sig machine{
  cpu : one CPU,
  ram : one RAM }
```

CPU et RAM sont aussi définis comme des signatures.

```
sig CPU{}
sig RAM{}
```

Un sous-type (sous-signature) peut être défini comme un sous-ensemble d'un autre ensemble. Par exemple, une machine virtuelle (`MV`) est un sous-type du type `machine` et hérite de toutes les propriétés de `machine`. La signature `MP` (machine physique) définit aussi un sous-type de `machine`. Il s'agit de définir une signature comme une extension d'une autre signature en utilisant le mot-clé « extends ».

```
sig MV extends machine{}
sig MP extends machine{}
```

Dans ce cas, `MV` et `MP` sont des types spécifiques qui étendent le type `machine`.

Une signature abstraite est une signature qui n'a comme instances que celles de ses sous-signatures. On a défini `machine` comme une signature abstraite, en utilisant le mot-clé « abstract ». Dans ce cas, une `machine` ne peut être qu'une `MV` ou une `MP`, sinon, Alloy peut générer d'autres instances de type `machine`.

Relation binaire : `cpu` est une relation binaire qui relie les `machine`s aux `CPU`. On utilise l'opérateur de jointure « . » pour accéder au CPU d'une machine.

```
machine.cpu
```

Relation ternaire : Dans l'exemple suivant, nous considérons une configuration (`config`) qui rassemble les liens d'interconnexion (`interconnect`) entre des machines physiques (`MP`). On utilise l'opérateur du produit cartésien « -> » pour définir des relations ternaires.

```
sig config{interconnect : set MP->MP}
```

`(config, mp1, mp2)∈ interconnect` signifie que les deux machines physiques `mp1` et `mp2` sont interconnectées dans config. Dans Alloy, cela peut être exprimé au moyen d'une des deux expressions suivantes qui sont équivalentes.

```
mp1->mp2 in config.interconnect

mp2 in mp1.(config.interconnect)
```

`interconnect` est une relation ternaire qui relie chaque couple de machines physiques à une configuration à laquelle elles appartiennent.

2.2.4.2 Fait

Un fait (mot-clé *fact*) est une contrainte qui est toujours vraie. Un fait est exprimé par une expression logique en utilisant les quantificateurs et les opérateurs logiques. les faits sont satisfaits par toutes les instances du modèle. Par conséquent, définir des faits contradictoires engendre un modèle sur-contraint et incohérent.

```
fact fait_cpu{  all disj mi, mj:MV |  mi.cpu != mj.cpu  }
```

`fait_cpu` exprime le fait que pour tout couple de `MV` disjointes, il faut que le `cpu` de l'une soit différent du `cpu` de l'autre. Le mot-clé « disj » exprime la disjonction. `disj mi, mj` signifie que l'intersection de `mi` et `mj` est vide ou que tout simplement dans cet exemple, `mi` est différente de `mj`.

2.2.4.3 Prédicat

Un prédicat (mot-clé *pred*) est une contrainte réutilisable. Son corps est une expression logique qui est évaluée à `vrai` ou `faux`. Le prédicat peut être satisfait dans une partie du modèle. Il peut avoir des paramètres en entrée qui sont utilisés dans son corps.

```
pred pred_cpu[mi, mj:MV] {  mi.cpu != mj.cpu }
```

Le prédicat `pred_cpu[ mi, mj ]` vérifie si le cpu de `mi` est différent de celui de `mj`, il retourne vrai si c'est le cas, et faux sinon. Ces deux `MV` sont utilisées en paramètres du prédicat.

2.2.4.4 Fonction

Une fonction (mot-clé `fun`) est une expression réutilisable qui retourne un résultat de type spécifique. Une fonction peut avoir des paramètres en entrée qui sont utilisés dans son corps.

```
fun fun_cpu[ mi:MV ]: MV{
  {   mk:MV | mi.cpu = mk.cpu  }
}
```

La fonction `fun_cpu[mi]` retourne l'ensemble des `MV`, tel que chacune a le même `cpu` que `mi` définie en paramètre de la fonction.

2.2.4.5 Assertion

Une assertion (`assert`) est une formule logique évaluée à vrai ou faux, elle ne prend pas de paramètre en entrée. Une assertion exprime une propriété. Alloy vérifie que cette propriété est satisfaite par toutes les instances du système et retourne vrai si c'est le cas. Il retourne un contre-exemple prouvant la violation de cette propriété dans le cas échéant.
Considérons la signature `Maillon` qui est reliée par la relation `suivant` avec un ensemble de Maillons.

```
sig Maillon{suivant : set Maillon}
```

Soit l'assertion `assert_maillon` présentée comme suit :

```
assert assert_maillon
{no m:Maillon | m in m.suivant}
```

La vérification de l'assertion `assert_maillon` retourne un contre-exemple s'il existe un maillon tel qu'il est le suivant de lui-même. Si un tel maillon n'existe pas, toutes les instances possibles du système satisfont cette assertion et donc elle est valide sous l'hypothèse de *small scope*.

2.2.5 Opérateurs

Opérateurs relationnels : Les opérateurs « + », « - », « & », « = » et « in » représentent l'union, la différence, l'intersection, l'égalité et l'inclusion des ensembles.
Considérons la signature Maillon définie plus haut.
L'opérateur * représente la fermeture transitive réflexive. L'expression `m.*suivant` telle que m est de type Maillon, retourne l'ensemble des éléments :

m,
m.suivant,
m.suivant.suivant,
...

L'opérateur ^ représente la fermeture transitive non-réflexive. L'expression `m.^suivant` telle que m est de type Maillon, retourne l'ensemble des éléments :

m.suivant,
m.suivant.suivant,
m.suivant.suivant.suivant,
...

Quantificateurs : Les quantificateurs que Alloy supporte sont :

no : aucune occurrence.
lone : zéro ou une occurrence.
one : exactement une occurrence.
set : zéro ou plusieurs occurrences.

some : une ou plusieurs occurrences.
all : toutes les occurrences.

Ces quantificateurs peuvent être utilisés pour spécifier la cardinalité des atomes (attributs) d'une signature, la cardinalité d'une signature, aussi pour spécifier des contraintes logiques dans un fait, un prédicat ou une assertion. Ils sont utilisés pour spécifier la cardinalité des atomes/relations pour lesquels un fait, un prédicat ou une assertion est appliqué. Par exemple : le fait `fait_cpu` présenté plus haut, est appliqué pour toutes les MV du modèle.

Opérateurs logiques : Les opérateurs logiques dans Alloy sont présentés dans le tableau suivant :

	Alloy
Non logique	!
ET logique	&&
OU logique	||
P implique Q	P $\Rightarrow$ Q
P est équivalent à Q	P $\Leftrightarrow$ Q
Si P alors Q, sinon R	P $\Rightarrow$ Q else R

Où P, Q et R sont des expressions logiques.

2.2.6 Exemple

Nous reprenons les exemples précédents afin d'illustrer un simple exemple de modèle Alloy représenté dans la figure 2.1.

Nous définissons une signature `machine` qui contient deux atomes `cpu` de type `CPU` et `ram` de type `RAM`. `CPU` et `RAM` sont aussi définies comme des signatures. `machine` est définie comme `abstract` donc elle ne peut prendre comme instance que `MV` ou `MP` qui sont des signatures étendues de `machine`. `config` est une signature contenant une relation `interconnect` entre des couples de deux `MP`.

```
abstract sig machine{
  cpu : one CPU,
  ram : one RAM
}
sig CPU{}
sig RAM{}

sig MV extends machine{}
sig MP extends machine{}

sig config{interconnect : set MP->MP}

fact fait_cpu{  all disj mi, mj:MV |  mi.cpu != mj.cpu  }

pred pred_ram[disj mi, mj:MV] {  mi.ram != mj.ram }
check { all disj mi, mj:MV | pred_ram[mi, mj] }

fun fun_ram[ mi:MV ]: MV{
  {  mk:MV | mi!=mk && mi.ram = mk.ram  }
}
run fun_ram for 3 but exactly 2 MV

assert assert_mp {no m:MP | m->m in config.interconnect}
check assert_mp
```

FIGURE 2.1 – *Exemple de modèle Alloy.*

Le fait `fait_cpu` impose que tout couple de `MV` disjointes ait des `cpu` différents. Cette contrainte est appliquée sur toutes les instances du modèle. À part cette contrainte imposée sur le modèle et les signatures définies, Alloy est libre de générer toutes les configurations possibles afin de vérifier des assertions et des prédicats et pour exécuter des fonctions, comme nous allons montrer ci-après.

Le prédicat `pred_ram[mi,mj]` a pour but de vérifier si les deux `MV` `mi` et `mj` ont différentes `ram`. Nous utilisons l'expression `check { all disj mi, mj:MV | pred_ram[mi, mj] }` pour vérifier que dans toutes les instances possibles de ce modèle Alloy, tout couple de `MV` ont des `ram` différentes. L'exécution de ce prédicat retourne le contre-exemple présenté dans la figure 2.2.

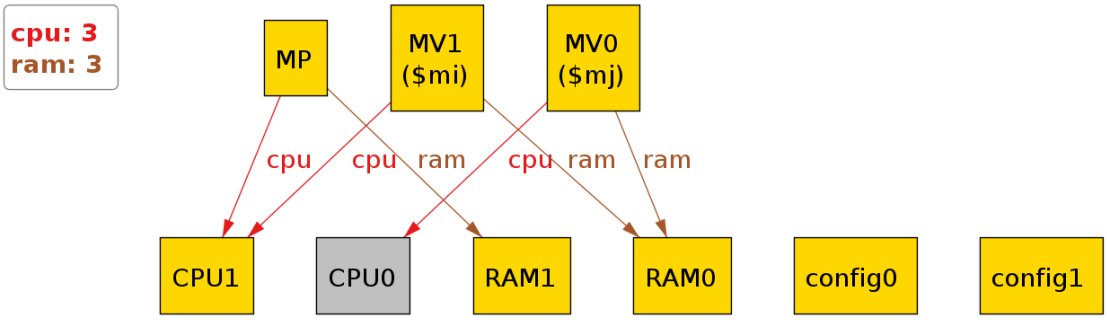

FIGURE 2.2 – *Contre-exemple d'un prédicat Alloy.*

Dans ce cas, nous n'avons pas limité la portée, Alloy donc se limite à 3 par atome/relation au maximum. Les deux `MV` (`MV0` et`MV1`) correspondent respectivement à `mj` et `mi` dans le prédicat et ont toutes les deux `RAM0` comme `ram`. Ce contre-exemple montre que ce prédicat n'est pas satisfait par toutes les instances du modèle. Nous pouvons exécuter un prédicat (en utilisant la commande **run**) ou un ensemble de prédicats à la fois afin d'avoir une instance du modèle qui satisfait les faits ainsi que les prédicats exécutés.
La fonction `fun_ram[mi]` retourne toutes les `MV` qui ont une `ram` égale à celle de `mi`, à part `mi`. Nous utilisons l'expression `run fun_ram for 3 but exactly 2 MV` pour exécuter cette fonction pour une portée maximale de 3 atome/relation mais exactement 2 `MV`. L'exécution retourne l'instance représentée dans la figure 2.3. Cette instance contient deux `MV` ayant la même `ram`.

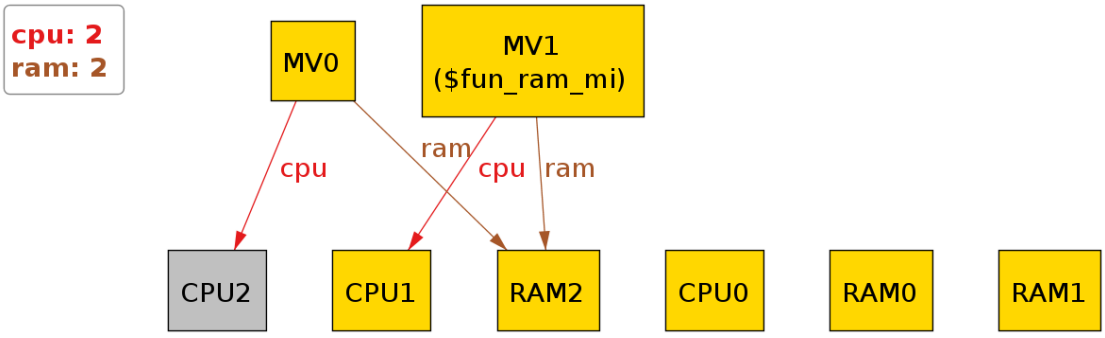

FIGURE 2.3 – *Instance d'exécution d'une fonction Alloy.*

L'assertion `assert_mp` sert à vérifier qu'il n'existe aucune `MP` reliée à elle même par la relation `config.interconnect`. Alloy retourne le contre-exemple de la figure 2.4. Ce dernier montre une `MP` ayant une relation `interconnect` avec elle même.

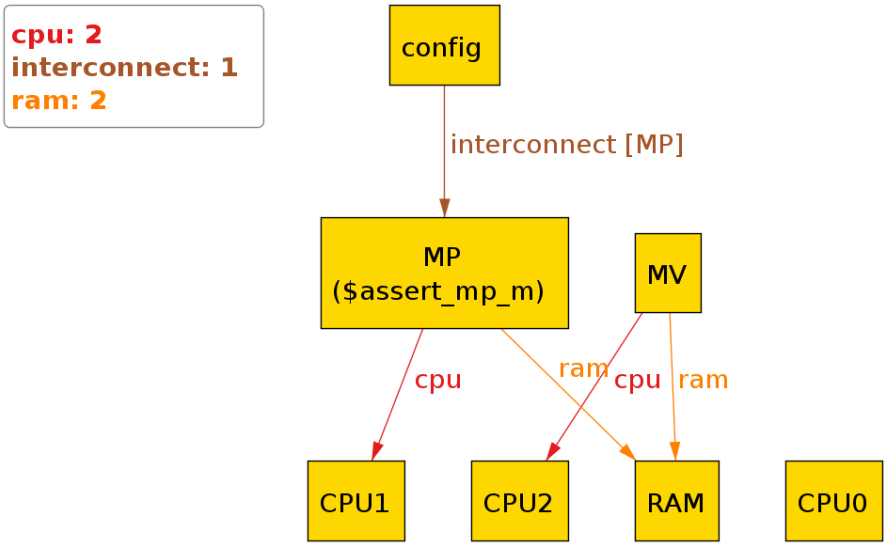

FIGURE 2.4 – *Contre-exemple d'une assertion Alloy.*

2.3 CONCLUSION

Plusieurs langages et outils de vérification formelle existent. Nous avons choisi d'utiliser Alloy car parmi ceux testés, il est le plus adéquat à nos besoins dans ce travail. En effet, l'élément le plus important est de pouvoir décrire des modèles complexes ainsi que leurs comportements.
NuSMV [1, 44] et SPIN [78] sont des vérificateurs de modèles (*model checkers*) et sont plus adaptés à la vérification des propriétés temporelles. Il est difficile de définir des structures et relations complexes en utilisant NuSMV ou PROMELA (langage utilisé par SPIN). NuSMV n'offre pas l'abstraction des instances d'entités [60]. Chaque instance d'une entité ou d'une relation doit être explicitement spécifiée. D'un autre coté, modéliser des spécifications entre plusieurs relations peut être compliqué en utilisant SPIN.

Alloy est un petit langage, son apprentissage est facile. Les concepts adaptés par Alloy sont des standards de la plupart des langages de programmation et de modélisation. Même s'il n'est pas orienté objet, il permet facilement de simuler le paradigme des objets grâce aux notions de signature et de l'héritage. En plus de ces deux dernières, les relations entre les signatures ainsi que les faits, permettent de définir des systèmes complexes. Comme c'est le cas dans ce travail où on considère des systèmes de l'informatique en nuage ainsi que des propriétés de sécurité fines.
Alloy supporte différents niveaux d'abstraction. De plus la notion de portée (*scope*) permet de gérer le nombre d'instances et la taille des systèmes à analyser.
Alloy offre différentes manières pour analyser un système et vérifier des propriétés grâce à sa logique de premier ordre. De plus, il est soutenu par l'outil « l'analyseur

Alloy » qui offre une analyse sémantique automatique et offre aussi des représentations textuelles et visuelles.
Pour tout cela, nous utilisons le langage et l'analyseur Alloy afin de modéliser les systèmes d'informatique en nuage, de les analyser et de vérifier la satisfiabilité des propriétés de sécurité exprimées par un client de nuage informatique.

État de l'art

3

Sommaire

3.1 Introduction 21
3.2 Informatique en nuage 22
3.2.1 Définition générale 22
3.2.2 Caractéristiques essentielles 22
3.2.3 Modèles de service 23
3.2.4 Modèles de déploiement 24
3.3 Courtage nuagique 24
3.3.1 Définition 25
3.3.2 Quelques solutions existantes 25
3.3.3 Synthèse 27
3.4 Sécurité dans l'informatique en nuage 27
3.4.1 Les menaces de la sécurité de l'informatique en nuage 28
3.4.2 Solutions de sécurité dans l'informatique en nuage 30
3.4.3 Cloud Security Alliance 36
3.4.4 Courtage nuagique basé sur la sécurité 38
3.5 Assurance de la sécurité au moyen des méthodes formelles 41
3.6 Analyse et synthèse 42
3.7 Conclusion 43

3.1 Introduction

L'explosion du nombre d'offres et de fournisseurs d'informatique en nuage a fait émerger le concept de courtage des services de l'informatique en nuage. Ce concept a beaucoup d'avantages, notamment la sélection des offres adaptées aux besoins du client, mais aussi la fédération des offres des fournisseurs. Cependant, les courtiers considèrent les besoins métiers et techniques des clients mais pas les besoins en terme de sécurité. Pourtant la sécurité reste le frein majeur de l'adoption de l'informatique en nuage.

Dans ce chapitre, nous présentons les différents concepts en lien avec cette thèse.

Dans la section 3.2, nous définissons le concept de l'informatique en nuage. Nous présentons ses caractéristiques, ses modèles de service et de déploiement. Dans la section 3.3, nous présentons le concept du courtage informatique et quelques travaux de

la littérature le concernant. Nous présentons ensuite les risques majeurs de sécurité dans l'informatique en nuage, dans la section 3.4. Nous présentons quelques solutions de sécurité notamment celles basées sur le contrôle d'accès. Ensuite, nous discutons les aspects d'intégration de la sécurité dans les processus de courtage. Dans la section 3.5, nous présentons des solutions de recherche qui utilisent les méthodes de vérification formelle pour analyser des politiques de sécurité. Enfin, nous soulignons les limitations des travaux de la littérature et nous motivons les travaux de thèse.

3.2 Informatique en nuage

Dans cette section, nous présentons l'informatique en nuage. Nous définissons ce concept. Nous discutons par la suite ses avantages qui concernent ses caractéristiques, ses modèles de service et de déploiement.

3.2.1 Définition générale

L'informatique en nuage, le nuage informatique, l'informatique nuagique est aussi connue sous la dénomination d'infonuagique (*cloud computing* en anglais).
Le NIST (*National Institute of Standards and Technology*) [122] définit l'informatique en nuage par un modèle qui permet l'accès via le réseau et à la demande à un ensemble partagé de ressources informatiques configurables (réseaux, serveurs, stockage, applications et services) qui peuvent être provisionnées et libérées rapidement et avec un minimum d'effort de gestion ou d'interaction avec le fournisseur de service.

Selon le NIST [122], l'informatique en nuage se compose de cinq caractéristiques essentielles, de trois modèles de service et de quatre modèles de déploiement, que nous détaillons ci-après.

3.2.2 Caractéristiques essentielles

L'informatique en nuage est caractérisée par :

Service à la demande : sans avoir besoin d'interaction humaine avec les fournisseurs de service, l'informatique en nuage permet aux clients de se procurer des ressources informatiques de façon flexible. La puissance de calcul et la capacité de stockage sont adaptées automatiquement selon les besoins des clients.

Accès au réseau : les services sont disponibles sur le réseau et sont accessibles à travers des mécanismes standards.

Mutualisation des ressources : les ressources physiques et virtuelles peuvent être regroupées et combinées afin de servir les clients. Elles sont attribuées aux clients et adaptées automatiquement selon la demande. Les clients n'ont pas connaissance de l'emplacement exact des ressources mais peuvent le spécifier à un plus haut niveau d'abstraction (pays par exemple).

Élasticité : les ressources peuvent être rapidement provisionnées et libérées selon la demande du client. Le client a l'impression que les ressources sont illimitées et qu'il peut y avoir accès à tout moment. Cependant, les fournisseurs définissent généralement des seuils.

Service sur mesure : le fournisseur mesure la consommation du client en durée et en quantité de ressources. Il facture le client en se basant sur le modèle « Payez uniquement ce que vous consommez ».

3.2.3 Modèles de service

1. **Infrastructure en tant que service (Infrastructure as a Service – IaaS).** Elle consiste à fournir à l'utilisateur des performances de calcul, des réseaux et du stockage. Le niveau IaaS offre des ressources matérielles virtualisées s'appuyant sur des ressources matérielles physiques. L'utilisateur peut louer des machines virtuelles (MV) sur lesquelles il est capable de déployer et d'exécuter des logiciels y compris des systèmes d'exploitation. Il peut aussi louer des MV pré-configurées par le fournisseur avec des systèmes d'exploitation et des applications installées. L'utilisateur ne peut pas contrôler l'infrastructure nuagique sous-jacente mais il a le contrôle sur les systèmes d'exploitation, le stockage et les applications. Il peut aussi avoir du contrôle sur des composants réseau comme les pare-feux.
   Amazon Web Services (AWS) est parmi les principaux fournisseurs IaaS. AWS fournit du calcul (EC2 [4]), du stockage (S3 [5]), des bases de données (SimpleDB [6]) et d'autres services. OpenStack [14] et OpenNebula [12] sont des plateformes libres qui permettent de mettre en place des infrastructures IaaS.
2. **Plateforme en tant que service (Platform as a Service – PaaS).** Il consiste à fournir un environnement configuré sur lequel l'utilisateur peut développer, tester et exécuter des applications. L'utilisateur peut déployer des applications et ajouter ses propres outils. Il peut contrôler l'application et sa configuration mais pas l'infrastructure sous-jacente (réseau, serveur, stockage). La PaaS est située au-dessus de l'IaaS, le système d'exploitation et les outils d'infrastructure sont sous le contrôle du fournisseur.
   Parmi les fournisseurs du PaaS on peut citer Google App Engine [8], Microsoft Windows Azure [10], Cloud Foundry [7] et OpenShift [13].
3. **Logiciel en tant que service (Software as a Service - SaaS).** Ce type de service offre des applications qui sont accessibles via des périphériques à travers une

interface utilisateur légère comme un navigateur web ou logiciel dédié. L'utilisateur n'a pas à se soucier des mises à jours des logiciels ou comment celles-ci sont fournies. Google Apps [9] et Office365 [11] de Microsoft sont des exemples de SaaS.

3.2.4 Modèles de déploiement

1. Nuage privé : une infrastructure déployée par une organisation pour utilisation interne restreinte à celle-ci. L'accès au nuage informatique est limité aux employés de l'organisation. Il peut être détenu, géré et opéré par l'organisation et/ou une tierce partie.
2. Nuage communautaire : une infrastructure qui est partagée entre plusieurs organisations pour les besoins d'une communauté spécifique. On retrouve ce type dans les milieux académiques par exemple.
3. Nuage public : une infrastructure mise en disposition par un fournisseur qui est ouverte au public, c'est-à-dire à tout le monde sans restriction particulière. Le fournisseur vend au public des services nuagiques. Ces derniers sont accessibles via internet.
4. Nuage hybride : une composition de plusieurs infrastructures différentes de nuages informatiques (public, privé, communautaire) amenées à partager des données et applications. Cela permet aux organisations de profiter des performances et faibles coûts du nuage informatique public et de conserver les données critiques dans le nuage informatique privé.

L'explosion des offres des fournisseurs de l'informatique en nuage a poussé à développer les courtiers nuagiques. Leur objectif est d'aider les clients à bien choisir leurs fournisseurs. Ils peuvent aussi proposer des services de coordination et collaboration entre plusieurs fournisseurs afin de fournir une meilleure qualité de service aux clients.

3.3 Courtage nuagique

Le courtier de nuage informatique est un intermédiaire entre les fournisseurs de services et les clients du nuage informatique.
Le nombre d'offres de services nuagiques augmente et le client trouve des difficultés à sélectionner les offres qui répondent le mieux à ses besoins. Un courtier peut dans ce cas aider le client à exprimer ses besoins et à choisir les meilleures offres.
Si le client souhaite utiliser des services de différents fournisseurs, alors un courtier est nécessaire pour gérer leur intégration et leur coordination. Il peut aussi assurer la flexibilité et l'interopérabilité des services.

Dans cette section, nous présentons le concept du courtage nuagique. Nous définissons d'abord ce concept, ensuite nous présentons les principaux travaux de la littérature qui s'y intéressent.

3.3.1 Définition

Le courtier de nuage informatique ou (*cloud broker* en anglais) est une entité intermédiaire entre les clients et les fournisseurs du nuage informatique. Il gère l'utilisation, la performance et la prestation de services de nuage informatique tout en négociant les relations entre les fournisseurs et le consommateur (client) de nuage informatique [107]. Le courtier fournit des services en trois catégories :

1. **Agrégation** : combiner de multiples offres de nuage informatique dans un ou plusieurs nouveaux services. Le courtier fournit l'intégration des données et des services et assure la sécurité du transfert de données entre le client et les différents fournisseurs des services nuagiques.
2. **Intermédiation** : renforcer un service donné en améliorant des capacités spécifiques et en offrant des services à valeur ajoutée au client. Cette amélioration peut consister en la gestion des accès aux services, la gestion d'identité ou l'amélioration de la sécurité.
3. **Arbitrage** : l'arbitrage est similaire à l'agrégation sauf que les services combinés ne sont pas fixés. Un courtier d'arbitrage a la flexibilité de choisir les services à fournir au client à partir de plusieurs services des fournisseurs. L'objectif est d'optimiser les services fournis au client.

Le courtier de nuage informatique peut fournir [76] :

- des services de support métier et relationnel entre le client et les fournisseurs (intermédiation métier). Le rôle du courtier consiste en la sélection des services des fournisseurs pour répondre aux demandes des clients et en la gestion des contrats entre les clients et les fournisseurs.
- des services de support technique (agrégation, arbitrage et intermédiation techniques). Un accent est mis sur la gestion des problèmes d'interopérabilité entre plusieurs fournisseurs.

3.3.2 Quelques solutions existantes

Plusieurs travaux ont été effectués dans le but de promouvoir le multi-nuages (*multi-cloud*) ou la fédération entre plusieurs fournisseurs de l'informatique en nuage. D'un autre coté, l'objectif est d'aider le client à sélectionner les offres selon ses besoins sans perdre beaucoup de temps à parcourir toutes les offres.

CompatibleOne : CompatibleOne [1] est un courtier libre (open-source). Il offre une interface simple et unique permettant la description des besoins du client en terme de ressources ainsi que leur approvisionnement sur un ou plusieurs fournisseurs adéquats. CompatibleOne permet d'approvisionner des ressources sur une fédération de nuages informatiques hétérogènes. Le modèle de description CompatibleOne Resource Description System (CORDS) peut couvrir les niveaux de nuage (PaaS et IaaS) [174].

Projet Contrail : Contrail est un projet de recherche européen qui propose des services du niveau IaaS et PaaS pour implémenter des fédérations de nuages informatiques. L'objectif de ce projet est de développer une infrastructure libre dans laquelle les ressources appartenant aux différents fournisseurs sont intégrées dans une seule fédération homogène de nuages informatiques. Les clients accèdent à la fédération de manière transparente [38]. Toute organisation est capable d'être fournisseur quand son infrastructure n'est pas utilisée à sa capacité maximale.

Projet OPTIMIS : Comme Contrail, le projet européen de recherche OPTIMIS [59] propose une plateforme multi-nuages pour l'approvisionnement de services. Elle gère le cycle de vie du service dans le nuage à partir de la construction, le déploiement, la configuration et jusqu'à l'exécution. Elle offre des services en se basant sur les performances ainsi que des critères tels que la confiance, le risque, l'éco-efficacité et le coût.

Projet mOSAIC : Le projet mOSAIC définit une plateforme PaaS et une API libre [127, 134] pour décrire et gérer les ressources offertes par les fournisseurs de nuage informatique. mOSAIC a pour objectif d'améliorer l'interopérabilité entre les fournisseurs de nuage informatique et simplifier le développement et le déploiement des applications dans un environnement multi-nuages.

Plusieurs autres projets visent à proposer des plateformes PaaS :
Cloud4SOA [2] est un projet de recherche européen qui vise à permettre la portabilité des applications. Il propose une solution interopérable de multi-PaaS [89]. Cloud4SOA permet la migration des applications d'une PaaS vers une autre.
PaaSage [3] est un projet de recherche européen qui propose une plateforme multi-nuages pour le développement et le déploiement des applications. Il permet la migration des applications existantes vers le multi-nuages.
Aneka [162] est une plateforme qui permet de développer et de déployer des applications réparties.

1. CompatibleOne: The Open Source Cloud Broker. `http://www.compatibleone.com/community/`
2. `http://www.cloud4soa.com/`
3. `http://www.paasage.eu/`

Sélection de services dans le nuage informatique : Plusieurs travaux comparent les offres des fournisseurs afin d'aider le client à choisir les services qui répondent mieux à ses besoins. Hussain et al. [80] proposent une formulation mathématique et une méthode de sélection des offres basée sur une abstraction multicritère. CloudCmp [99] a été développé pour comparer les offres des fournisseurs en terme de l'élasticité du nuage, la persistance du stockage et les services réseaux. SMICloud [64] est un autre framework proposé pour la comparaison et l'évaluation des fournisseurs en se basant sur les besoins du client. Les attributs de comparaison sont le prix, l'élasticité, la disponibilité, la bande passante et les performances de stockage. La sécurité est aussi considérée comme une valeur numérique valorisant le niveau de sécurité de chaque fournisseur. Un framework de sélection des fournisseurs de services SaaS a été proposé dans [23]. Il considère des critères techniques tels que la gestion du stockage et des propriétés de sécurité de haut niveau, par exemple : l'audit est effectué par une partie tierce.

3.3.3 Synthèse

Les travaux de recherche effectués et les courtiers commerciaux existants ne considèrent pas les besoins du client en matière de sécurité. Les contraintes de sécurité personnalisées selon les besoins du client ne peuvent pas être considérées. Ils s'intéressent plus aux critères de tarifs, de capacité de stockage, de performance de calcul et d'élasticité. Quelques solutions considèrent des critères tels que le choix de la localisation géographique des données ainsi que des propriétés de sécurité de haut niveau, par exemple : le fournisseur doit assurer le chiffrement des données stockées.

Dans la section suivante, nous présentons dans un premier temps les risques de sécurité auxquels est exposée l'informatique en nuage. Ensuite, nous présentons quelques travaux de la littérature ayant pour objectif de renforcer la sécurité de l'informatique en nuage. Nous discutons aussi les propositions qui considèrent la sécurité comme critère de sélection des fournisseurs.

3.4 Sécurité dans l'informatique en nuage

Malgré les caractéristiques avantageuses de l'informatique en nuage, le manque de sécurité dans cet environnement représente un frein majeur à son adoption [123]. Le nuage informatique est exposé à plusieurs risques parce qu'il rassemble plusieurs technologies classiques. Il est alors exposé aux mêmes risques que les anciennes technologies ainsi que de nouveaux.

Nous commençons d'abord par présenter les menaces auxquelles est exposée l'informatique en nuage. Nous dressons un aperçu rapide des solutions de la littérature qui ont pour but de renforcer la sécurité de l'informatique en nuage. Ensuite, nous détaillons et discutons les travaux sur l'évaluation et la sélection des offres de sécurité des fournisseurs tout en considérant des besoins exprimés par le client.

3.4.1 Les menaces de la sécurité de l'informatique en nuage

Les risques et problèmes de sécurité dans l'informatique en nuage ainsi que les solutions de l'état de l'art ont été discutés dans plusieurs travaux [153, 70, 130, 172, 123, 139, 56, 16]. Dans cette sous-section, nous dressons un aperçu des vulnérabilités et les menaces de sécurité dans l'informatique en nuage.

1. Fuite et perte de données. Les vulnérabilités d'une MV ou d'une application peuvent être exploitées afin de divulguer des informations aux concurrents.
   À cause de faibles mécanismes d'authentification et de chiffrement, les données peuvent être modifiées ou supprimées. Les données peuvent aussi être perdues de façon non-intentionnelle à cause d'une mauvaise utilisation ou accidentelle à cause des catastrophes naturelles. Il faut utiliser des API sécurisées, du stockage sécurisé, utiliser des mécanismes renforcés du chiffrement et renforcer les clés. Il faut vérifier l'intégrité des données et utiliser les solutions de conservation (*retention*). Les fournisseurs doivent utiliser des moyens de backup de données et de récupération de donnée en cas de sinistre (*disaster recovery*).
2. Détournement des comptes ou du trafic de service. Cela peut être fait en exploitant les vulnérabilités des services, la réutilisation des identifiants et des mots de passe et les attaques de phishing. Si un attaquant récupère l'identifiant et le mot de passe d'un compte de nuage informatique alors il peut altérer la confidentialité, l'intégrité et la disponibilité des services associés. Il peut aussi rediriger les utilisateurs de ces services vers des sites illicites.

   Parmi les solutions, on cite : l'utilisation des politiques de sécurité, la gestion des activités et le renforcement de l'authentification, l'interdiction du partage des identifiants et des mots de passe des comptes.
3. API et interfaces non sécurisées. Quelques fournisseurs proposent des API pour permettre aux clients d'interagir avec les services du nuage informatique. Les vulnérabilités présentes sur les API et l'utilisation malicieuse de ces dernières peuvent transférer directement les vulnérabilités au nuage informatique. Cela peut être évité en utilisant un mécanisme d'authentification et de contrôle d'accès renforcé. Néanmoins, il faut renforcer les modèles de sécurité des interfaces des fournisseurs.
4. Déni de service. Ce sont des attaques qui empêchent les utilisateurs d'accéder aux services de nuage informatique en les rendant indisponibles. Cela peut être réalisé en forçant le service à consommer des quantités anormales de la capacité du processeur, de la mémoire, du disque de stockage ou en surchargeant le réseau.
5. Initiés malveillants. Les employés ou les partenaires d'un fournisseur de nuage informatique peuvent gagner l'accès aux données et aux services et parfois sans aucun risque d'être détectés. Une utilisation malicieuse de tels privilèges d'accès

peut entraîner des effets considérables sur les services offerts. Il est nécessaire de renforcer la transparence des services nuagiques.

6. Abus de services. Les fournisseurs offrent de grandes capacités de stockage et de bande passante ainsi que des périodes d'évaluation gratuites pour que les clients puissent tester leurs offres. Des attaquants peuvent profiter des offres gratuites pour effectuer des attaques. Les fournisseurs de services doivent renforcer les mécanismes d'authentification même pour leurs offres d'évaluation gratuites.
7. Manque de compétences. Beaucoup d'entreprises se sont lancées dans le nuage informatique sans avoir conscience des procédures et des compétences requises. Parmi celles-ci : les risques de sécurité et la qualité de service. Les entreprises doivent effectuer des vérifications de compétences avant une migration vers l'informatique en nuage.
8. Vulnérabilités liées au partage de ressources. Les fournisseurs offrent des services évolutifs grâce à la mutualisation et au partage des ressources. Une application peut être utilisée par plusieurs utilisateurs ayant accès à une machine virtuelle. Une machine physique hôte peut aussi être utilisée par plusieurs utilisateurs via des machines virtuelles. Des vulnérabilités qui peuvent exister au niveau de l'hyperviseur peuvent être exploitées pour gagner l'accès à des machines virtuelles ou des applications et altérer leurs fonctionnements. Il faut renforcer les mécanismes d'authentification et de contrôle d'accès pour les opérations administratives. Renforcer l'isolation des ressources virtualisées peut aussi être une solution.

Discussion

La mutualisation et le partage des ressources sont réalisés grâce aux environnements virtuels. Cependant, les vulnérabilités des machines virtuelles représentent une menace pour la sécurité des services de l'informatique en nuage sur tous les niveaux [123]. Il existe principalement deux types d'attaques sur une infrastructure de virtualisation [123] : le saut entre MV et le rootkit hyperviseur.

Le saut entre MV (*VM Escape*) : L'attaquant exécute des programmes dans une machine virtuelle pour compromettre l'isolation et gagner des privilèges sur l'hyperviseur. Ainsi, l'attaquant peut modifier l'hyperviseur et gagner le contrôle de la machine hôte et par conséquent il gagne l'accès aux autres machines virtuelles hébergées par cette machine hôte.

Le rootkit hyperviseur (*Rootkit in hypervisor*) : Il s'agit d'implanter un hyperviseur malveillant sous l'architecture légitime. Il transforme le système d'exploitation de la machine hôte en MV invitée sur le rootkit. Ainsi, le rootkit peut intercepter tout le trafic, manipuler les activités du système et contrôler toutes les MV hébergées par la machine hôte.

Les risques qui surviennent en exploitant les vulnérabilités de la virtualisation, peuvent être réduits ou évités en utilisant des systèmes de détection/prévention

d'intrusion et des pare-feux. Les politiques de contrôle d'accès doivent être bien implémentées afin de prévenir les accès illégaux internes ou externes. L'isolation entre les ressources virtualisées ainsi qu'entre les machines hôtes, l'hyperviseur et les machines virtuelles doit être renforcée. L'utilisation des pare-feux virtuels peut aussi être une solution afin de contrôler tous les types de trafics sur tous les niveaux [130].

Nous nous concentrons plus sur ces aspects. Nous proposons une solution de placement des ressources du client chez les fournisseurs de nuage informatique. Cette solution satisfait des contraintes du client telles que : isoler des machines virtuelles, contrôler et bloquer le trafic entre les machines virtuelles (sous-section 4.3.2). Dans notre solution, nous considérons le placement des machines physiques, les réseaux qui les interconnectent ainsi que les IDS/IPS et les pare-feux installés dans l'infrastructure des fournisseurs (section 4.2). Notre solution assure au client : en tenant compte de l'architecture du nuage informatique, nous lui proposons un placement de ses ressources tout en respectant ses contraintes de sécurité (chapitre 5).

3.4.2 Solutions de sécurité dans l'informatique en nuage

Plusieurs concepts et mécanismes doivent être utilisés afin de sécuriser les environnements d'informatique en nuage, notamment : l'authentification et la gestion d'identité, le contrôle d'accès, les systèmes de détection et de prévention d'intrusion. Ces mécanismes de sécurité existaient depuis longtemps et ont été adaptés afin de répondre aux caractéristiques de l'informatique en nuage, telles que la mutualisation des ressources et l'élasticité. Dans la littérature, différentes extensions et adaptations des mécanismes de sécurité classiques ont été proposées. De nouvelles architectures et frameworks de sécurité ont aussi été implémentés. Dans la suite, nous présentons les travaux de la littérature sur la sécurité des environnements d'informatique en nuage. Nous considérons notamment le contrôle d'accès et les systèmes de détection d'intrusion.

3.4.2.1 Contrôle d'accès

Le contrôle d'accès est ce qui permet de gérer l'accès aux entités d'un système. Une politique de contrôle d'accès définit les règles d'autorisation ou d'interdiction d'accès. Le contrôle d'accès est un des mécanismes indispensables pour sécuriser les environnements de l'informatique en nuage. Cependant, étant donnée la dynamicité de ces derniers, il faut adapter les modèles classiques de contrôle d'accès ou en proposer de nouveaux qui répondent mieux aux caractéristiques évolutives de ces environnements. Une analyse des différents modèles de contrôle d'accès dans l'informatique en nuage a été dressée dans [96]. Parmi les modèles classiques, ABAC est plus flexible que les modèles DAC, RBAC, ou MAC [87, 96].
Dans [15], les auteurs proposent de combiner ces quatre modèles de contrôle d'ac-

cès classiques afin de sécuriser les accès aux données de santé dans l'informatique en nuage.
Dans la suite, nous présentons les principaux modèles de contrôle d'accès implémentés et utilisés dans les environnements d'informatique en nuage.

Contrôle d'accès discrétionnaire : (DAC, *Discretionary Access Control*) Le contrôle d'accès *discrétionnaire* (DAC) est le plus ancien modèle de protection. Les règles d'accès à chaque ressource sont définies par son propriétaire. Ce modèle a été formalisé dans [95] en utilisant des listes de contrôle d'accès.
Des travaux tels que [42, 166, 150] ont utilisé le modèles DAC dans l'informatique en nuage.
Il a été démontré que le modèle DAC (ainsi que ses améliorations successives, telles que HRU [75] que le modèle MTAM [143]) ne permettent pas de garantir des propriétés de sécurité sur un système réel. En effet, ce n'est qu'au prix de simplifications qui ne correspondent pas à la réalité des systèmes d'exploitation que l'on peut arriver à garantir des propriétés sur un système discrétionnaire. Le modèle DAC est par exemple très vulnérable aux fuites d'information. De plus, la gestion des listes de contrôle d'accès peut augmenter le temps d'exécution et par conséquent réduire la productivité [167, 22].

Contrôle d'accès basé sur les rôles : (RBAC, *Role Based Access Control*) RBAC a été proposé dans [57] pour améliorer la gestion des politiques de contrôle d'accès. Les autorisations d'accès sont basées sur des rôles attribués aux utilisateurs selon leurs fonctions dans le système [57].

Bien que la notion de *rôle* introduite dans RBAC ne change rien à la faiblesse sécuritaire d'un modèle DAC (elle permet seulement de factoriser les règles de protection et donc de simplifier l'écriture de la politique), RBAC a été grandement étudié. Plusieurs extensions de RBAC ont été proposées, telles que : OrBAC [55], hierarchical RBAC [58], TrustBAC [41], Geo-RBAC [25], etc.

Et plus récemment, le modèle RBAC a été utilisé par plusieurs plateformes de l'informatique en nuage [68], par exemple : OpenStack [3] et Xen [2]. Dans [67, 68], les auteurs analysent les politiques de contrôle d'accès basées sur le modèle RBAC dans des environnements collaboratifs tels que le nuage informatique. Ils vérifient des propriétés de sécurité comme la séparation des privilèges.
Le modèle RBAC et ses variantes ont été beaucoup utilisés comme modèle d'autorisation et de protection dans l'informatique en nuage [37, 149, 161, 104, 158, 43, 29, 156, 157].

Le NIST a proposé un standard [58] pour le modèle RBAC qui a été adopté en 2004. Par la suite, le NIST a introduit une révision au modèle RBAC et a proposé un modèle basé sur les attributs ABAC [93]. Selon [93], les attributs peuvent remplacer

les rôles et les règles d'accès définies en fonction d'attributs peuvent apporter plus de flexibilité aux modèles RBAC.

Contrôle d'accès obligatoire : (MAC, *Mandatory Access Control*) impose une politique non modifiable par les utilisateurs finaux. Pour contrôler les accès entre sujets et objets, Anderson [19] propose d'utiliser un *Moniteur de Référence* (*Reference Monitor*). Bien qu'extrèmement puissant en terme de sécurité, le modèle MAC, s'il est bien appliqué, est compliqué à administrer. En effet, le très grand nombre de règle et leur extreme finesse rend l'écriture des politiques MAC très difficile, et donne un modèle de protection assez rigide. Cela rend l'utilisation du modèle sur des systèmes complexes (i.e. comportant beaucoup d'entités actives et passives) très difficile. Cela peut donc fortement réduire la productivité des fournisseurs d'accès et même des clients. [167].
Cependant, la garantie forte de sécurité qu'offre MAC fait que sVirt [126] l'intègre par exemple pour protéger les plateformes de virtualisation. Il a pour objectif de renforcer l'isolation entre les MV et protéger contre l'exploitation des vulnérabilités de l'hyperviseur.
De même, sHype [142] applique le modèle MAC dans l'hyperviseur open-source Xen, afin de gérer et de contrôler les communications entre les MV.

Contrôle d'accès basé sur les attributs : (ABAC, *Attribute Based Access Control*) ABAC est un modèle qui définit les autorisations en fonction des caractéristiques des entités. Chaque règle d'accès est une fonction des attributs des sujets, des ressources et de l'environnement [175, 79].

Les attributs dynamiques tels que la localisation géographique et le temps peuvent être gérés facilement dans le modèle ABAC [51]. Il a été prouvé dans [87] que ABAC peut couvrir les modèles RBAC, MAC et DAC. Cependant, l'explosion du nombre d'attributs peut compliquer la gestion de la politique de contrôle d'accès. Plusieurs travaux ont implémenté des modèles ABAC pour l'informatique en nuage [31, 18, 92, 40] et pour les environnements collaboratifs [178]. D'autres se sont intéressés à la spécification et la gestion des contraintes sur les attributs [28, 30].

Chiffrement basé sur les attributs : Le chiffrement basé sur les attributs ABE (*Attribute Based Encryption*) a été introduit dans [141] et appliqué dans l'informatique en nuage dans [101]. Plusieurs extensions de ABE ont été proposées : HABE [165], HASBE [164] et CP-ABE [26], KP-ABE [69].
Dans ces modèles, la politique de contrôle d'accès est représentée par une structure d'accès basée sur les attributs, ce qui permet d'appliquer des politiques complexes.
HASBE [164] est proposé afin d'assurer la sécurité des données stockées dans le nuage informatique. ABE et ses extensions ont été beaucoup utilisés afin de sécuriser le partage et le stockage de données dans l'informatique en nuage, notamment dans le contexte médical [102, 103, 110, 111].

Ces solutions nécessitent d'énormes capacités de calculs et des mécanismes compliqués pour la gestion des clés et structures de chiffrement.

Sah et al. [140] proposent un contrôle d'accès au réseau pour filtrer les accès non-légitimes aux applications dans l'informatique en nuage. Une architecture de gestion des API dans l'informatique en nuage est proposée dans [170]. Elle fournit du contrôle d'accès basé sur des jetons.

Les modèles de contrôle d'accès existants ne permettent pas d'exprimer toutes les exigences de sécurité qu'un client peut exprimer dans l'informatique en nuage. En effet, il faut que les politiques soient adaptées aux services client déployés chez plusieurs fournisseurs. Ces politiques doivent aussi supporter les différentes contraintes concernant les relations entre ces services qui peuvent être très dynamiques.

3.4.2.2 Systèmes de détection/prévention d'intrusion

Différents mécanismes ont été utilisés afin d'améliorer la sécurité des communications dans l'informatique en nuage. Les systèmes de détection et de prévention d'intrusion ont été beaucoup étudiés et développés dans la littérature. On distingue différentes méthodes de détection d'intrusion. La détection basée sur les signatures définit des motifs d'attaque et détecte les tentatives similaires [108, 119, 124]. La détection basée sur les anomalies utilise des tests et statistiques afin de définir des événements anormaux par rapport au comportement normal du système [63, 27]. La méthode basée sur les réseaux de neurones artificiels identifie et classifie les activités réseaux normales et anormales en se basant sur les caractéristiques des attaques apprises à partir d'une source de données durant une phase d'apprentissage [124]. Dans la suite, nous présentons quelques types de systèmes IDS développés et utilisés dans les environnements d'informatique en nuage.

IDS machine : (HIDS, *Host based Intrusion Detection System*) collecte, analyse et gère les informations des fichiers et appels système, événements réseau, ... d'une machine spécifique. Ce système prend un cliché (*snapshot*) des fichiers système par exemple et les compare aux clichés précédents. Une alerte est émise en cas de détection de changement ou de suppression des fichiers. Son inconvénient est qu'un IDS doit être installé sur chaque MV, hyperviseur et machine hôte.

Différents mécanismes offrant ce type d'IDS ont été proposés dans [94, 21]. Un système de détection d'intrusion multi-niveau a été proposé dans [98]. Cet IDS est installé dans chaque MV et s'adapte au niveau de risque calculé à partir des comportements interceptés de la MV. La motivation d'un tel travail est de réduire les ressources qu'un IDS peut consommer sur une MV. Cependant, ce type d'IDS ne peut pas détecter les attaques à large échelle, étant donné que chaque IDS agit seul et indépendamment sur une machine [132].

Des comportements anormaux sont détectés dans les MV [27] dans le but de détecter les vers. Les auteurs listent quelques comportements qu'ils considèrent anormaux, par exemple : identifier le processus qui s'est répandu dans plusieurs MV, si le nombre de ces MV dépasse un seuil, ces MV sont isolées. L'inconvénient de cette démarche est que les vers changent souvent de processus et qu'ils sont du fait difficiles à tracer sur le système.

IDS réseau : (NIDS, *Network based Intrusion Detection System*) analyse et gère le trafic d'un segment de réseau. Il peut être composé de plusieurs détecteurs, chacun contrôle le trafic d'un segment de réseau. Un tel IDS ne peut pas scanner du trafic chiffré, il scanne seulement les parties non-chiffrées des paquets dans le cas du trafic chiffré [132].
Snort est un système de détection d'intrusion libre et est utilisé dans Eucalyptus pour détecter les intrusions issues des réseaux externes [119]. Il est également utilisé dans l'environnement de virtualization Xen [173].
D'autres solutions pour sécuriser les réseaux virtuels ont été proposées dans [169, 100]. DCPortalsNg [125] est une technique qui isole le réseau virtuel de chaque MV.
Un *IDS as a Service* a été proposé dans [74] et est fourni à l'utilisateur comme un service à la demande. Cet IDS utilise Snort et un algorithme performant de détection d'intrusion [168].

IDS hybride : (DIDS, *Distributed Intrusion Detection System*) consiste en plusieurs IDS (HIDS et NIDS, etc.) interconnectés à un serveur central. Chaque IDS est installé dans une région du nuage informatique. Quand un IDS détecte une intrusion dans sa région, il alerte les autres. Des mécanismes d'IDS distribués pour l'informatique en nuage ont été proposés dans [52, 108]. Ce type d'IDS permet de prévenir les points uniques de défaillance. L'inconvénient est que le serveur central peut être surchargé et donc peut être coûteux car il consomme beaucoup de ressources de calcul et de stockage.

IDS hyperviseur : (NIDS, *Hypervisor based Intrusion Detection System*) analyse les communications entre les MV, entre les MV et l'hyperviseur et dans le réseau virtuel de l'hyperviseur.
[63] propose un IDS basé sur l'introspection des MV. Ce mécanisme est aussi utilisé dans [88] pour contrôler l'exécution des applications et des systèmes d'exploitation. Dans [97], les auteurs proposent d'analyser les données des MV par un processus de l'IDS qui est externe à cette MV.

3.4.2.3 Framework sécurisé

Dans la suite, nous présentons un ensemble de travaux de la littérature qui a proposé des infrastructures sécurisées ou des couches logicielles afin d'assurer plus de sécurité à l'informatique en nuage.

ACPS [109] est une architecture de contrôle de l'intégrité des MV et des composants de l'infrastructure. Il vérifie périodiquement les fichiers exécutables du système. Il contrôle les MV et les communications entre les composants de l'infrastructure. Ce système réussit à bloquer plusieurs types d'attaques entre les MV, par exemple : l'introduction d'un module invisible qui intercepte les fichiers système et l'activité réseau.

A. Ibrahim et al [81] proposent CloudSec, une solution de virtualisation qui contrôle la mémoire physique des MV en utilisant une technique d'introspection de MV. D'autres environnements d'exécution sécurisés pour MV ont été proposés dans [176, 171].

Dans [86], les auteurs proposent un modèle où le déploiement des MV se fait en considérant un modèle de menace défini par le client. À chaque serveur est attribuée une capacité selon des informations de sécurité collectées auprès des serveurs. Le déploiement des MV se fait en considérant le modèle de menace et les capacités des serveurs. En cas de violation de sécurité ou de changement de la capacité d'un serveur, une migration des MV se fait automatiquement vers un autre serveur plus adéquat au modèle de menace des MV. L'inconvénient de cette solution est que le client doit être capable de définir un modèle de menace pour sa MV. Par ailleurs, l'évaluation des capacités des fournisseurs est basée sur les attaques potentielles, alors que ces dernières évoluent constamment.

Ces auteurs ont déjà proposé plusieurs travaux sur le risque de sécurité interne qui peut provenir d'un hyperviseur malicieux. NoHype [91, 154] est une architecture d'informatique en nuage qui n'a pas d'hyperviseur. Elle repose sur les capacités matérielles pour assurer l'isolation entre les MV. Cette architecture garde quand même confiance dans les autres logiciels tels que le gestionnaire des MV. De plus, elle nécessite beaucoup de changements au niveau matériel et au niveau de la couche de virtualisation. Bien que les auteurs aient proposé plusieurs solutions pour renforcer la sécurité au niveau IaaS, aucune démonstration, implémentation ou mesure de performance n'est présentée dans leurs travaux.

3.4.2.4 Discussion

D'une part, les mécanismes de sécurité qui ont été développés pour apporter plus de sécurité à l'informatique en nuage sont hors d'atteinte des clients. La vision qui leur en est offerte est très simplifiée. Un client ne peut donc évidemment pas paramétrer les mécanismes de sécurité éventuellement présents chez un fournisseur, s'il sait à peine quels sont-ils. Au mieux, si le client peut avoir accès à certains mécanismes de sécurité, ils sont la plupart du temps techniquement compliqués à comprendre

par celui-ci. De plus, les modèles de contrôle d'accès ou les systèmes IDS existants ne permettent pas au client de spécifier des propriétés de sécurité spécifiques à son cas d'usage. Sans compter qu'il faut les adapter dans le cas ou le client utilise des ressources de plusieurs fournisseurs.

Il est donc nécessaire d'avoir un modèle de sécurité plus global qui peut couvrir toutes les propriétés de sécurité dans l'informatique en nuage et qui peut être adapté à tous les cas d'usage, et auquel le client puisse avoir un accès clair et simplifié.

3.4.3 Cloud Security Alliance

La CSA (*Cloud Security Alliance*) est une organisation à but non lucratif. Elle a pour mission de promouvoir l'utilisation des meilleures pratiques pour assurer la sécurité dans l'informatique en nuage.
Après avoir défini les risques de sécurité auxquels est exposé un nuage informatique, les membres du CSA ont lancé une initiative dans le but de rendre la sécurité dans l'informatique en nuage transparente et d'adapter les normes comme ISO 27001 à l'informatique en nuage. ISO 27001 est une norme de sécurité internationale très largement adoptée qui définit des exigences à respecter pour les systèmes de gestion de la sécurité des informations. Cependant, cette norme et d'autres ne couvrent pas la complexité de l'informatique en nuage [17, 133].
CSA a proposé des projets afin de guider le client à bien choisir ses fournisseurs et aider les fournisseurs à vérifier leur conformité à des normes de sécurité et d'auto-évaluer leur niveau de sécurité. La CSA a proposé un registre STAR (*Security, Trust & Assurance Registry*). Ce dernier regroupe les rapports des fournisseurs documentant leurs offres de sécurité. Les fournisseurs peuvent remplir un questionnaire CAIQ (*Consensus Assessments Initiative Questionnaire*) ou renseigner leurs conformités aux standards fournis dans la matrice CCM (*Cloud Controls Matrix*). L'objectif des initiatives de la CSA est d'aider les fournisseurs à vérifier leurs conformités vis-à-vis des exigences les plus élevées définies par la CSA. D'autre part, l'objectif est de permettre aux clients d'évaluer la capacité d'un fournisseur à éviter les risques cités dans la sous-section 3.4.1 et sa compétence à assurer la sécurité des données et des services dans le nuage informatique.

3.4.3.1 CSA Security, Trust & Assurance Registry (STAR)

STAR [45] est un registre gratuit, accessible publiquement, qui documente les contrôles de sécurité fournis par les différentes offres de l'informatique en nuage. STAR offre des meilleures pratiques, des standards et des normes de la sécurité dans l'informatique en nuage. Il a pour objectif de fournir de la transparence, de l'audit et de la surveillance continue des offres de sécurité des fournisseurs de l'informatique en nuage. Ce registre permet aux fournisseurs de publier leurs évaluations CSA de sécurité. Il permet aux clients ou futurs clients d'évaluer et comparer les niveaux de

sécurité des fournisseurs.
Afin d'être inclus dans le registre STAR, les fournisseurs peuvent remplir le questionnaire CAIQ de CSA ou répondre aux contrôles de la matrice CCM de CSA. Plusieurs fournisseurs ont répondu à cette initiative : Amazon AWS, Red Hat OpenShift, Microsoft Azure, Dropbox, etc. Les fournisseurs peuvent surestimer leurs contrôles de sécurité en répondant aux CAIQ ou CCM. Pour cela, la CSA a proposé trois niveaux d'assurance [45] :

1. Auto-Évaluation CSA STAR (niveau 1) : Les fournisseurs envoient des rapports documentant leur conformité aux contrôles CSA fournis dans le CAIQ ou la CCM. Ces rapports sont directement publiés dans le registre STAR.
2. Évaluation, Attestation, Certification (niveau 2) :
   - Attestation CSA STAR : Une évaluation d'un fournisseur de service de l'informatique en nuage menée par un tiers indépendant. Elle considère des principes des services de confiance définis par l'AICPA (*American Institute of Certified Public Accountants*) ainsi que la matrice CCM.
   - Certification CSA STAR : Une évaluation rigoureuse d'un fournisseur menée par un tiers indépendant. Elle s'appuie sur les exigences du standard ISO/IEC 27001:2005 avec la matrice CCM.
   - Évaluation CSA C-STAR : Une évaluation solide d'un fournisseur menée par un tiers indépendant pour le marché de la grande Chine qui harmonise les meilleures pratiques du CSA avec les normes nationales chinoises.
3. Contrôle continu CSA STAR (niveau 3) : Actuellement, en cours de développement, le contrôle continu est basé sur l'audit et l'évaluation continus des propriétés de sécurité pertinentes. Il permet l'automatisation des pratiques de sécurité des fournisseurs.

3.4.3.2 Cloud Controls Matrix (CCM)

La matrice CCM [49] est un liste de contrôles de sécurité dans l'informatique en nuage, correspondant aux normes, aux meilleures pratiques et aux réglementations. Elle offre aux organisations les structures, les détails et la clarté concernant les informations de sécurité dans l'informatique en nuage. Chaque contrôle est croisé avec d'autres normes de sécurité des systèmes d'information également utilisées dans l'industrie de l'informatique en nuage, comme ISO 27001/27002, ISACA COBIT, PCI, NIST, etc.
Ces contrôles sont groupés en plusieurs domaines alignés avec les 14 qui sont définis par la CSA [47]. Parmi ces domaines : des questions juridiques, la gestion de la conformité et de l'audit, la gestion de l'information et la sécurité des données, le chiffrement et la gestion des clés, la virtualisation sont adressés. L'objectif de la CCM est de fournir des principes de sécurité afin de guider les fournisseurs et leur permettre d'auto-évaluer leur niveau de sécurité. D'autre part, la CCM permet d'assister les clients dans l'évaluation des risques de sécurité d'un fournisseur.

3.4.3.3 Consensus Assessments Initiative Questionnaire (CAIQ)

Le questionnaire CAIQ [46] est basé sur la CCM. Il fournit un ensemble de questions qu'un client ou un auditeur peut poser à un fournisseur de service de l'informatique en nuage afin de vérifier leur conformité aux meilleures pratiques de la CSA et aux contrôles de la matrice CCM. Il permet aux fournisseurs de référencer et documenter leurs offres de sécurité aux niveaux IaaS, PaaS et SaaS.

Ce questionnaire fournit un ensemble détaillé de questions pour évaluer les capacités et compétences de sécurité des fournisseurs. Parmi les questions, on retrouve dans le domaine « architecture de sécurité » la question : « Est ce que les environnements système et réseau sont logiquement séparés afin d'assurer la protection et l'isolation des données sensibles ? ». Les fournisseurs peuvent répondre seulement par oui ou non (ou non applicable). Ils peuvent aussi développer leurs réponses en faisant référence aux normes qu'ils suivent.

3.4.4 Courtage nuagique basé sur la sécurité

L'obtention d'un certificat de sécurité tel que ISO 27000 ou NIST-FISMA améliore la confiance que font les clients dans les offres de sécurité des fournisseurs de l'informatique en nuage. Cependant, les normes assurées par de tels certificats ne couvrent pas toute la complexité de l'informatique en nuage [17, 133]. La mutualisation et le partage des ressources exigent des besoins de sécurité différents pour chaque client et chaque fournisseur de services.

Il est donc nécessaire d'avoir des standards pour définir et mesurer des propriétés importantes dans le nuage informatique telles que la disponibilité de service [77]. L'objectif est de rendre la qualité de services et de sécurité des fournisseurs transparente aux clients et de pouvoir la mesurer de façon commune à toutes les parties dans l'informatique en nuage. Il est possible de mesurer le niveau de confidentialité des services d'un fournisseur, mais à la vue de l'absence de standards, chaque fournisseur, courtier ou client la définit et la mesure différemment. Le niveau de confidentialité peut être mesuré par la robustesse du mécanisme utilisé (contrôle d'accès, chiffrement, etc.), par la taille des clés de chiffrement, etc. La localisation géographique peut être considérée comme un pays, une région, des plages d'adresses IP, etc. Plusieurs questions restent ouvertes, telles que : quel langage utiliser pour décrire les besoins en matière de sécurité des clients et les offres des fournisseurs, comment considérer les critères de sécurité dans le processus de négociation entre le client et les fournisseurs, comment intégrer et gérer la sécurité dans les contrats SLA, comment rassurer le client que ses exigences de sécurité sont bien implémentées et assurées.

Ce domaine offre plusieurs opportunités en recherche. La CSA [49, 46, 45], le NIST [107] et l'ISO [82] essayent de définir des normes et des standards afin de réunir toutes les parties à adopter ces normes et faciliter la transparence dans le nuage informatique.

Plusieurs travaux de recherche ont été menés en exploitant les initiatives de CSA et de NIST. Ces travaux ont pour objectifs principalement d'évaluer les offres de sécurité des fournisseurs, de classer les meilleures offres en prenant ou pas en compte les besoins du client. Dans la suite, nous dressons un état de l'art non exhaustif de ces travaux.

Langages de spécification et SLA SLA (*Service Level Agreement*) est un document qui spécifie les termes et les conditions d'un contrat entre le client et le ou les fournisseurs de service [16]. Il précise la capacité, la quantité de ressources et la qualité de service que le fournisseur doit assurer. D'autres critères comme les tarifs, la localisation géographique ainsi que les pénalités en cas de violation font partie du contrat SLA.

Les langages de spécifications des SLA tels que *WS-Agreement* [20] et *WSLA* [112] se concentrent sur les mesures de qualité de services et manquent de la sémantique nécessaire pour exprimer le niveau de sécurité des services [24].
Dans [53], les auteurs mettent l'accent sur la nécessité d'améliorer le niveau de spécification de la sécurité dans les contrats SLA ainsi qu'avoir un framework de gestion et de contrôle de la sécurité dans les SLA. De plus, des contrats secSLA (*security SLA*) doivent faire partie des contrats SLA signés entre les fournisseurs et les clients [115]. Les secLA sont des contrats SLA qui décrivent les contrôles de sécurité que le fournisseur doit assurer.
Les auteurs de [24] proposent un framework de gestion de la sécurité dans les contrats SLA. Ils décrivent six phases du cycle de vie des contrats de sécurité SLA : la publication des offres, la négociation des contrats de sécurité, l'engagement et la signature des contrats, l'approvisionnement des mécanismes de sécurité, le suivi et le contrôle des détails de sécurité et enfin la résiliation des contrats. Ils considèrent des propriétés de sécurité de haut niveau, par exemple : tous les messages doivent être signés numériquement. Ils décrivent leur proposition sans détailler chaque processus du cycle de vie, par exemple les détails de la négociation et le choix des offres appropriées.
Dans [121], les auteurs considèrent un courtier qui négocie les contrats SLA entre les fournisseurs et les clients. Ils discutent la possibilité d'exprimer les besoins de sécurité dans sept langages de spécification existants qui permettent de spécifier quelques aspects de la sécurité comme WS-agreement [20]. Ils évoquent l'importance d'avoir un langage de spécification commun qui supporte les critères de sécurité. Cela facilite la découverte des offres de sécurité chez les fournisseurs ainsi que la négociation.
CloudSurfer [61] est un prototype d'outil de sélection des offres en fonction des critères de sécurité du client. C'est une application disponible en libre accès qui montre la possibilité d'intégrer les propriétés de sécurité dans un processus de courtage des services de l'informatique en nuage. Les propriétés de sécurité ne peuvent pas être personnalisées selon les besoins du client. De plus, c'est un prototype qui peut être amélioré afin de mettre à jour les offres des fournisseurs et besoins du client dynamiquement.

Évaluation

Les auteurs dans [17] proposent d'évaluer les offres de sécurité des fournisseurs en se basant sur des mesures de confidentialité, d'intégrité et de disponibilité. Ils ont pour objectif d'adapter la norme FISMA [152] de NIST à l'informatique en nuage. Cette solution ne considère pas les besoins du client et elle retourne un rang qualitatif de sécurité (haut, moyen, faible).
Luna et al. [113, 114, 116, 155] ont proposé plusieurs méthodes d'évaluation et classement des offres des fournisseurs en terme de sécurité.
Une méthode pour quantifier la qualité de sécurité des politiques a été proposée dans dans [39] et appliquée dans [114] afin de quantifier le niveau de sécurité des fournisseurs à partir du registre STAR [45]. Les auteurs proposent dans [113] et [114] de mesurer le niveau de sécurité des fournisseurs de l'informatique en nuage. Ils améliorent leurs propositions dans [116] en permettant aux clients d'attribuer des poids aux critères de sécurité selon leur importance.
Dans [155] est proposé un framework de sélection des offres des fournisseurs qui répondent aux besoins du client en matière de sécurité. L'évaluation du niveau de sécurité peut être quantitative en attribuant une valeur numérique à chaque contrôle de sécurité (selon le niveau d'importance, par exemple : selon la taille de la clé de chiffrement). L'évaluation peut aussi être qualitative en attribuant des poids à chaque contrôle de sécurité selon leur importance (extrêmement important, moyennement important ou non-requis). Les auteurs utilisent les offres publiées dans le registre STAR comme un cas d'usage.

Confiance Un système de gestion de confiance pour le nuage informatique a été proposé dans [73]. Ce système permet de mesurer la fiabilité des fournisseurs en se basant sur les capacités extraites du dépôt CAIQ.
Goettelmann et al. [66] présentent une approche pour mesurer le niveau de risque de sécurité des fournisseurs avant de distribuer l'exécution des processus sur des fournisseurs multiples. Le niveau de risque est une fonction entre les vulnérabilités de sécurité, les menaces et leurs impacts sur les données clients. Ils utilisent le dépôt CAIQ pour extraire les informations concernant les offres de sécurité des fournisseurs.

Négociation Un mécanisme de négociation des besoins de sécurité entre les fournisseurs et le client a été proposé dans [106]. L'objectif était de proposer un langage sémantique commun qui permet de comparer les besoins du client exprimés dans un langage de haut niveau avec les offres des fournisseurs spécifiées dans un langage technique.
Un protocole de négociation basée sur une évaluation quantitative du niveau de sécurité des fournisseurs, est proposé dans [62].
Dans [105], les auteurs comparent deux politiques de contrôle d'accès et mesurent leur similarité. Ils pensent utiliser leur algorithme dans un processus de négociation des

services entre les fournisseurs et les clients de l'informatique en nuage. Selon [105], leur algorithme pourrait être utilisé pour filtrer les fournisseurs avec les mêmes niveaux de sécurité.

3.4.4.1 Discussion

Nous distinguons différents types de travaux de la littérature. Des travaux qui proposent des mécanismes de sécurité dans l'informatique en nuage et la plupart exigent des modifications matérielles et logicielles au niveau de l'architecture des fournisseurs de l'informatique en nuage. De nombreux travaux ajoutent des couches logicielles ou des API afin de renforcer la sécurité. Cela rend la gestion de l'informatique en nuage plus lourde et peut paradoxalement l'exposer à plus de vulnérabilités. D'autres travaux essayent d'adapter des normes de sécurité existantes aux exigences de l'informatique en nuage.
Beaucoup de travaux s'intéressent seulement à l'évaluation quantitative des offres de sécurité des fournisseurs. Plusieurs se basent sur les rapports des fournisseurs publiés dans le registre STAR. Très peu de chercheurs développent leur propre mécanisme d'évaluation de risque des offres de sécurité des fournisseurs. L'inconvénient de ces approches est que toutes les propriétés de sécurité doivent être mesurées quantitativement. Même les travaux qui proposent une évaluation qualitative, ne considèrent que des valeurs statiques, telles que « important », « moyennement important » ou « non-requis ». Cela n'est pas suffisant pour juger de la robustesse et du niveau de sécurité d'une offre d'un fournisseur. Plusieurs travaux évaluent et classent les offres des fournisseurs en ignorant complément les besoins du client, or une offre d'un fournisseur qui correspond à un client pourrait ne pas correspondre à un autre.
Malgré tous les efforts fournis ces dernières années, Luna et al. insistent toujours [115] sur l'importance d'intégrer les contrats secSLA (security SLA) dans les contrats SLA signés entre les fournisseurs et les clients. Selon [133], les contrats SLA ne couvrent pas certains critères détaillés de sécurité avant 2016, comme le suivi continu de sécurité.

3.5 Assurance de la sécurité au moyen des méthodes formelles

Principalement, les méthodes de vérification formelle sont utilisées pour vérifier les politiques de contrôle d'accès. Plusieurs travaux analysent les politiques de contrôle d'accès en utilisant des outils pour vérification formelle afin de détecter les erreurs de spécification. Dans [67, 68], les auteurs vérifient des propriétés de sécurité des politiques RBAC dans des environnements collaboratifs. Ils utilisent l'outil pour vérification formelle (*model checker*) NuSMV [1]. Ils considèrent des politiques RBAC qui supportent la hiérarchie des rôles. L'héritage est possible entre différents domaines.

Les propriétés de sécurité vérifiées sont : la séparation statique des tâches (SOD - *separation of duties*) et l'élévation des privilèges (*privilege escalation*). L'inconvénient de leur approche est qu'elle manque d'abstraction. En effet, ils effectuent les vérifications sur un cas d'usage bien limité. Il faut redéfinir les propriétés à vérifier pour chaque politique. De plus, ils devraient considérer d'autres propriétés de sécurité concernant les environnements collaboratifs.

Dans [145, 144], les auteurs utilisent le langage et l'analyseur Alloy pour décrire des politiques de contrôle d'accès et vérifier qu'elles sont bien implémentées. Ils considèrent des politiques de contrôle d'accès RBAC. Ils définissent les autorisations en fonction des rôles des utilisateurs. Ensuite, ils vérifient différentes propriétés de séparations des tâches. Toahchoodee et al. [159, 160], reprennent le même concept et développent plus la délégation des permissions tout en considérant des contraintes de temps et de localisation. Ils utilisent Alloy pour vérifier des propriétés de séparation des tâches.

Alloy a aussi été utilisé dans [36] afin de vérifier la sureté et la sécurité d'une architecture d'un système lors de la conception. L'objectif est d'aider les ingénieurs concepteurs à détecter les erreurs d'une architecture d'un système. Un système aéroponique embarqué est utilisé comme un cas d'usage. Les auteurs spécifient dans un premier temps à quoi doit correspondre un système fiable. Ils définissent les types d'attaques que le système peut subir, par exemple un signal satellite est brouillé. Ils vérifient ensuite la cohérence de ces spécifications en utilisant Alloy.

Bleikertz et al. [32] proposent un langage formel pour exprimer des objectifs de sécurité de haut niveau tels que : la gestion d'isolation dans les environnements d'informatique en nuage. Ils introduisent dans [33] une approche d'analyse des environnements virtualisés statiques de l'informatique en nuage. Ils vérifient que le déploiement d'un système dans le nuage informatique est correct. Ce déploiement représente l'état actuel de la configuration de l'infrastructure de virtualisation.

3.6 Analyse et synthèse

Notre solution propose plusieurs contributions originales par rapport à la littérature. D'abord, nous tenons en compte les besoins exprimés par le client. Le client a la possibilité d'exprimer les besoins en terme de performance ainsi qu'en terme de sécurité (chapitre 4). Ensuite, nous utilisons des méthodes formelles pour analyser les besoins du client et vérifier que les politiques de sécurité sont bien implémentées et ne contiennent pas de conflit. Nous vérifions la possibilité du déploiement des ressources exigées par le client dans le nuage informatique et d'assurance des propriétés de sécurité requises avant le déploiement. Nous construisons une configuration de placement des ressources dans le nuage informatique qui satisfait tous les besoins et les contraintes du client (chapitre 5) et nous la retournons au client qui peut accepter

ou demander une autre solution. De plus, nous permettons au client de décrire la politique de contrôle d'accès à ses ressources par les utilisateurs qu'il va devoir gérer. Nous vérifions que les politiques de contrôle d'accès ne contiennent pas de conflit ou erreur pour éviter des violations de sécurité une fois les ressources déployées et accessibles sur le nuage informatiques (chapitre 6). Notre solution a pour objectif premier de donner une assurance et une confiance au client qui souhaite adopter le nuage informatique. Le client est rassuré avec le système qu'il a décrit. Ses besoins fonctionnels techniques et ses politiques de sécurité ne contiennent pas de conflit et peuvent être assurés par des offres de fournisseurs. Autrement dit, notre démarche rassure le client dans l'exacte obtention de ce qu'il a demandé, que ce soit au niveau de ressource qu'au niveau de sécurité.
Les besoins techniques du client et les exigences de sécurité doivent être intégrés dans les contrats SLA établis entre le client, le courtier et les fournisseurs de service (section 7.2). Cela est considéré comme un engagement de toutes les parties à respecter les consignes du contrat. Des pénalités sont aussi spécifiées dans les contrats pour gérer les violations de contrats.

3.7 Conclusion

Dans ce chapitre, nous avons présenté la technologie de l'informatique en nuage. Nous avons présenté ses caractéristiques, ses avantages et ses inconvénients. Parmi ces dernier, la sécurité est le point le plus important qui représente un frein majeur à l'adoption de l'informatique en nuage. Par la suite, nous avons présenté le concept de courtage informatique. Il s'agit des solutions qui permettent de comparer les offres des fournisseurs de l'informatique en nuage. Plusieurs courtiers ont été présentés dans ce chapitre. Nous avons souligné le manque de solutions qui prennent en compte la sécurité pour la sélection des offres des fournisseurs. Plusieurs travaux de recherche ont profité des projets STAR, CCM et CAIQ de la CSA afin de proposer des solutions de comparaison et de classement des offres de sécurité des fournisseurs. En effet, la CSA a proposé ces initiatives afin d'aider les fournisseurs à documenter leur offres et mécanismes de sécurité et à les rendre transparents. Les clients bénéficient aussi de ces initiatives car ils récupèrent les contrôles de sécurité proposés par chaque fournisseurs ce qui leur permet de comparer et de choisir l'offre qui répond le mieux à leur exigences de sécurité.
À notre connaissance, tous les travaux qui existent jusqu'à maintenant, ne font que comparer les offres de sécurité publiées par les fournisseurs dans les dépôts de CSA. L'avantage est que cette méthode pousse les fournisseurs à s'engager pour fournir exactement le niveau de sécurité qu'ils publient dans leurs offres. En effet, tout engagement en terme d'offre de sécurité doit être mentionné dans les contrats SLA établis entre le fournisseur et le client. De plus, ces offres sont principalement basées sur les

normes telles que ISO 27100. Par contre, ces questionnaires ne permettent de décrire que des propriétés de sécurité de haut niveau, par exemple : « Est ce que vous fournissez un moyen d'isolation de MV ». Les réponses des fournisseurs consistent à citer les normes appliquées afin d'assurer cette propriété. Le mécanisme de sécurité appliqué afin d'assurer cette propriété n'est pas clairement mentionné. De plus, le mécanisme utilisé pour assurer un contrôle de sécurité peut être différent selon les cas d'usage.
Nous considérons que ces initiatives sont déjà un bon point pour commencer à considérer la sécurité dans le processus de sélection des offres parmi plusieurs. Cela permet aux clients d'exprimer leurs exigences de sécurité et les encourage à migrer leurs services et données vers l'informatique en nuage. Ces initiatives poussent les fournisseurs à mieux documenter leurs offres et mécanismes de sécurité et à les rendre transparents. Principalement pour des raisons économiques et pour soigner leur réputation, les fournisseurs s'engagent à fournir les meilleures solutions pour assurer la sécurité des services et données des clients.
Dans cette thèse, nous proposons un processus de courtage des services de l'informatique en nuage, tout en considérant des critères de sécurité. Dans les chapitre 4, chapitre 5 et chapitre 6, nous présentons notre solution de courtage. Nous considérons des critères de sécurité personnalisés et de plus bas niveau, tel que l'isolation des MV, la confidentialité et l'intégrité des MV. Nous permettons au client de décrire l'architecture de son système et de décrire les échanges d'information entre les ressources de son système. Le client spécifie des contraintes de sécurité et de placement qui sont considérées afin de choisir leur placement chez les fournisseurs. Les travaux de la littérature n'assurent pas que les propriétés exigées par le client sont bien appliquées dans l'informatique en nuage. Cependant, le courtier que nous proposons utilise la vérification formelle afin de détecter les erreurs de spécification tôt avant le déploiement et aussi pour vérifier que les exigences du client sont assurées si le placement choisi est appliqué.

4 Spécification des besoins du client et offres des fournisseurs

Sommaire

4.1 Introduction . . . 45
4.2 Offres des fournisseurs . . . 46
4.3 Besoins du client . . . 48
4.3.1 Besoins fonctionnels . . . 48
4.3.2 Besoins non-fonctionnels . . . 48
4.4 Spécifications en langage Alloy . . . 50
4.4.1 Offres des fournisseurs . . . 52
4.4.2 Besoins du client . . . 56
4.5 Détection des inconsistances coté client . . . 63
4.5.1 Conflits des spécifications fonctionnelles . . . 64
4.5.2 Conflits des spécifications non-fonctionnelles . . . 65
4.6 Conclusion . . . 74

4.1 Introduction

Le taux d'adoption des architectures informatiques en nuage par des clients augmente grâce aux avantages qu'elles présentent. On cite principalement la haute disponibilité et l'extensibilité des services et le non-investissement dans du matériel et de la maintenance qui nécessite de la maîtrise et coûte cher. Un client de nuage informatique peut représenter une entreprise ou un particulier. Un client contacte un fournisseur d'accès à des services nuagiques pour externaliser ou mettre en place son système d'information. Devant la demande croissante des clients et la multiplicité des fournisseurs et des services, des acteurs intermédiaires appelés courtiers (*broker* en anglais) apparaissent. Un courtier est une vitrine intelligente des nuages informatiques. Il récupère les offres de plusieurs fournisseurs et aide le client à faire le choix des fournisseurs.

Les courtiers se développent de plus en plus car les offres des fournisseurs de nuage informatique augmentent rapidement [147, 131]. Il est très difficile pour le client de parcourir toutes les offres existantes afin de choisir la plus adaptée à ses besoins. D'un

autre côté, les fournisseurs adoptent les courtiers car ils assurent la publicité de leurs offres et leur permettent d'avoir plus de clients. De plus, grâce aux courtiers, les fournisseurs peuvent participer à des fédérations de nuages informatiques sans fournir d'effort supplémentaire.
Le courtier cherche les offres qui peuvent satisfaire les besoins du client. Pour ce faire, le client définit ses besoins de services, de performances de calcul et de capacité de stockage. De plus, le client définit des critères de placement de ses services dans le nuage informatique ainsi que des propriétés de sécurité afin de protéger ses données et services.
Dans ce chapitre, nous allons présenter les offres des fournisseurs de services dans le nuage informatique. Nous présentons aussi les besoins d'un client de nuage informatique. Ces besoins sont de deux natures : fonctionnelles et non-fonctionnelles. Les besoins fonctionnels décrivent les ressources nécessaires comme les machines virtuelles, les performances de calcul ou les capacités de stockage. Les besoins non-fonctionnels décrivent des contraintes entre les ressources comme les propriétés de sécurité. Nous décrivons dans un premier temps les offres et les besoins à l'aide du langage naturel. Cela correspond à la façon naturelle avec laquelle les fournisseurs exprimeraient leurs offres et les clients, leurs besoins.
Ensuite nous expliquons comment spécifier formellement ces descriptions des offres et des besoins, en utilisant le langage Alloy.
Nous commençons dans la deuxième section par décrire les offres que les fournisseurs présentent au courtier. La troisième section présente ce qu'un client peut demander du courtier. Le client décrit un système constitué de ressources à allouer chez les fournisseurs. Il spécifie des propriétés de sécurité à assurer pour protéger ses données dans le nuage informatique. Les offres des fournisseurs et les besoins du client sont spécifiés en utilisant le langage formel Alloy dans la quatrième section. Les besoins du client sont analysés et les conflits de spécifications sont signalés au client, cela fait l'objet de la cinquième section.

4.2 Offres des fournisseurs

Les fournisseurs décrivent leurs offres et les publient afin de les rendre accessibles au courtier. Dans cette thèse nous considérons des fournisseurs du nuage informatique de niveau IaaS. Le nuage informatique de type IaaS offre des performances de calcul et des capacités de stockage de haute disponibilité et extensibilité qui peuvent être ajustées à la demande du client. Il peut aussi offrir des mécanismes de sécurité tels que des gardiens (pare-feux) [122].
Le courtier doit être un tiers de confiance. Des relations de confiance le relient aux clients mais aussi aux fournisseurs. En effet, Le fournisseur donne une description de ses serveurs physiques et leur localisation, de ses grappes et réseaux virtuels. Le

fournisseur décrit le réseau d'interconnexion de ses grappes virtuelles ainsi que l'interconnexion de ces dernières à l'extérieur. Les composants de base des offres des fournisseurs consistent en les éléments suivants :

- Un ensemble de **nœuds physiques** (ou hôtes physiques) qui consistent en du matériel de serveurs et de stockage. Ils hébergent les ressources virtuelles (nœuds de calcul et de stockage). Les nœuds physiques sont inter-connectés par des réseaux. Chaque nœud dispose d'une quantité de ressources physiques (capacité CPU et quantité de mémoire).
- Un ensemble de **grappes** qui rassemblent des machines virtuelles (nœuds de calcul) et des nœuds de stockage. Pour simplifier, nous considérons qu'une grappe est composée de plusieurs nœuds physiques.
- Un ensemble de **réseaux physiques** qui permettent d'inter-connecter les grappes et les nœuds physiques.
- Un ensemble de **gardiens** qui sont placés entre les grappes. Ils peuvent représenter des systèmes de prévention d'intrusion, des systèmes de détection d'intrusion ou des systèmes de filtrage comme les pare-feux.
- Un ensemble de **modèles de** MV qui sont en fait des configurations de machines virtuelles (MV) ainsi que leurs caractéristiques telles que : la capacité de CPU, la taille de mémoire et la capacité de stockage.
- Des **offres de sécurité** de haut niveau qui peuvent par exemple représenter la localisation géographique des nœuds physiques, des mécanismes de chiffrement et de certification utilisés pour protéger les communications et les données.
- Un ensemble de **conflits** entre les grappes, les nœuds physiques ou les fournisseurs. Cette relation de conflit peut être spécifiée par les fournisseurs dans le but de limiter les échanges entre des serveurs d'hébergement et des données sensibles par exemple et protéger les données de domaines critiques.
  Par exemple des grappes hébergeant des données de gouvernement ou de défense devraient être en conflit avec toute autre grappe.
  Un fournisseur de service pourrait aussi par exemple spécifier que quelques grappes localisées dans la région européenne devraient être en conflit avec celles localisées dans une région autre que l'Europe. Plusieurs propriétés peuvent être exprimées en fonction de cette relation de conflit par exemple, les fournisseurs/-hôtes/grappes qui sont en conflit ne doivent pas participer dans une même fédération de nuages informatiques.

En plus des descriptions des offres des fournisseurs, nous considérons la préexistence de deux types de certification tierce partie.

La certification QdS assure que les offres de Qualité de Service (QdS) ou (*Quality of Service - QoS* en anglais) d'un fournisseur sont conformes à ses descriptions [137]. Le tiers de confiance effectue un ensemble de tests pour vérifier les paramètres de QdS

décrits comme la disponibilité de service, la bande passante et les prix. Le tiers de confiance délivre un certificat de QdS pour ce fournisseur.

La certification CSA STAR mesure la capacité d'un fournisseur à assurer la sécurité des services dans le nuage informatique. La CSA évalue l'efficacité d'un fournisseur à maîtriser les différents types de sécurité [48] en se basant sur l'audit de la matrice de contrôle du nuage informatique [49] (voir sous-section 3.4.3.1).

4.3 Besoins du client

Le client décrit le système qu'il souhaite déployer dans le nuage informatique. Les besoins spécifiés par les clients sont classifiés en deux catégories : besoins fonctionnels et besoins non-fonctionnels.

4.3.1 Besoins fonctionnels

Le client définit les besoins fonctionnels du système à héberger dans le nuage informatique. Il décrit les besoins des ressources dont son système aura besoin. Cela consiste en tout ce qui est nombre d'instances de ressources virtuelles ainsi qu'en performances de calcul, capacités de stockage et quantités de mémoire souhaitées. Le client décrit alors les éléments suivants :

- machines virtuelles (nœuds de calcul) et leurs caractéristiques telles que : capacité de CPU, quantité de mémoire, capacité de stockage et bande passante,
- nombre total de machines virtuelles.

4.3.2 Besoins non-fonctionnels

Les besoins non-fonctionnels sont constitués de propriétés de sécurité et de contraintes de placement. Les propriétés de sécurité sont spécifiées pour protéger les ressources du client une fois déployées dans le nuage informatique. Le courtier considère les contraintes de placement afin de placer les ressources dans le nuage informatique de façon à respecter les besoins du client. Nous détaillons par la suite quelques exemples de besoins non-fonctionnels.

4.3.2.1 Propriétés de sécurité

Les propriétés de sécurité définissent des contraintes de sécurité spécifiques au système du client. Elles sont représentées par des relations entre les MV. Cela permet de décrire des propriétés de sécurité et de gérer les MV du client de façon plus simple. Par exemple, autoriser, interdire ou contrôler les communications entre les MV en fonction des relations qui les relient. Nous traduisons ces relations en des propriétés

de flux d'information. Nous pouvons ainsi vérifier la satisfiablité des propriétés de sécurité en vérifiant si un flux d'information direct ou transitif existe ou est absent entre deux MV. Nous utilisons des moyens de protection des réseaux comme les pare-feux que nous appelons dans ce document des gardiens. Les propriétés de sécurité que le client peut définir pour son système sont :

Collaboration : Le client peut définir des collaborations entre MV. Deux MV définies en collaboration sont alors autorisées à partager des données ou des ressources. La connectivité des MV en collaboration doit être assurée. Des flux d'information bidirectionnels doivent être possibles entre deux MV en collaboration.

Concurrence (conflit) : Le client peut définir des concurrences entre MV. Deux MV définies en concurrence ne sont pas autorisées à partager des informations et des ressources. L'information ne doit pas transiter entre ces MV par des flux directs ou transitifs. Le client peut par exemple tolérer les liens transitifs reliant les MV mais il faut que les liens soient contrôlés par des gardiens.

Flux unidirectionnels : Le client peut définir des flux à sens unique entre les MV. Un flux représente un transfert d'information vers une MV (flux d'écriture) ou une récupération d'information à partir d'une MV (flux de lecture).
Nous ne faisons pas de différence entre les flux de lecture et les flux d'écriture. Nous considérons les deux types comme un flux d'information à sens unique qui peut représenter l'un des deux types. En effet, notre objectif est de montrer comment il est possible de spécifier les besoins du client et les analyser de façon rigoureuse. Donc, dans cette thèse nous ne détaillons pas tous les types de flux possibles dans un système et nous pensons que notre démarche peut être étendue pour couvrir ce qui n'est pas traité dans ce travail.

Isolation : le client peut décrire que quelques MV doivent être complètement isolées des autres (par exemple les MV avec un niveau de sécurité top-secret, défini ci-après). Toute MV isolée ne doit pas communiquer avec d'autres MV, à moins d'être reliée avec d'autres MV via des flux contrôlés par des gardiens afin de bloquer les flux indésirables.

Confidentialité/Intégrité : Le client peut exiger la confidentialité (resp. l'intégrité) de quelques MV en interdisant tout flux d'information sortant de ces MV (resp. entrant à ces MV). Nous pouvons exprimer la confidentialité (resp. l'intégrité) d'une MV $\mathcal{A}$ par rapport à une autre MV $\mathcal{B}$. Cela revient à assurer

qu'il n'existe pas de flux direct ou transitif qui provient de la MV $\mathcal{A}$ vers la MV $\mathcal{B}$ (resp. de la MV $\mathcal{B}$ vers la MV $\mathcal{A}$).

Niveau de sécurité des MV : Le client peut étiqueter les MV par des niveaux de sécurité : public, normal (semi-public) et privé (secret et top-secret). L'objectif de cette étiquette est de simplifier la définition des autres propriétés de sécurité. Par exemple, au lieu de définir plusieurs fois une propriété de concurrence entre de nombreuses MV, le client peut définir une seule fois la propriété entre deux MV qui ont le niveau de sécurité secret.

4.3.2.2 Contraintes de placement

Le client peut définir des contraintes de placement de ses ressources dans le nuage informatique. Il définit ses contraintes en fonction des propriétés de sécurité spécifiées ci-dessus. Les contraintes qui peuvent être exprimées sont :

Groupement : Héberger les MV en collaboration sur la même grappe ou sur le même nœud physique ou chez le même fournisseur de service. Cela est utile pour partager les mêmes espaces de stockage et pour augmenter la bande passante ou le niveau de sécurité.

Séparation : Héberger les MV séparément sur différentes grappes, différents nœuds physiques ou différents fournisseurs. Cela permet d'éviter le risque de fuite de données de façon malicieuse ou non aux MV voisines. Le client peut tolérer un placement de MV dans une même grappe ou chez un même fournisseur mais à condition que les communications soient contrôlées par des gardiens. Nous allons développer cette idée dans le chapitre 5.

Isolation : Une MV peut être isolées physiquement dans des nœuds physiques ou virtuellement dans des grappes virtuelles. Par exemple, allouer la MV seule sur un nœud physique préalablement vide. Cela est utile quand le client veut s'assurer de la confidentialité et l'intégrité de ses données sensibles. Ainsi, le risque des attaques par corésidence [138] est réduit. En effet, une MV qui est hébergée par la même machine physique qu'une autre MV, peut observer ses comportements et en profiter pour réaliser des attaques.

4.4 Spécifications en langage Alloy

Dans cette section, nous définissons un modèle formel pour spécifier les offres des fournisseurs du nuage informatique et les besoins du client du nuage informatique. Ce

modèle formel est basé sur le langage Alloy. Nous utilisons ce langage car il permet d'exprimer avec souplesse les systèmes nuagiques et il est supporté par l'analyseur Alloy. Ce dernier permet une analyse syntaxique et sémantique des systèmes et vérifie la satisfiabilité des spécifications exprimées.
Les fournisseurs et les clients définissent leurs offres et besoins sur une interface utilisateur. Nous récupérons ces données exprimées par les fournisseurs et les clients et nous les traduisons automatiquement en spécifications Alloy.
Nous décrivons un système d'un client (des besoins) ou un système d'un fournisseur (une offre) dans des modèles Alloy.
Nous utilisons des signatures et des faits Alloy pour exprimer les offres et les besoins. Les signatures sont utilisées pour définir les types de ressources et les types des caractéristiques ainsi que leurs instances. Les faits Alloy sont des formules qui représentent des propriétés et qui doivent être toujours vraies dans le système.
Nous utilisons aussi des prédicats et des assertions pour exprimer des propriétés à vérifier par rapport au système global décrit dans les modèles Alloy.

Attributs numériques

On définit d'abord les MV qui sont caractérisées par plusieurs attributs. Dans nos travaux nous considérons des critères non numériques tels que le système d'exploitation de la MV, noté OS (*Operating System*).
Nous ne considérons pas des critères numériques quantitatifs tels que la capacité de stockage ou la quantité de mémoire qu'un fournisseur peut fournir. En effet, l'analyseur Alloy ne traite pas de façon simple et naturelle les quantités numériques. *Alloy n'étant pas destiné au calcul numérique, il n'implémente pas toute l'arithmétique des entiers* [128]. Par contre, nous considérons des critères numériques tels que la vitesse de CPU. Par exemple, nous attribuons à ce dernier les valeurs suivantes CPU17, CPU24 et CPU27 pour représenter les vitesses 1.7GHZ, 2.4GHZ et 2.7GHZ respectivement. Donc nous proposons de représenter les valeurs numériques par des valeurs constantes non-numériques définies comme des signatures. C'est le cas des valeurs de l'attribut CPU. Nous pouvons appliquer la même procédure pour des attributs tels que la quantité de mémoire en définissant par exemple les signatures 4GO, 8GO et 16GO comme valeurs d'attribut.

Une autre solution consiste à séparer l'étape de mise en correspondance en deux parties (voir figure 1.1, page 4). Une partie consiste à la mise en correspondance des attributs numériques en utilisant un solveur de contraintes. Une autre partie est consacrée à la mise en correspondance des attributs non-numériques en utilisant Alloy.
Dans cette thèse, nous avons considéré la première solution qui consiste à définir les valeurs numériques comme des signatures. Le type de MV ainsi que ses caractéristiques sont définis par les signatures Alloy dans Listing 4.1.
Un modèle abstrait du système client est constitué d'un ensemble de MV. Chacune

se caractérise par une vitesses de calcul, un système d'exploitation et une localisation géographique. Nous définissions aussi quelques instances pour chaque caractéristique pour illustrer notre démarche.

Listing 4.1 –

```
abstract sig VM {
        vm_cpuSpeed: one CPU,
        vm_os: one OS,
        vm_location: lone Location
        }
abstract sig OS{}
abstract sig CPU{}
abstract sig Location{}

one sig linux, windows extends OS{}
one sig cpu2, cpu24, cpu3, cpu34 extends CPU{}
one sig europe, usa extends Location{}
```

Les machines virtuelles MV (nœuds de calcul) sont définies par la signature `VM`. Chaque `VM` est caractérisée par trois attributs (appelés en Alloy des relations). Le premier attribut `vm_cpuSpeed` de type `CPU` est une unique vitesse de calcul. Le deuxième attribut `vm_os` de type `OS` est un unique système d'exploitation à installer sur la MV. `CPU` et `OS` sont aussi définis comme des signatures ainsi que leurs instances respectives `cpu2`, `cpu24`, `cpu3`, `cpu34` pour `CPU` et `linux`, `windows` pour `OS`. Le troisième attribut `vm_location` représente la région géographique où la MV sera allouée. Le client spécifie au plus une `Location` pour l'attribut `vm_location`. Les valeurs possibles sont `europe` et `usa`.

4.4.1 Offres des fournisseurs

Le fournisseur exprime ses offres de nuage informatique. Il s'agit des ressources et services qu'il peut fournir aux clients et des garanties de sécurité qu'il peut assurer pour protéger les données de ses clients.
Ces spécifications d'offres sont publiées auprès du courtier de nuage informatique. Le courtier utilise ces informations pour trouver les correspondances avec les besoins du client.
Dans cette sous-section, nous traduisons les offres décrites dans la section 4.2 en langage Alloy. Dans un premier temps, nous présentons le modèle de base d'une offre d'un fournisseur de nuage informatique. Ensuite, nous illustrons avec un exemple une offre spécifiée en Alloy.

4.4.1.1 Modèle de base d'une offre

Les spécifications Alloy du Listing 4.2 présentent le modèle de base d'une offre d'un fournisseur.
Un fournisseur propose un ensemble de grappes (`Cluster`). Chaque grappe rassemble un ensemble de MV qui sont hébergées par un ensemble d'hôtes (nœuds) physiques `PhysicalNode`. Un nœud physique est caractérisé par plusieurs attributs tels que `cpuSpeed`, `os` et `location`. Ces attributs représentent les vitesses de CPU, les systèmes d'exploitations OS que le nœud physique peut fournir avec les MV qu'il fait tourner. Chaque nœud doit proposer au moins une valeur de ces deux attributs. L'attribut `Location` représente la région géographique dans laquelle ce nœud est installé. Le quatrième attribut `vms` est l'ensemble de MV allouées sur le nœud physique. Chaque nœud peut ne pas héberger de MV comme il peut en héberger plusieurs, d'où l'utilisation du quantificateur *set* qui signifie ici un ensemble de zéro ou plusieurs MV. Ces attributs des nœuds physiques représentent des modèles de MV que chaque nœud peut proposer aux clients. Une MV d'un client ne peut être allouée sur un nœud physique que si ses caractéristiques peuvent être fournies par ce nœud physique. Nous utilisons cette contrainte de placement dans le chapitre 5.
D'autres attributs peuvent être considérés dans l'offre des fournisseurs tels que la capacité en nombre de MV à héberger. Nous n'avons pas considéré ce type de contraintes numériques dans nos travaux car Alloy n'est pas destiné à faire des calculs numériques.

Listing 4.2 –

```
abstract sig Provider{clusters: some Cluster}
abstract sig Cluster {host: some PhysicalNode}
abstract sig PhysicalNode{
        cpuSpeed: some CPU,
        os: some OS,
        location: one Location,
        vms: set VM
        }
```

On définit aussi un ensemble de relations entre grappes. Elles représentent les réseaux qui interconnectent les grappes et les gardiens qui contrôlent les communications entre grappes. La signature du Listing 4.3 `cluster_relation` rassemble toutes les relations `link` et `guardian`. La relation `link` consiste en tous les couples de grappes interconnectées par un réseau. La relation `guardian` consiste en tous les couples de grappes entre lesquels un gardien (pare-feu par exemple) est placé, c'est-à-dire `guardian` est un lien contrôlé qui relie deux grappes.
Cela représente la vision globale que le courtier a du nuage informatique. En effet chaque fournisseur a connaissance des réseaux et des gardiens qui interconnectent ses

propres grappes et ceux qui interconnectent ses grappes avec les autres fournisseurs du nuage informatique. À partir des offres que les fournisseurs proposent au courtier, ce dernier peut constituer une vue globale des nuages informatiques sur lesquels il intervient.
`conflict` est une relation entre deux grappes. Cette donnée est exprimée par les fournisseurs de nuage informatique. Elle exprime le fait que deux grappes sont en conflit d'intérêt et cela induit des conséquences sur le placement des MV des clients dans le chapitre 5. Par exemple, les grappes qui sont en conflit ne peuvent pas participer à une même collaboration.

Listing 4.3 –

```
one sig cluster_relation{
   link : set Cluster−>Cluster,
  guardian: set Cluster−>Cluster,
   conflict : set Cluster−>Cluster
  }
```

4.4.1.2 Exemple d'offre d'un fournisseur

La spécification Alloy du Listing 4.4 présente un exemple d'offre que peut faire un fournisseur de nuage informatique.
Cette spécification est traduite automatiquement à partir des offres que le fournisseur a saisies sur une interface utilisateur.
Dans cet exemple, on définit le fournisseur `amazon`. Ce dernier déclare qu'il détient trois grappes `amazonZone1`, `amazonZone2` et `amazonZone3`, chacune rassemble un ensemble de nœuds physiques. Par exemple `amazonZone1` rassemble `amazonZone1node1` et `amazonZone1node2`. Chaque nœud est caractérisé par un ensemble de caractéristiques fonctionnelles qu'il peut fournir à l'allocation de MV. Les deux nœuds physiques `amazonZone1node1` et `amazonZone1node2` assurent les mêmes caractéristiques de `CPU`, `OS` et `Location`. Tous les deux peuvent fournir des MV avec une vitesse de CPU : `cpu2`, `cpu24`, `cpu3` ou `cpu34`, un OS `windows` ou `linux`. Les deux nœuds sont installés en région `européenne`.

Listing 4.4 –

```
one sig amazon extends Provider{}{
        clusters  = amazonZone1 + amazonZone2 + amazonZone3
        }
one sig amazonZone1 extends Cluster{}{
        host = amazonZone1node1 + amazonZone1node2
        }
```

```
one sig amazonZone1node1, amazonZone1node2 extends PhysicalNode{}{
        cpuSpeed = cpu2 + cpu24 + cpu3 + cpu34
        os = linux + windows
        location = europe
        }
```

Le fait du Listing 4.5 représente les relations que le fournisseur `amazon` peut définir entre les grappes qu'il propose ou entre ses grappes et des grappes des autres fournisseurs. Il s'agit ici, d'un lien entre les grappes `amazonZone1` et `amazonZone2`, un lien contrôlé par un gardien entre les grappes `amazonZone2` et `amazonZone3`. Le fournisseur `amazon` peut aussi définir un conflit entre sa grappe `amazonZone1` et une grappe `orangeZone1`. Nous supposons que `orangeZone1` est une des grappes d'un fournisseur nommé `orange`.

Listing 4.5 –

```
fact fact_cluster_relation {
  cluster_relation . link = amazonZone1->amazonZone2
  cluster_relation .guardian= amazonZone2->amazonZone3
  cluster_relation . conflict =amazonZone1->orangeZone1
}
```

Le courtier conçoit une vue (Figure 4.1) des offres qu'il reçoit des fournisseurs. Dans la figure 4.1 retournée par l'analyseur Alloy, nous représentons cette vue que le courtier a. Nous ne représentons pas les nœuds physiques et leurs caractéristiques fonctionnelles pour ne pas encombrer la figure.

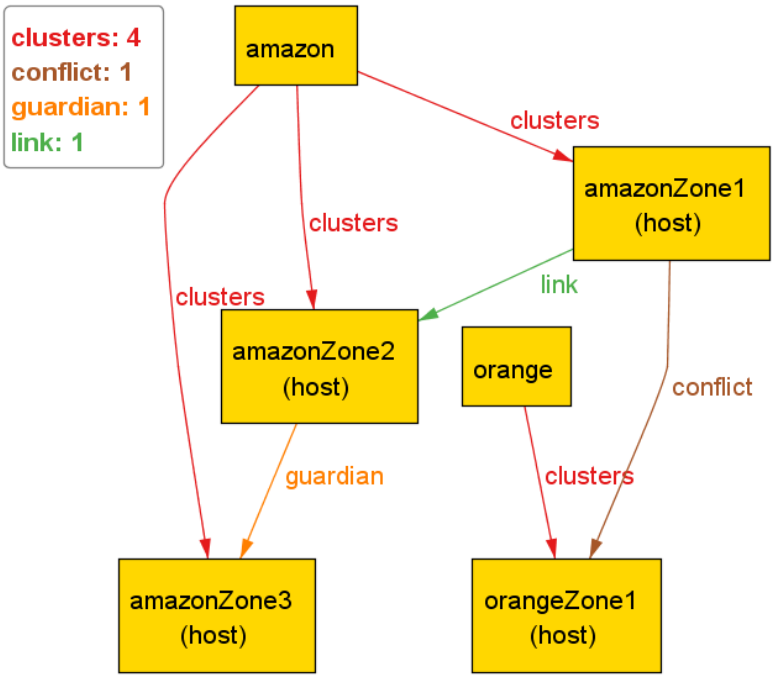

FIGURE 4.1 – *Exemple des offres des fournisseurs.*

4.4.2 Besoins du client

Un client qui souhaite se procurer des ressources de calcul dans le nuage informatique, se dirige vers un courtier, décrit ses besoins afin d'avoir une proposition qui réponde au mieux à ses besoins.
La requête d'un client auprès d'un courtier contient les deux types de besoins : fonctionnels et non-fonctionnels. Dans cette sous-section, nous présentons le modèle de base d'une requête d'un client. Nous utilisons Alloy pour décrire les besoins fonctionnels et non-fonctionnels du client. Enfin, nous illustrons avec un exemple les besoins d'un client.

4.4.2.1 Besoins fonctionnels

La spécification Alloy du Listing 4.6 représente le modèle de base d'une requête d'un client.
Un client demande un ensemble de MV. Il précise pour chaque machine ses caractéristiques fonctionnelles (telles que OS et CPU) comme présenté dans Listing 4.1.

Listing 4.6 –

```
abstract sig Customer{
  customer_vms: some VM
  }
```

4.4.2.2 Besoins non-fonctionnels

Dans cette partie, nous présentons un modèle Alloy qui permet de spécifier les besoins non-fonctionnels du client. Ces besoins peuvent représenter des propriétés de sécurité, des contraintes de placement ou des contraintes d'interconnexion. Les contraintes d'interconnexion peuvent être considérées comme besoins fonctionnels car les interconnexions assurent le transfert d'information entre les MV. Cependant, dans notre cas d'usage, nous considérons l'aspect non-fonctionnel. En effet, on ne tolère pas l'existence d'un lien d'interconnexion indirect entre deux MV en concurrence, par exemple. Cet exemple représente une contrainte non-fonctionnelle.
Nous traduisons les besoins du client en spécifications Alloy pour pourvoir les analyser et vérifier leur satisfiabilité ou mettre en évidence les inconsistances des besoins du client.

Propriétés de sécurité : Le client décrit des relations entre les ressources demandées telles que les collaborations et les concurrences. Il spécifie les flux à sens unique entre les ressources et définit aussi les ressources à isoler. Nous rassemblons ces propriétés dans une signature nommée `vm_relation`. Cette signature est présentée dans Listing 4.7. Elle contient les relations `vm_collaboration` (resp. `vm_concurrence`) qui représentent les couples de MV définies en collaboration (resp. en concurrence). Elle contient également la relation `vm_unidirectional_flow` qui représente les couples de MV entre lesquelles il y a un flux d'information à sens unique. La relation `vm_isolation` représente les MV que le client a définies comme isolées. Nous exprimons toutes ces propriétés de sécurité en fonction de flux d'information entre MV et nous rassemblons tous les flux du système du client dans la relation `vm_flow`.
`vm_sym_collaboration` (resp. `vm_sym_concurrence`) est un ensemble qui contient les relations inverses de l'ensemble `vm_collaboration` (resp. `vm_concurrence`). En effet, les relations de collaboration et concurrence sont symétriques.

Listing 4.7 –

```
one sig vm_relation{
    vm_collaboration: set VM->VM,
    vm_sym_collaboration: set VM->VM,
    vm_concurrence: set VM->VM,
    vm_sym_concurrence: set VM->VM,
    vm_unidirectional_flow: set VM->VM,
    vm_isolation: set VM,
    vm_flow: set VM->VM
    }
```

Nous utilisons dans les modèles Alloy les faits suivants (présentés dans Listing 4.8) pour générer les ensembles symétriques de collaboration et de concurrence.

Listing 4.8 –

```
fact{ all v, k :VM |
  (v->k in vm_relation.vm_collaboration) <=>
  (k->v in vm_relation.vm_sym_collaboration)
  }
fact{ all v, k :VM |
  (v->k in vm_relation.vm_concurrence) <=>
  (k->v in vm_relation.vm_sym_concurrence)
  }
```

Nous utilisons deux ensembles `vm_collaboration` et `vm_sym_collaboration` pour spécifier les relations de collaborations (pour la concurrence également). Le premier ensemble contient ce que le client a spécifié au début et le deuxième contient les relations symétriques des relations spécifiées par le client. L'objectif est d'assurer que la collaboration et la concurrence sont des ensembles de relations symétriques et de réduire le risque d'erreur de spécifications faites par le client.

L'ensemble des flux du système `vm_relation.vm_flow` est constitué à partir des relations de collaboration et des relations de flux unidirectionnels. Si deux MV sont définies en collaboration alors les deux flux qui relient ces deux MV font partie de l'ensemble des flux du système. Le flux unidirectionnel entre deux MV appartient aussi à l'ensemble des flux du système. De plus, cet ensemble ne contient que les flux déduits des relations définies par le client. Il rassemble tous les flux possibles qui relient les MV du client. Cet ensemble simplifie la gestion et la spécification des propriétés de sécurité et contraintes d'interconnexion entre MV. Le fait du Listing 4.9 définit cet ensemble.

Listing 4.9 –

```
fact{ vm_relation.vm_flow =
  vm_relation.vm_collaboration +
  vm_relation.vm_sym_collaboration +
  vm_relation.vm_unidirectional_flow
      }
```

4.4.2.3 Spécification des contraintes de placement et d'interconnexion en utilisant Alloy

Le client spécifie plusieurs contraintes de placement des MV sur les grappes et nœuds physiques des fournisseurs. Il exprime aussi des contraintes d'interconnexion de ses MV. Ces contraintes peuvent être des contraintes de groupement, de séparation ou d'isolation de MV. Les contraintes d'interconnexion concernent l'assurance ou l'interdiction des liens qui permettent d'échanger des informations entre les MV.
La taille du système du client peut être grande, ses exigences et contraintes peuvent être nombreuses et contiennent potentiellement des contradictions. D'ailleurs, c'est pour cela que nous vérifions les spécifications du client avant de procéder au placement des MV dans le nuage informatique. Un modèle Alloy sur-contraint ne nous permet pas d'analyser les besoins du clients. C'est pour cela que nous devons choisir un moyen pour représenter les différents éléments d'un système client et éviter les modèles sur-contraints.

Nous avons deux façons de considérer les besoins du client :

1. Forcer les contraintes dans le système client : nous pouvons traduire toutes les contraintes exigées par le client en des faits (*fact*) dans les modèles Alloy qui décrivent le système du client. Les faits Alloy sont considérés comme toujours vrais dans les modèles Alloy. L'analyseur Alloy les applique pour générer n'importe quelle instance du système et pour vérifier n'importe quelle assertion dans les modèles.

   L'utilisation des faits Alloy pour exprimer toutes les contraintes du client, augmente les probabilités d'avoir un système sur-contraint.

   L'analyseur Alloy ne peut pas retourner une instance d'un système client sur-contraint (contient des contradictions de spécifications). Il ne peut pas vérifier la validité des assertions. En analysant un système Alloy sur-contraint l'analyseur Alloy retourne ces deux résultats :
   - aucune instance trouvée, `show` peut être inconsistante, (« *No instance found. show may be inconsistent* ») quand on cherche à générer une instance du système en exécutant un prédicat `show`,
   - aucun contre-exemple trouvé, `assertion` peut être valide, (« *No counterexample found. assertion may be valid* ») quand on vérifie la validité d'une assertion.

   Nous pouvons utiliser cette méthode, nous avons dans ce cas plusieurs possibilités : soit l'analyseur Alloy réussit à trouver une instance du système, c'est-à-dire que tous les besoins du clients traduits en faits Alloy sont appliqués et donc ses besoins sont assurés. Il est possible que l'analyseur Alloy ne réussisse pas à trouver une instance, car le système est sur-contraint. Dans ce second cas, une solution que nous pouvons utiliser est d'éliminer un fait Alloy, d'analyser le système à nouveau et de voir si l'analyseur trouve une solution. Si aucune solution n'est trouvée, on élimine un autre fait Alloy ou une combinaison de plusieurs

faits à la fois.
Le problème de cette méthode est que nous ne pouvons pas savoir quel est le fait ou la signature qui pose problème. Il est possible d'avoir plusieurs faits contradictoires ou des faits et des signatures contradictoires. Il est aussi possible d'avoir un seul fait qui impose plusieurs contraintes dans le système et que ces contraintes sont contradictoires. La difficulté est alors de trouver la signature, le fait ou les faits qui causent la ou les contradictions. Cela est possible mais compliqué à mettre en œuvre.

2. Ne pas forcer les contraintes dans le système client : nous traduisons toutes les contraintes et les propriétés exigées par le client en assertions dans Alloy. À l'exception des définitions de base telles que : les ressources de CPU, ou le nombre de MV et les relations entre les MV par exemple.
   Une assertion décrit une propriété. Nous utilisons l'analyseur Alloy pour vérifier une assertion : soit elle est satisfaite par toutes les instances possibles du système client, soit il retourne un contre-exemple qui montre que cette assertion est violée par une instance du système et l'assertion n'est pas assurée par le système analysé.
   En utilisant cette méthode, nous laissons plus de liberté à Alloy de générer des instances possibles du système. Nous considérons une instance à la fois et nous vérifions que les propriétés et les contraintes exigées par le client et traduites en assertions Alloy sont valides. Si c'est le cas, donc tous les besoins du clients ne contiennent aucun conflit ou contradiction.
   Si l'analyseur détermine quelques assertions comme non-valides, il retourne des contre-exemples qui peuvent aider à corriger le problème. Par exemple, le client a défini un flux entre deux MV en concurrence. En analysant une assertion qui exprime l'interdiction de flux entre MV en concurrence, l'analyseur retourne un contre-exemple qui met en évidence la violation de cette propriété. Dans ce cas, le problème est plus facile à trouver et à corriger qu'avec la méthode précédente.

4.4.2.4 Exemple des besoins d'un client

Dans cette partie, nous spécifions en Alloy un exemple de système que le client peut demander. Le client décrit le nombre de MV et leurs caractéristiques. Il spécifie les relations entre les MV et les propriétés de sécurité associées.
La spécification Alloy du Listing 4.10 présente un exemple des besoins que peut faire un client de nuage informatique. Cette spécification est traduite automatiquement à partir des besoins que le client a exprimés et des critères qu'il a sélectionnés sur une interface utilisateur (figure 4.2).

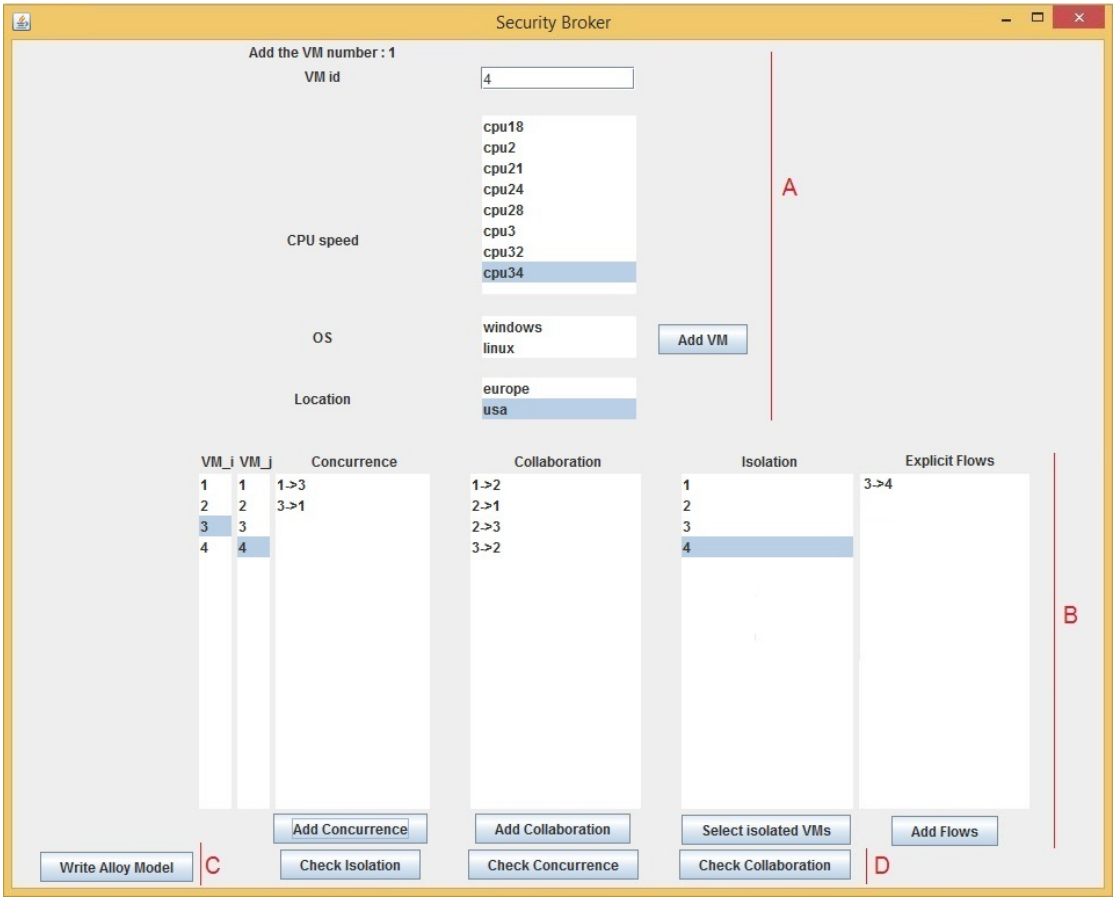

FIGURE 4.2 – *Interface graphique pour la spécification des besoins du client.*

La partie A de cette interface permet de spécifier les caractéristiques de CPU, l'OS et la localisation de chaque MV. La partie B permet de définir les relations de collaboration, de concurrence, d'isolation et les flux unidirectionnels entre les MV. Cette étape nous permet aussi de faire des premières vérifications. Par exemple, le client ne peut pas ajouter une relation de collaboration entre la MV et elle même. La partie C correspond au bouton qui permet de générer les modules Alloy. Ces derniers représentent toutes les spécifications et besoins du client dans le langage Alloy. Les boutons de la partie D permettent de déclencher les vérifications des différentes propriétés de sécurité des modules générés précédemment, en utilisant l'API Alloy. Cette interface correspond à une étape de notre processus et elle peut être améliorée.

Le client `customer1` demande quatre MV et spécifie pour chacune des caractéristiques fonctionnelles. Par exemple la `vm1` a `cpu2` et l'OS `windows` comme caractéristiques et elle doit être allouée en région européenne. De la même façon, on définit celles des autres MV : `vm2` : (CPU24, windows, europe), `vm3` : (CPU2, linux, europe) et `vm4` : (CPU34, linux, usa).

Listing 4.10 –

```
one sig customer1 extends Customer{}
{customer_vms= vm1 + vm2 + vm3 + vm4}

one sig vm1 extends VM{}{
  vm_cpuSpeed = cpu2
  vm_os = windows
  vm_location=europe
  }
```

Le fait du Listing 4.11 définit les propriétés de sécurité que le client exige. Il spécifie les relations de collaboration et de concurrence entre les MV. Il définit une MV comme isolée. Il définit des flux d'information unidirectionnels entre les MV.
L'instance du système client que nous illustrons est représentée par la figure 4.3. `vm1` et `vm2` sont en collaboration et reliées par des flux bidirectionnels, de même pour le couple `vm2` et `vm3`. Les MV `vm1` et `vm3` sont définies en concurrence. La `vm4` est une machine isolée. Le client définit un flux d'information unidirectionnel sortant de la `vm3` et ayant pour destination la `vm4`.

Listing 4.11 –

```
fact fact_vm_relation_instance{
  vm_relation.vm_concurrence = vm1−>vm3
  vm_relation.vm_collaboration = vm1−>vm2 + vm2−>vm3
  vm_relation.vm_isolation = vm4
  vm_relation.vm_unidirectional_flow = vm3−>vm4
  }
```

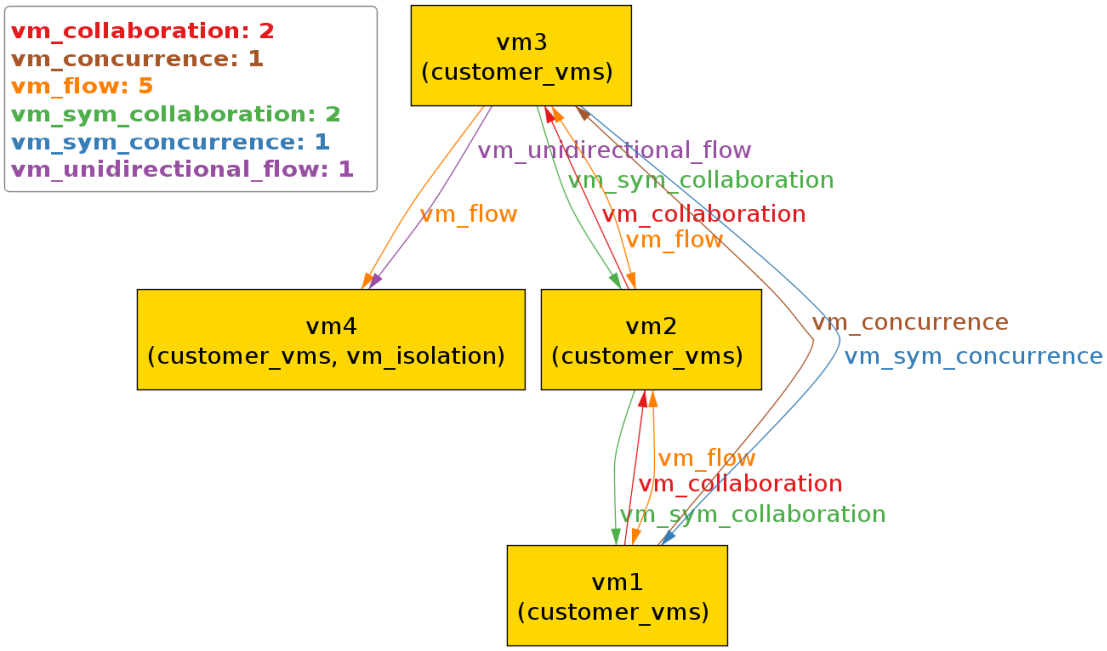

FIGURE 4.3 – *Instance des besoins du client.*

Cette spécification implique quelques propriétés de sécurité que nous avons définies dans la sous-sous-section 4.4.2.2. Le client a défini `vm1` et `vm3` en concurrence, il faut les allouer sur différentes grappes. Il faut aussi que ces deux machines ne soient pas reliées par des flux d'information directs ou indirects. Cela est aussi valable pour toutes les autres propriétés de sécurité. Si deux MV sont définies en collaboration (c'est le cas pour les couples (`vm1`, `vm2`) et (`vm2`, `vm3`)), elle doivent être reliées par des flux bidirectionnels. Les machines isolées ne doivent pas être reliées par des flux à une autre ressource.
Comme le montre la figure 4.3, ces propriétés de sécurité ne sont pas respectées. Dans la suite, nous allons détecter ces conflits que le client a introduits lors de la définition de ses besoins.

4.5 Détection des inconsistances coté client

Nous analysons les spécifications Alloy du client, nous détectons les conflits et les retournons au client avec un contre-exemple qui peut le guider pour corriger ces conflits. Une fois les conflits corrigés, nous passons à l'étape de recherche des offres des fournisseurs qui puissent satisfaire les besoins du client.
Nous utilisons la plupart du temps des assertions pour exprimer les propriétés à vérifier. Une assertion Alloy est une spécification qui peut être analysée. Alloy peut vérifier la satisfiabilité de la spécification dans le système ou trouver un contre-exemple qui montre la non-satisfiabilité de cette spécification dans le système.
Alloy nous donne la possibilité de retourner toutes les instances possibles d'un système, ainsi que tous les contre-exemples d'une assertion non-satisfiable, étant donné que Alloy analyse un système de taille finie.

Pourquoi analyser le système du client avant le placement ? Nous considérons les MV en collaboration comme des machines reliées de façon constante par des flux pour qu'elles puissent échanger des informations couramment. Cela est contradictoire avec le principe de concurrence, qui empêche les MV de collaborer et d'échanger des informations. Le principe de flux d'information unidirectionnel concerne les échanges d'information entre deux MV à sens unique seulement. Le concept de la MV isolée permet de la séparer complètement des autres, c'est-à-dire elle n'est pas reliée avec l'extérieur par des flux d'information.
Détecter les conflits de description des besoins du client nous permet de définir de façon correcte le réseau d'interconnexion et d'échange d'information entre les ressources du système client. Cela facilite au courtier la tâche de placement des ressources chez les fournisseurs. En effet, quand le courtier reçoit une description d'un système client qui ne contient pas de conflit, il peut les placer et les interconnecter de façon plus fiable. Cela réduit aussi les conflits qui peuvent exister après le placement et l'interconnexion chez les fournisseurs. Par exemple, le courtier peut définir les règles d'un pare-feu pour protéger les ressources du client de façon sûre si les spécifications du client ne contiennent pas de conflit. Ainsi, on réduit le risque d'autoriser les flux d'information là où il ne faut pas ou de bloquer des flux d'information là où il faut les autoriser.

4.5.1 Conflits des spécifications fonctionnelles

Il se peut que le client se trompe en définissant ses besoins, car le système qu'il définit est de très grande taille par exemple. Le client peut par exemple attribuer deux vitesses de CPU différentes à la même MV. Nous traitons ce genre de conflits à l'étape de spécification des besoins. Nous autorisons au client de définir les caractéristiques en limitant le nombre de valeurs de chaque attribut. Par exemple, on définit une MV de la manière suivante:

Listing 4.12 –

```
abstract sig VM {
  vm_cpuSpeed: one CPU,
  vm_os: one OS,
  vm_location: lone Location
  }
```

On utilise le terme `one` pour préciser le nombre d'occurrences de vitesse de CPU et de l'OS. Cela exige une seule vitesse de CPU et un seul OS pour chaque MV. Ainsi, le client ne peut définir qu'une seule valeur d'attribut pour chacune des deux caractéristiques. Pour l'attribut `vm_location`, le client peut spécifier au plus une seule valeur. En plus, nous faisons de telles restrictions au niveau de la saisie dans l'interface graphique.

4.5.2 Conflits des spécifications non-fonctionnelles

Dans cette sous-section nous traduisons les besoins non-fonctionnels du client en spécifications Alloy afin de vérifier leur satisfiabilité par le système du client. Nous analysons toutes les propriétés et contraintes exprimées par le client par rapport à son système. Nous mettons en évidence les propriétés violées et les conflits de spécifications. Le client prend en compte les retours afin de corriger les conflits s'il y en a. Nous commençons d'abord par analyser les relations que le client a définies entre les MV. Après, nous analysons l'ensemble des flux d'information du système client pour détecter tous les flux contradictoires.

4.5.2.1 Conflit des relations

Une MV ne peut être en collaboration ou en concurrence avec elle même. Il ne doit pas y avoir un flux d'information unidirectionnel qui a comme source et comme destination la même MV. Cette propriété est exprimée en Alloy dans l'assertion du Listing 4.13. Il n'existe aucune MV v, telle que la relation v→v soit une relation de collaboration ou de concurrence ou une relation qui représente un flux unidirectionnel entre la MV v et elle même.

Listing 4.13 –

```
assert assert_relation_auto{
  no v: VM |
  v->v in vm_relation.vm_collaboration ||
  v->v in vm_relation.vm_sym_collaboration ||
  v->v in vm_relation.vm_concurrence ||
  v->v in vm_relation.vm_sym_concurrence ||
  v->v in vm_relation.vm_unidirectional_flow
  }
```

Si deux MV disjointes sont définies en collaboration, alors elles ne peuvent pas être définies en concurrence en même temps et il ne peut y avoir un flux unidirectionnel entre les deux. De plus, ces deux MV ne peuvent être des machines isolées.
En Alloy, nous spécifions cette propriété en utilisant une assertion comme présenté dans Listing 4.14. Si vi→vj est une relation de collaboration alors vi→vj ne peut être une relation de concurrence ni un flux unidirectionnel. Les deux MV vi et vj ne peuvent pas être des machines isolées.

Listing 4.14 –

```
assert assert_relation_collaboration{
  all disj vi, vj: VM |
  ( vi->vj in vm_relation.vm_collaboration ||
    vi->vj in vm_relation.vm_sym_collaboration
  ) =>
  ( vi->vj !in vm_relation.vm_concurrence &&
    vi->vj !in vm_relation.vm_sym_concurrence &&
    vi->vj !in vm_relation.vm_unidirectional_flow &&
    vi !in vm_relation.vm_isolation &&
    vj !in vm_relation.vm_isolation
  )
}
```

Nous exprimons les mêmes propriétés pour détecter les conflits des descriptions du client. De la même façon que l'assertion précédente, dans Listing 4.15, nous spécifions que si deux MV sont en concurrence alors elles ne peuvent pas être en collaboration et il ne peut pas y avoir un flux unidirectionnel entre les deux. De même, les deux MV ne peuvent pas être isolées.

Listing 4.15 –

```
assert assert_relation_concurrence{
  all disj vi, vj: VM |
  ( vi->vj in vm_relation.vm_concurrence ||
    vi->vj in vm_relation.vm_sym_concurrence
  ) =>
  ( vi->vj !in vm_relation.vm_collaboration &&
    vi->vj !in vm_relation.vm_sym_collaboration &&
    vi->vj !in vm_relation.vm_unidirectional_flow &&
    vi !in vm_relation.vm_isolation &&
    vj !in vm_relation.vm_isolation
  )
}
```

L'assertion du Listing 4.16 suivante spécifie que si une relation entre deux MV est définie comme un flux unidirectionnel, alors elle ne peut pas représenter une relation de collaboration ou de concurrence et les deux MV de cette relation ne peuvent pas être isolées.

Listing 4.16 –

```
assert assert_relation_unidirect_flow{
  all disj vi, vj: VM |
  vi->vj in vm_relation.vm_unidirectional_flow =>
  ( vi->vj !in vm_relation.vm_collaboration &&
    vi->vj !in vm_relation.vm_sym_collaboration &&
    vi->vj !in vm_relation.vm_concurrence &&
    vi->vj !in vm_relation.vm_sym_concurrence &&
    vi !in vm_relation.vm_isolation &&
    vj !in vm_relation.vm_isolation
  )}
```

En analysant les assertions `assert_relation_auto`(4.13), `assert_relation_collaboration`(4.14) et `assert_relation_concurrence`(4.15), l'analyseur Alloy ne retourne pas de contre exemple. Cela veut dire que les spécifications exprimées par ces trois assertions sont satisfiables pour le système défini par le client dans l'exemple 4.4.2.4.

Cependant, un contre-exemple est retourné en analysant l'assertion 4.16 `assert_relation_unidirect_flow`.

La figure 4.4 illustre ce contre-exemple. Il y a un flux unidirectionnel provenant de `vm3` (`$assert_relation_unidirect_flow_vi`) et allant vers `vm4` (`$assert_relation_unidirect_flow_vj`). La `vm4` qui correspond à `vj` dans l'assertion est une MV isolée. Cela est un contre-exemple qui montre la non-validité de la propriété « on n'autorise pas d'avoir un flux entre deux MV et que l'une des deux soit isolée ».

Comme présenté dans la figure 4.4, le contre-exemple retourné par l'analyseur Alloy met en évidence les entités (les MV ici) qui participent dans la violation de la spécification exprimée par l'assertion. Connaissant l'objectif de cette spécification, nous pouvons déduire les autres éléments qui ont causé le conflit comme le flux unidirectionnel et le fait que `vm4` est isolée.

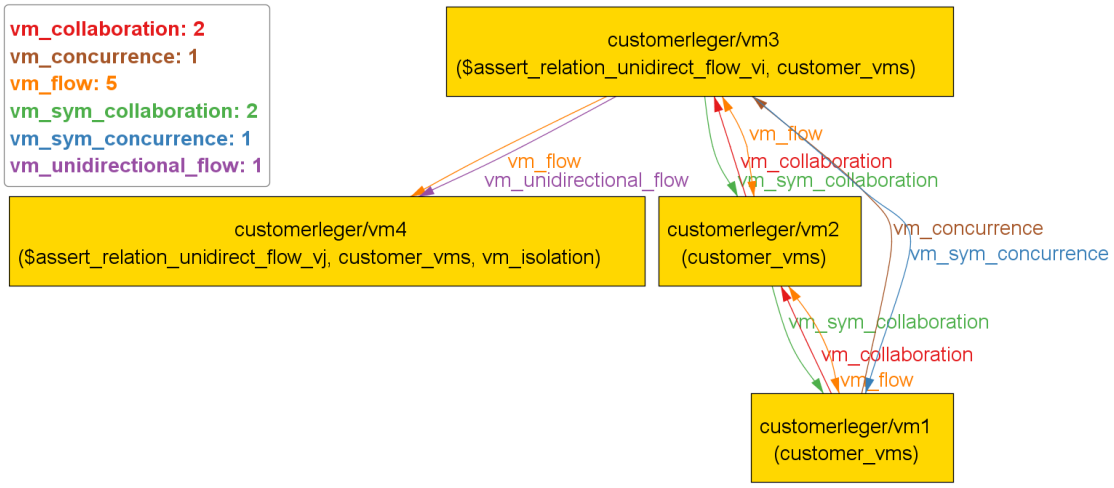

FIGURE 4.4 – *Contre-exemple : relation de flux non satisfiable.*

4.5.2.2 Conflit des flux d'interconnexion

Nous analysons ici l'ensemble des flux d'information du système client. Nous vérifions en utilisant des assertions et prédicats Alloy que toutes les propriétés et contraintes d'interconnexion exigées par le client sont respectées par rapport au système défini. Autrement dit, nous vérifions qu'il existe bien un lien reliant des MV qui doivent communiquer et qu'il n'existe pas de flux direct ou indirect entre celles qui ne doivent pas communiquer.

Isolation : Il n'existe aucun flux entrant vers ou sortant d'une MV isolée. L'assertion 4.17 décrit cette propriété. Pour toute MV vi, il n'existe aucune MV vj telles que, vi est une MV isolée et il existe un flux entre vi et vj dans l'un des (ou les deux) sens.

Listing 4.17 –

```
assert assert_vm_flow_isol{
  all vi:VM | no vj: VM |
  vi in vm_relation.vm_isolation &&
  ( vi->vj in vm_relation.vm_flow ||
    vj->vi in vm_relation.vm_flow
  ) }
```

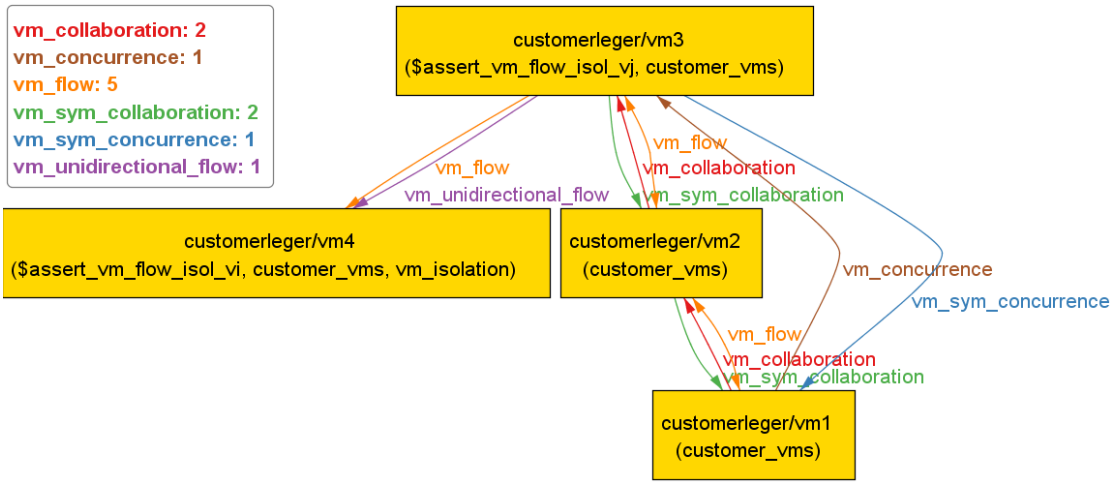

FIGURE 4.5 – *Contre-exemple : propriété d'isolation non satisfiable.*

La figure 4.5 décrit un contre exemple qui prouve que la propriété d'isolation n'est pas satisfiable dans le système client 4.4.2.4. La machine `vm4` est définie comme isolée. Elle correspond à la `vi` de l'assertion `assert_vm_flow_isol`. Cette machine est reliée par un flux unidirectionnel venant de la `vm3`. Cette dernière correspond à `vj` de la même assertion.

Collaboration : Nous illustrons la propriété de collaboration en utilisant les deux méthodes que nous avons expliquées précédemment dans la sous-sous-section 4.4.2.3. Nous distinguons la méthode dans laquelle on force la propriété de collaboration dans le système et la méthode dans laquelle on ne la force pas mais on vérifie qu'elle est satisfiable par les instances du système client.

- **Traduction de la propriété en fait Alloy :** la propriété suivante exige que toutes les MV en collaboration doivent être reliées par des flux d'information. Cette propriété peut être exprimée dans le fait `fact_vm_flow_collab` 4.18. Pour tout couple de MV disjointes, si ces deux MV sont en collaboration, alors il faut que les flux d'information vi→vj et vj→vi existent et appartiennent à l'ensemble `vm_relation.vm_flow` des flux du système du client. Cela assure l'échange d'information dans les deux sens entre les MV qui sont en collaboration.

Listing 4.18 –

```
fact fact_vm_flow_collab{
  all disj vi, vj:VM |
  ( vi->vj in vm_relation.vm_collaboration ||
    vi->vj in vm_relation.vm_sym_collaboration
  ) =>
  ( vi->vj in vm_relation.vm_flow &&
    vj->vi in vm_relation.vm_flow
  ) }
```

Comme nous l'avons expliqué dans la sous-sous-section 4.4.2.3, si on procède en utilisant cette méthode et que le client définit des faits contradictoires (une MV isolée et en collaboration avec une autre MV par exemple), ce sera difficile de les détecter.

Un autre exemple : si on force en utilisant un fait, une propriété de flux d'information unidirectionnel entre `vm3` et `vm4`. On force aussi un fait pour la propriété d'isolation de la `vm4` qui ne doit être reliée par aucun flux d'information. Dans ce cas, le système est sur-contraint, en l'analysant, on n'aura aucune instance, aucun contre-exemple et aucune indication sur les conflits.

– **Traduction de la propriété en assertion Alloy :** Pour vérifier la validité de cette propriété, il faut que tout couple de MV en collaboration soit relié par des liens dans les deux sens. Pour cela, il suffit de trouver un contre-exemple qui prouve que la propriété est violée. Autrement dit, si on arrive à trouver dans une instance du système client, un couple de MV entre lesquelles il n'y a pas de lien dans un sens ou dans les deux sens, alors la propriété n'est pas satisfiable. Si on ne trouve aucun contre-exemple dans le cadre du système défini par le client (c'est-à-dire pour toutes les instances possibles du système), alors cette propriété est satisfiable.
L'assertion du Listing 4.19 retourne un contre-exemple s'il existe deux MV qui sont en collaboration et qu'il n'y a pas de flux d'information entre les deux dans un ou les deux sens.

Listing 4.19 –

```
assert assert_vm_flow_collab{
  no disj vi, vj:VM |
  ( vi->vj in vm_relation.vm_collaboration ||
    vi->vj in vm_relation.vm_sym_collaboration
  ) &&
  ( vi->vj ! in vm_relation.vm_flow ||
    vj->vi ! in vm_relation.vm_flow
  ) }
```

Dans l'exemple du système client 4.4.2.4, toutes les MV en collaboration sont interconnectées par des flux. Dans ce cas, l'analyseur Alloy ne parvient pas à trouver un contre-exemple. Cela veut dire que dans le cadre du système défini par le client et en se limitant seulement aux éléments définis, la propriété de collaboration spécifiée par cette assertion semble valide. Cette propriété peut ne pas être valide si le système analysé change ou sa taille augmente (par exemple s'il y aura plus de MV).

Nous avons choisi de ne pas forcer (en utilisant des faits) toutes les contraintes dans les modèles Alloy à l'exception des contraintes techniques basiques comme le nombre

de MV, les ressources comme le CPU ou la localisation ainsi que les relations de collaboration/concurrence/flux unidirectionnel/isolation entre les MV. En effet, les relations entre les MV et les propriétés de flux d'informations deviennent rapidement complexes.
En se basant sur les spécifications de base du client, l'analyseur Alloy génère une solution possible et vérifie qu'elle satisfait toutes les assertions qui représentent les contraintes du client. Dans la suite, nous allons présenter seulement les assertions Alloy qui décrivent les différentes contraintes et propriétés exprimées par le client.

Concurrence : Il n'existe aucun flux direct reliant deux MV en concurrence. L'assertion du Listing 4.20 retourne un contre exemple, si en analysant le système défini par le client, il se trouve que deux MV sont en concurrence et sont reliées par un flux direct.

Listing 4.20 –

```
assert assert_vm_flow_concur{
  no disj vi, vj:VM |
  ( vi->vj in vm_relation.vm_concurrence ||
    vi->vj in vm_relation.vm_sym_concurrence
  ) &&
  ( vi->vj in vm_relation.vm_flow ||
    vj->vi in vm_relation.vm_flow
  ) }
```

L'analyseur Alloy ne retourne pas de contre-exemple. En effet, dans le système du client 4.4.2.4, il n'y pas de flux direct entre les deux MV en concurrence `vm1` et `vm3`. C'est le seul couple de MV définies en concurrence par le client.

Nous pouvons vérifier la propriété de concurrence de façon à interdire les liens indirects qui relient les MV en concurrence. Car les informations peuvent aussi transiter indirectement entre les MV en concurrence.
Dans ce cas, le client exige qu'il ne doit exister aucun flux direct ou transitif reliant deux MV en concurrence. L'assertion `assert_vm_flow_concur_trans` (Listing 4.21) sert à vérifier la satisfiabilité de cette propriété dans le système client. Un contre-exemple est retourné s'il existe deux MV définies en concurrence et reliées par un flux d'information transitif.

Listing 4.21 –

```
assert assert_vm_flow_concur_trans{
  no disj vi, vj:VM |
  ( vi->vj in vm_relation.vm_concurrence ||
    vi->vj in vm_relation.vm_sym_concurrence
  ) &&
  ( (vj in vi.^(vm_relation.vm_flow)) ||
    (vi in vj.^(vm_relation.vm_flow))
  ) }
```

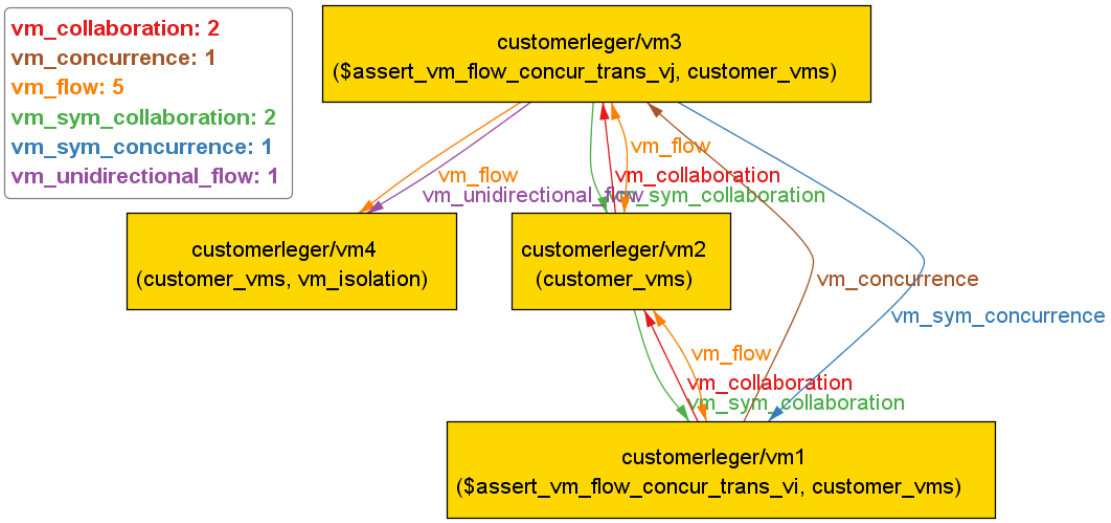

FIGURE 4.6 – *Contre-exemple : propriété de concurrence non satisfiable.*

La figure 4.6 montre le contre-exemple qui prouve la non-satisfiabilité de la propriété de concurrence exprimée par l'assertion `assert_vm_flow_concur_trans`. En effet, les deux MV `vm1` et `vm3` sont en concurrence et sont reliées par des flux indirects. Il y au moins un flux transitif, par exemple, un flux entre `vm1` et `vm3` qui passe via `vm2`.

Cette assertion `assert_vm_flow_concur_trans`(4.21) inclut l'assertion précédente `assert_vm_flow_concur`(4.20) qui détecte seulement les flux directs entre les MV en concurrence. Cependant, nous préférons séparer les deux assertions pour la raison suivante. Si l'assertion `assert_vm_flow_concur` retourne un contre-exemple, on retrouve facilement ce flux qui relie deux MV en concurrence et introduit un conflit. Le client peut donc le corriger facilement. Par contre, si l'assertion `assert_vm_flow_concur_trans` retourne un contre-exemple, il faut fournir un effort pour trouver le flux indirect qui relie les deux MV en concurrence. Le client doit aussi fournir un effort pour résoudre le conflit, surtout s'il s'agit d'un long flux transitif.

Alloy ne retourne pas explicitement ce chemin transitif comme déjà constaté dans la figure 4.6. L'analyseur Alloy met des étiquettes sur les signatures concernées par le contre-exemple (par exemple `$assert_vm_flow_concur_trans_vi` sur `vm3`). Cependant, il ne met pas en évidence les chemins de flux indirects que se soit en exploitant le contre-exemple visuellement ou textuellement. La détection de ce flux transitif peut être faite de façon automatique. La résolution du conflit peut aussi être faite automatiquement. Cela ne fait pas partie des travaux de cette thèse.

Confidentialité : Le prédicat `confidentiality` du Listing 4.22 vérifie qu'une MV `vi` est confidentielle par rapport à une autre `vj`. Ceci revient à vérifier l'absence de flux sortant de `vi` et ciblant `vj`.

Listing 4.22 –

```
pred confidentiality [vi, vj : VM]{
  (vj !in vi.^(vm_relation.vm_flow))
  }
```

Intégrité : Le prédicat `integrity` du Listing 4.23 vérifie l'intégrité d'une MV par rapport à une autre. Une MV `vi` est intègre par rapport à une autre `vj` quand il n'existe aucun flux d'écriture sortant de la deuxième `vj` MV et ciblant la première `vi`.

Listing 4.23 –

```
pred integrity [vi, vj : VM]{
  (vi !in vj.^(vm_relation.vm_flow))
  }
```

`vm4` est définie comme isolée dans le système du client 4.4.2.4. On veut vérifier la confidentialité et l'intégrité de `vm4` par rapport à `vm3`. L'analyseur Alloy ne retourne pas de contre-exemple pour la confidentialité. Cela revient à dire que `vm4` est confidentielle par rapport à `vm3`. Par contre, `vm4` n'est pas intègre par rapport à `vm3` parce qu'il existe un flux d'information provenant de `vm3` et ciblant `vm4`.

Comment lever les conflits ? Le client du nuage informatique a quelques pistes pour corriger les contradictions de ses définitions de système. En exploitant le (ou les) contre-exemple retourné par l'analyseur Alloy, le client a les possibilités suivantes :

Il redéfinit son système, c'est-à-dire les besoins fonctionnels et non-fonctionnels, par exemple allouer deux MV au lieu de la `vm1`. La première MV contiendrait les données avec lesquelles elle collabore avec la `vm2`. La deuxième MV contiendrait l'autre partie

des données sensibles à ne pas divulguer à la MV concurrente vm3. Ainsi, on évitera le risque de flux d'information transitif entre les deux MV en concurrence vm1 et vm3.

Il redéfinit les relations entre MV, par exemple en supprimant la relation de lien unidirectionnel entre vm3 et vm4 la propriété d'isolation devient satisfiable. En supprimant la relation de concurrence entre vm1 et vm3, il n'y aura plus de conflit de flux reliant des machines concurrentes.

Il modifie ou réduit ses exigences de propriétés de sécurité et contraintes d'interconnexion. Il peut par exemple interdire les flux directs entre MV en concurrence mais pas forcement les flux indirects.

Il peut garder les mêmes besoins qu'il a exigés mais demande au courtier de lui proposer un placement de gardiens de façon à assurer toutes les propriétés de sécurité. Par exemple, le courtier place un gardien virtuel entre vm2 et vm3. Ainsi, la propriété de concurrence entre vm1 et vm3 reste assurée car le gardien placé va contrôler tous les flux circulant et bloquer les flux indirects indésirables.
Nous allons présenter plus de détails dans le chapitre 5.

4.6 Conclusion

Dans ce chapitre, nous avons présenté les offres que les fournisseurs de services de nuage informatique peuvent offrir aux clients. Nous avons présenté les besoins qu'un client peut demander quand il souhaite adopter le nuage informatique.
Nous avons par la suite présenté le modèle formel Alloy que nous utilisons pour décrire les offres des fournisseurs et les besoins du client.
Les besoins du client sont divisés en deux parties : les besoins fonctionnels et non-fonctionnels. Les besoins fonctionnels consistent en tout ce qui est ressources et leurs caractéristiques techniques. Il s'agit des nœuds de calcul (MV) et les performances associées comme CPU, RAM et OS. Les besoins non-fonctionnels représentent les propriétés de sécurité et les contraintes d'interconnexion et de placement des ressources chez les fournisseurs.
Les offres et les besoins sont spécifiés en utilisant le modèle formel Alloy et sont analysés en utilisant l'analyseur Alloy. Nous vérifions que les spécifications du client ne contiennent pas de contradictions et conflits. L'analyseur Alloy retourne des contre-exemples dans le cas de présence de conflit pour permettre au client de les corriger.
Quand le courtier reçoit des spécifications des besoins du client et qu'il ne détecte pas de conflit, il procède à la mise en correspondance des besoins aux offres, et par la suite au placement. Cela fait l'objet du chapitre 5.

Placement 5

Sommaire

5.1 Introduction 75
5.2 Mise en correspondance des besoins et des offres 76
5.3 Mise en correspondance des besoins fonctionnels 78
5.3.1 Recherche des correspondances 79
5.3.2 Affectation des MV aux nœuds physiques 80
5.4 Mise en correspondance des besoins non-fonctionnels 86
5.4.1 Vérification de la validité des contraintes de placement du client 89
5.4.2 Vérification de la validité des contraintes d'interconnexion du client . . 92
5.4.3 Exemple - Le courtier vérifie la satisfiabilité des besoins non-fonctionnels 94
5.5 Conclusion 99

5.1 Introduction

Dans le chapitre précédent, nous avons présenté la première étape du mécanisme de courtage durant laquelle le courtier reçoit les offres des fournisseurs de nuage informatique et les besoins du client. Il analyse les besoins du client et lui signale les conflits de ses spécifications. Le courtier passe à l'étape suivante après qu'il ait reçu un système client ne contenant pas de conflit.
Dans ce chapitre, nous présentons la deuxième étape de notre mécanisme de courtage qui est le placement des ressources du système client chez les fournisseurs. L'objectif de ce chapitre est de montrer la possibilité d'intégrer les exigences des clients en termes de sécurité dans un processus de courtage des services de nuage informatique. Le placement des ressources du client chez les fournisseurs est effectué en considérant à la fois les critères fonctionnels et non-fonctionnels de son système, y compris des propriétés de sécurité et des contraintes d'interconnexion. Tout cela devra faire partie des contrats SLA établis entre le client, le courtier et les fournisseurs des services, ce qui donne plus d'assurance et garantie au client.

Le placement est basé sur un algorithme de mise en correspondance des besoins du client avec les offres des fournisseurs. Cet algorithme est présenté dans la section 5.2. Nous détaillons dans la section 5.3 la démarche de mise en correspondance des critères

fonctionnels et dans la section 5.4, celles des critères non-fonctionnels. Nous illustrons les étapes de l'algorithme de placement tout au long de ce chapitre avec un exemple détaillé. Nous décrivons toutes les contraintes exprimées par le client, les offres exprimées par les fournisseurs ainsi que les procédures de mise en correspondance en utilisant le langage Alloy.

5.2 Mise en correspondance des besoins et des offres

Dans le chapitre précédent, le courtier reçoit les offres des fournisseurs et les offres du client de nuage informatique. Il analyse les besoins du client. S'il trouve des conflits entre les spécifications des besoins du client, il lui signale ces conflits en lui fournissant des contre-exemples qui l'aident à corriger le problème.
Le courtier passe à l'étape suivante, après avoir reçu des spécifications consistantes des besoins (c'est-à-dire des spécifications qui ne contiennent pas de conflit). Durant cette étape le courtier essaye de trouver des correspondances entre les besoins du client et les offres des fournisseurs. Il essaye de placer les MV dans des nœuds physiques et de satisfaire tous les besoins fonctionnels et non-fonctionnels du client. La figure 5.1 résume l'algorithme de sélection des nœuds physiques, d'affectation des MV aux nœuds physiques et de la génération des configurations de placement et d'interconnexion des MV chez les fournisseurs de services nuagiques.

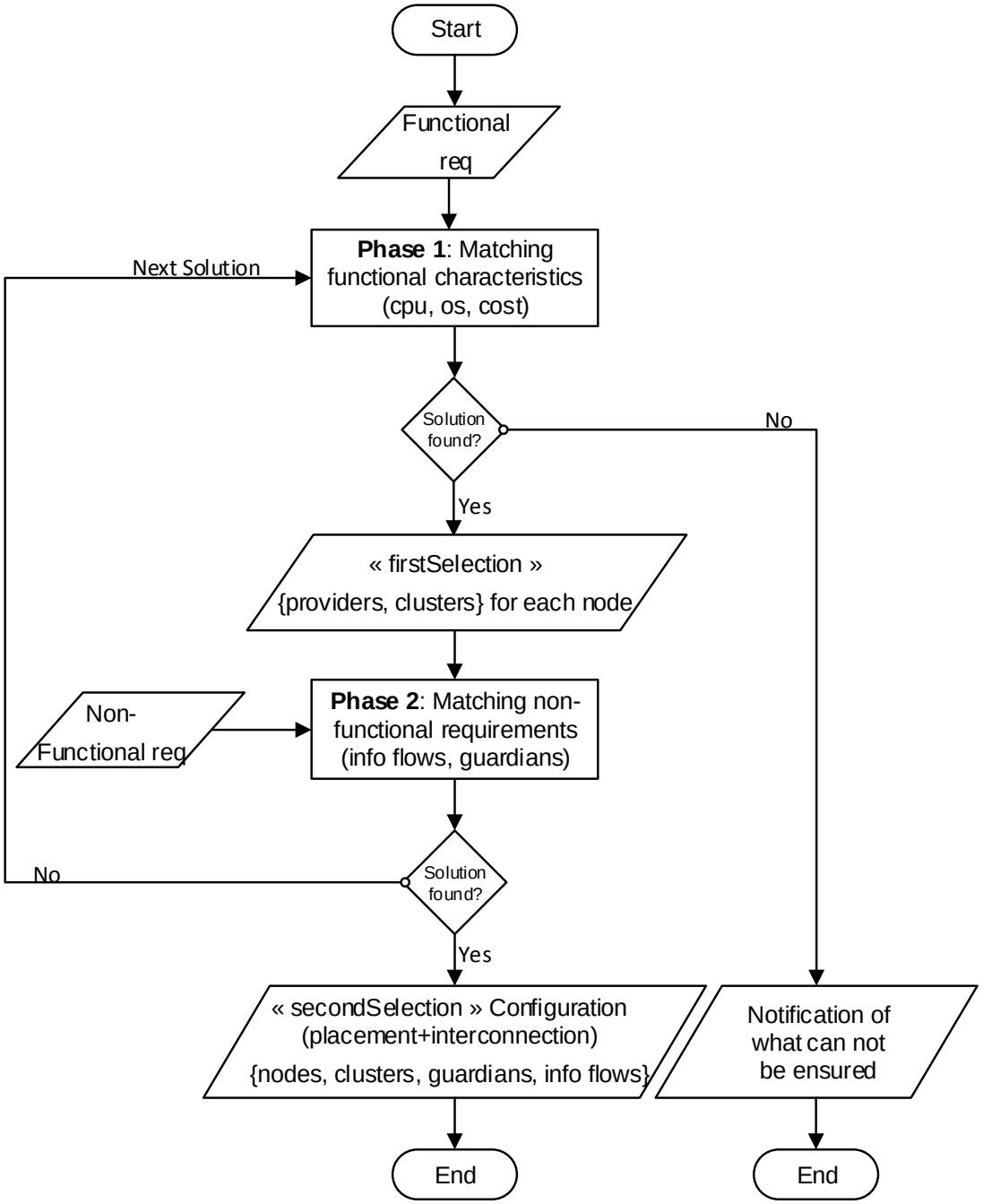

FIGURE 5.1 – *L'algorithme de placement.*

L'algorithme consiste en deux phases principales. **La phase 1** est l'étape où le courtier reçoit la description des besoins fonctionnels du système du client. Le courtier a au préalable connaissance des offres des fournisseurs. Il essaye donc de trouver pour chaque MV un nœud physique qui peut assurer toutes ses caractéristiques fonctionnelles.

Cette phase retourne une solution et la renvoie en entrée de la phase 2. Cette solution consiste en un ensemble d'affectations MV-nœud physique nommé `firstSelection`. Cependant, il se peut que le courtier ne trouve pas de correspondances. En effet, les offres des fournisseurs que le courtier gère peuvent ne pas assurer certains critères

exigés par le client. Cela peut arriver soit pour des problèmes d'indisponibilité de ressources soit parce que les quantités de ressources sont insuffisantes pour répondre aux besoins de ce client.
Dans ce cas, le courtier analyse de façon rigoureuse les caractéristiques fonctionnelles une à une, afin de retourner au client des moyens qui peuvent le guider à comprendre ce qui ne peut pas être assuré. Le client a deux choix, soit il modifie les critères de son système et relance la phase 1 à nouveau, soit il arrête la procédure auprès de ce courtier.
La phase 2 utilise le résultat `firstSelection` de la phase 1 et reçoit en entrée les offres et les besoins non-fonctionnels. Dans cette étape, le courtier vérifie que les placements de MV et les interconnexions entre les grappes ne violent pas les propriétés de sécurité exigées par le client. Dans le cas échéant, le courtier retourne la solution au client, cette solution est nommée `secondSelection`. Elle répond à toutes les contraintes fonctionnelles et non-fonctionnelles posées par le client.
Dans le cas où quelques propriétés ne sont pas satisfaites par le placement proposé de la phase 1, le courtier annule cette solution et demande une nouvelle solution de la phase 1 afin de la traiter dans la phase 2. La phase 2 est donc exécutée à nouveau avec une nouvelle solution. Ce processus est répété jusqu'à ce que le courtier trouve un placement qui satisfait les contraintes non-fonctionnelles ou que le courtier épuise les solutions de la phase 1.

Nous avons séparé cet algorithme en deux phases dans le but que la phase 1 soit traitée en utilisant un outil autre que Alloy. Alloy n'est pas adapté aux calculs numériques, c'est pour cela que la phase 1 devrait être traitée par un solveur de programmation par contraintes par exemple. Cet outil peut manipuler des calculs numériques afin de trouver des correspondances aux besoins fonctionnels du client. La solution retournée par cet outil doit être traduite en spécifications Alloy. Comme décrit dans ce chapitre, la phase 2 sera exécutée en utilisant Alloy car il est adapté aux vérifications logiques des propriétés d'un modèle. Dans ce manuscrit, la phase 1 est aussi réalisée en utilisant Alloy.

5.3 Mise en correspondance des besoins fonctionnels

Cette étape consiste à mettre en correspondance les besoins fonctionnels du client avec les offres des fournisseurs et de sélectionner les offres compatibles. Il s'agit de la phase 1 sur la figure 5.1.

5.3.1 Recherche des correspondances

Dans cette étape, seulement les caractéristiques fonctionnelles des besoins et des offres sont prises en compte pour la recherche des correspondances.
Une MV est allouée sur un nœud physique d'une grappe seulement si les caractéristiques de cette MV (c'est-à-dire CPU, RAM, taille de disque, bande passante) sont fournies par ce nœud physique. D'autres critères comme le prix, la localisation, les mécanismes de chiffrement ou d'authentification peuvent aussi être mis en correspondance dans cette étape. En effet, il s'agit seulement de faire correspondre les valeurs de localisation par exemple entre ce que le client demande et ce que le nœud physique peut assurer.

La mise en correspondance se fait en utilisant Alloy. Le fait `vm_to_cluster` du Listing 5.1 effectue la mise en correspondance des caractéristiques fonctionnelles. Ce fait assure qu'un nœud physique ne peut héberger une MV que s'il propose toutes les caractéristiques fonctionnelles exigées pour cette MV. En effet, pour toutes les MV du système client et tous les nœuds physiques des fournisseurs, si l'une des caractéristiques telles que la vitesse de CPU, l'OS de la MV ou sa localisation n'est pas fournie par un nœud physique donné, alors cette MV ne peut pas être allouée sur ce nœud physique.

Listing 5.1 –

```
fact vm_to_cluster{
  all v: VM | all p: PhysicalNode |
  ( (v.vm_cpuSpeed !in p.cpuSpeed)||
    (v.vm_os !in p.os) ||
    (v.vm_location!=p.location)
  )=>
  (v !in p.vms)
}

fact vm_to_cluster_one_pnode{
  all v:VM | one p:PhysicalNode|
  v in p.vms
}
fact vm_to_cluster_only_one_pnode{
  all v:VM | all disj pi, pj:PhysicalNode|
  v in pi.vms =>
  v !in pj.vms
}
```

Le fait `vm_to_cluster_pnode` assure que chaque MV est allouée sur un nœud

physique. Le fait `vm_to_cluster_only_one_pnode` assure que si une MV est allouée sur un nœud physique donné, alors elle ne peut pas être allouée sur un autre nœud physique. Ceci est utile pour s'assurer que le courtier affecte chaque MV à un seul nœud physique uniquement.

5.3.2 Affectation des MV aux nœuds physiques

Le courtier forme un premier ensemble qui contient les affectations de chaque MV à un nœud physique sélectionné. Cet ensemble est nommé `firstSelection` (Figure 5.1, page 77). Ces nœuds physiques satisfont les besoins fonctionnels des MV comme spécifié par le client. Pour chaque MV, un nœud physique qui satisfait toutes les caractéristiques fonctionnelles de cette MV est sélectionné comme hôte possible. Ces nœuds physiques sont des hôtes potentiels des ressources du client.

La phase 1 peut avoir plusieurs solutions possibles. Il peut y avoir plusieurs solutions possibles, c'est-à-dire pour une MV donnée, plusieurs nœuds physiques sont capables de l'héberger car ils satisfont ses caractéristiques fonctionnelles. Ainsi, la phase 1 peut avoir plusieurs solutions possibles. Nous traitons une seule solution à la fois. Si cette solution satisfait les besoins non-fonctionnels (phase 2) alors la solution est adoptée et retournée au client. Dans le cas où la solution que nous considérons ne satisfait pas les besoins non-fonctionnels (phase 2), nous traitons d'autres solutions de la phase 1.

On ne trouve aucune correspondance. Dans la phase 1, si les caractéristiques fonctionnelles des MV ne sont pas satisfaites, c'est-à-dire, si aucune correspondance n'est trouvée, le courtier notifie au client les critères qui posent problème. Les offres de fournisseurs que le courtier gère peuvent ne pas satisfaire les critères d'une ou plusieurs MV. Cela peut arriver parce que les offres que le courtier gère ne proposent pas un critère donné, par exemple : aucun nœud physique n'est localisé dans la région de Canada ou aucun nœud physique ne peut assurer un CPU de vitesse 3.4GHZ. Il se peut aussi que le critère peut être assuré mais la quantité disponible est insuffisante pour répondre aux besoins du client, par exemple : il n'y a pas assez d'espace de stockage dans la région du Danemark pour un client qui souhaite réserver des moyens de stockage dans cette région.
Le client a deux choix, soit il annule la procédure auprès de ce courtier soit il continue. Dans ce dernier cas, le client prend en compte les retours du courtier et modifie ses besoins fonctionnels. Cela peut aussi impacter les besoins non-fonctionnels. Autrement dit, le client décrit de nouveaux besoins pour son système qu'il souhaite déployer dans le nuage informatique et les envoie à nouveau au courtier.
À ce moment là, le courtier cherche les correspondances des besoins fonctionnels à nouveau, c'est-à-dire que la phase 1 est effectuée de nouveau.

Exemple : Nous décrivons dans la suite un cas d'usage simple des besoins d'un client et des offres des fournisseurs. La figure 5.2 représente une description des besoins fonctionnels d'un système client. Il s'agit d'un nombre de MV avec des caractéristiques fonctionnelles pour chacune. La figure 5.3 représente la vue que le courtier a des offres fonctionnelles des fournisseurs. Chaque fournisseur propose un nombre de nœuds physiques groupés dans des grappes ainsi que les caractéristiques fonctionnelles proposées.

Taille des systèmes à analyser : Dans ce manuscrit, nous illustrons notre démarche avec des exemples d'offres des fournisseurs et des besoins de clients de petites tailles. Cependant, notre courtier est capable de traiter des systèmes de plus grandes tailles. Les limites qu'on a sont le nombre de variables que peut générer Alloy (voir sous-section 2.2.3) ou la quantité de mémoire de la machine. Pour **décrire des systèmes de grandes tailles** nous avons deux possibilités :

- l'assistant de l'interface graphique qui permet aux utilisateurs de décrire leurs systèmes de façon aisée. Un utilisateur peut saisir dans l'interface qu'il a besoin de 100 MV par exemple. Ensuite, il peut étiqueter des sous-ensembles de MV avec des niveaux de sécurité (voir sous-sous-section 4.3.2.1). L'utilisateur peut donc définir des propriétés de sécurité et des relations entre les MV en fonction des niveaux de sécurité. La traduction en spécifications Alloy se fait automatiquement.
- Alloy offre un moyen pour générer un nombre d'instances des éléments d'un modèle comme l'utilisateur le souhaite. Dans les trois spécifications Alloy suivantes, la première permet de définir 4 relations d'isolation. Le symbole # représente la cardinalité dans Alloy. Avec cette spécification, Alloy génère des instances du modèle qui ont 4 relations d'isolation (c'est-à-dire 4 MV isolées).

```
1) #vm_relation.isolation=4
2) check {assert_vm_flow_concur} for 10 but exactly 100 VM
3) run show for 3 but exactly 100 VM
```

La deuxième spécification permet de vérifier l'assertion `assert_vm_flow_concur` en fixant le nombre de MV à générer à 100 (exactly 100 VM). Le nombre des instances des autres éléments du modèle est soit défini explicitement dans le modèle (one sig cpu3, cpu34 extends CPU), soit il est plafonné à 10 instances (for 10 but ...).

La troisième spécification exécute un prédicat `show` dans Alloy. Il permet de générer une instance du modèle Alloy pour exactement 100 MV (exactly 100 VM) et au maximum 3 instances pour les autres éléments du modèle (for 3 but ...) sauf si les instances ont été définies explicitement dans le modèle. Alloy génère par défaut 3 instances pour chaque élément du modèle si on n'a rien spécifié.

La vérification des assertions des modèles Alloy (propriétés de sécurité et tous les besoins du client) se fait sans problème. En effet, nous avons écrit les assertions, les faits et les fonctions de notre courtier de façon abstraite, c'est-à-dire, la vérification d'une assertion ne dépend ni des instances ni de la taille du système à analyser.

Détails des besoins du client : Les besoins du client consistent en quatre MV : `vm1`, `vm2`, `vm3` et `vm4` avec des caractéristiques fonctionnelles (figure 5.2). Ces dernières consistent en un triplet constitué de la vitesse de CPU, le système d'exploitation de la MV ainsi que sa localisation souhaitée par le client. Le client décrit les quatre MV comme suit : `vm1` : (`cpu2`, `window`, `europe`), `vm2` : (`cpu24`, `window`, `europe`), `vm3` : (`cpu2`, `linux`, `europe`) et `vm4` : (`cpu34`, `linux`, `usa`).

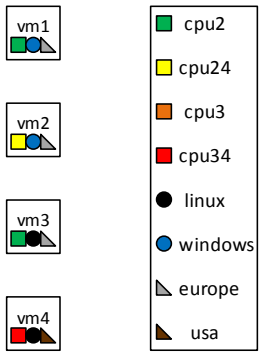

FIGURE 5.2 – *Exemple : les besoins fonctionnels du client.*

Détails des offres des fournisseurs : La figure 5.3 représente la vue que le courtier a des offres des fournisseurs. Dans cet exemple, le courtier gère des offres de trois fournisseurs (amazon, orange et sfr). amazon propose trois grappes (amazonZone1, amazonZone2 et amazonZone3), les deux premières grappes regroupent chacune deux nœuds physiques (amazonZone1node1 et amazonZone1node2 pour la grappe amazonZone1). La grappe amazonZone3 présente un seul nœud physique amazonZone3node1. orange propose une grappe (orangeZone1) qui regroupe deux nœuds physiques. Également, sfr propose une grappe regroupant deux nœuds physiques.
Tous les nœuds du fournisseur amazon peuvent héberger des MV qui ont des caractéristiques fonctionnelles parmi la liste suivante : une vitesse de CPU de `cpu2`, `cpu24`, `cpu3` ou `cpu34`, un système d'exploitation `linux` ou `windows` et une localisation en `europe`. Les offres fonctionnelles des autres nœuds sont présentées dans la figure 5.3.

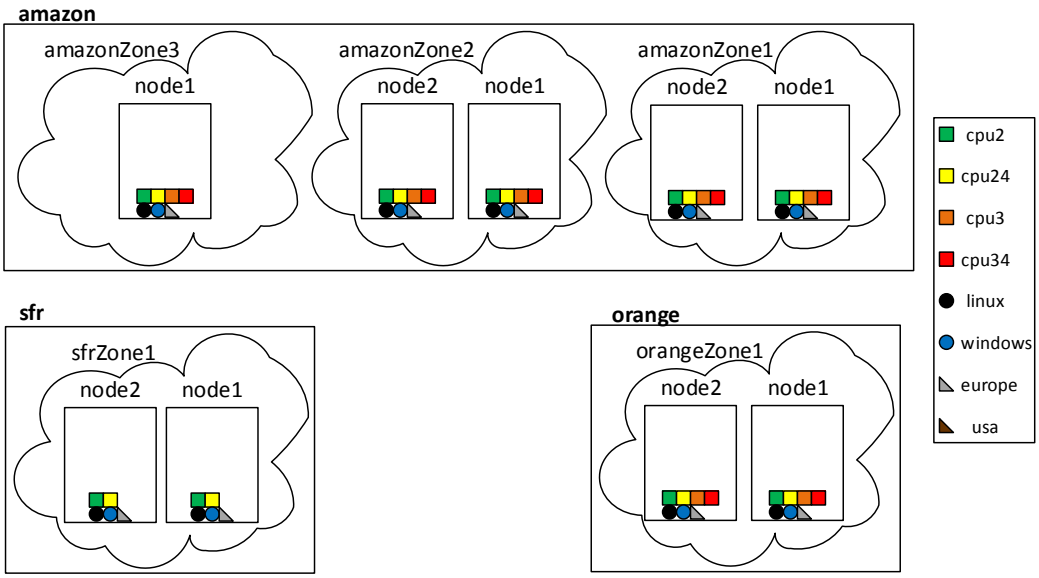

FIGURE 5.3 – *Exemple : les offres fonctionnelles des fournisseurs.*

Résultat de la recherche des correspondances : Dans cet exemple, Alloy ne trouve pas de correspondance. Les offres que le courtier gère ne peuvent pas répondre à l'ensemble des besoins de ce client.

Nous analysons le système du client par rapport aux offres proposées en utilisant toujours Alloy. Nous constatons que la `vm4` ne peut être allouée sur aucun des nœuds physiques, comme présenté par la figure 5.4. Le contre-exemple présenté dans cette figure est obtenu en vérifions une assertion `assert_functional` qui consiste en : pour chaque MV, il existe au moins un nœud physique qui peut assurer toutes ses caractéristiques fonctionnelles.

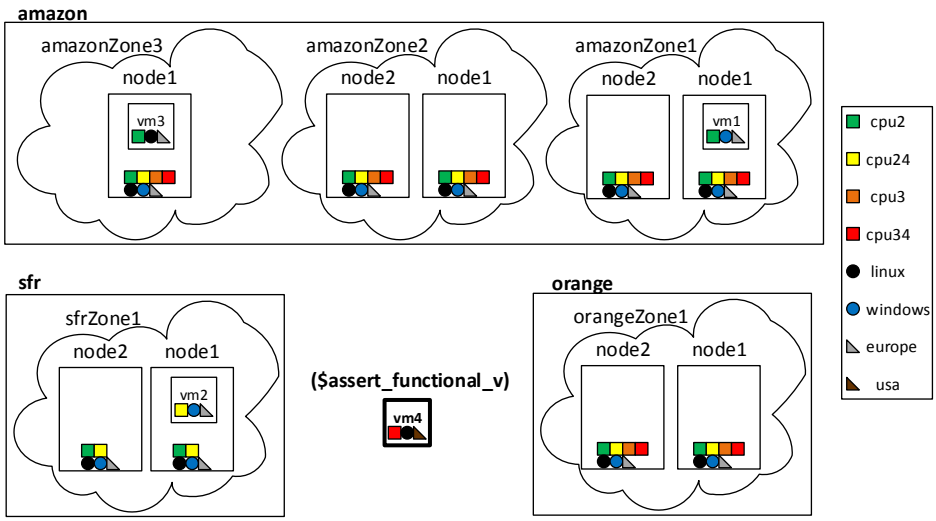

FIGURE 5.4 – *Exemple : une des* MV *ne peut être allouée sur aucun nœud physique.*

Alloy peut retourner un contre-exemple justifiant les caractéristiques qui ne peuvent pas être assurées. Cela est effectué en analysant plus en détails la disponibilité des caractéristiques dans les offres proposées. Pour cela, nous vérifions des assertions spécifiques à chaque caractéristique. Par exemple l'assertion `assert_functional_location` 5.2 vérifie que pour chaque MV, il existe au moins un nœud physique qui peut assurer sa localisation. Des assertions similaires peuvent être vérifiées afin de prouver la possibilité de satisfaire les caractéristiques fonctionnelles indépendamment.

Listing 5.2 –

```
assert assert_functional_location{
  all v: VM | some p: PhysicalNode |
  v.vm_location=p.location
  }
```

En analysant cette assertion, Alloy retourne un contre-exemple. Ce contre-exemple concerne la caractéristique localisation de la `vm4` qui est égale à `usa` et qui ne peut être assurée par aucun des nœuds physiques proposés. En effet, tous les nœuds physiques dans cet exemple sont localisés dans la région `europe`.

Retour au client : Alloy retourne au client les contre-exemples qui justifient la non-satisfiabilité des besoins exigés. Dans l'exemple que nous avons illustré, la localisation de la `vm4` ne peut pas être assurée. Si le client souhaite se procurer des ressources chez ce courtier, il doit modifier ses besoins fonctionnels (et non-fonctionnels si nécessaire) afin que le courtier puisse répondre à ses besoins en utilisant les offres des fournisseurs qu'il gère.

Nouvelles spécifications des besoins du client : Le client modifie les besoins que le courtier n'a pas pu satisfaire. Le nouveau système du client contient cinq MV. Il garde `vm1` : (`cpu2`, `window`, `europe`), `vm2` : (`cpu24`, `window`, `europe`) et `vm3` : (`cpu2`, `linux`, `europe`). Le client remplace la `vm4` par deux MV. En effet, au lieu d'allouer une MV dans la région `usa`, il partage les données censées être traitées par `vm4` en deux. Une MV contient des données sensibles et doit être isolée même si elle est allouée dans une région autre que `usa`. L'autre MV préserve les mêmes interconnexions avec les autres MV comme `vm4` dans le système précédent et contient des données moins sensibles. Les caractéristiques des deux nouvelles MV sont comme suit : `vm4` : (`cpu34`, `linux`, `europe`) et `vm5` : (`cpu34`, `linux`, `europe`). Les nouveaux besoins fonctionnels du client sont présentés dans la figure 5.5.

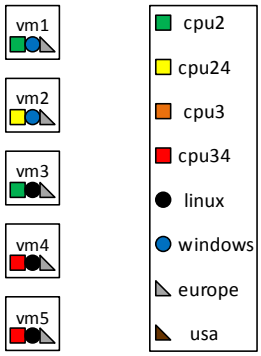

FIGURE 5.5 – *Exemple : les nouveaux besoins fonctionnels du client.*

Phase 1 - Recherche des correspondances : Le courtier reçoit la nouvelle description du système du client. Il essaye à nouveau de mettre en correspondance les besoins fonctionnels du client avec les offres des fournisseurs. Dans ce cas, le courtier réussit à trouver une solution. Il affecte chacune des MV à un nœud physique et propose le placement présenté dans la figure 5.6.

La figure 5.6 présente les affectations MV-nœud physique possibles répondant aux besoins du client décrits dans la figure 5.5. La figure 5.6 présente l'ensemble firstSelecion obtenu après la mise en correspondance des besoins fonctionnels et les offres des fournisseurs. Pour chaque MV, un nœud physique qui satisfait toutes les caractéristiques fonctionnelles de la MV est sélectionné comme hôte possible. Ces affectations sont obtenues en appliquant les faits 5.1 décrits en page 79.

Par exemple : le nœud physique `sfrZone1node1` fournit `cpu24`, `windows` et est localisé en `europe`, donc la MV `vm2` peut être allouée sur ce nœud.

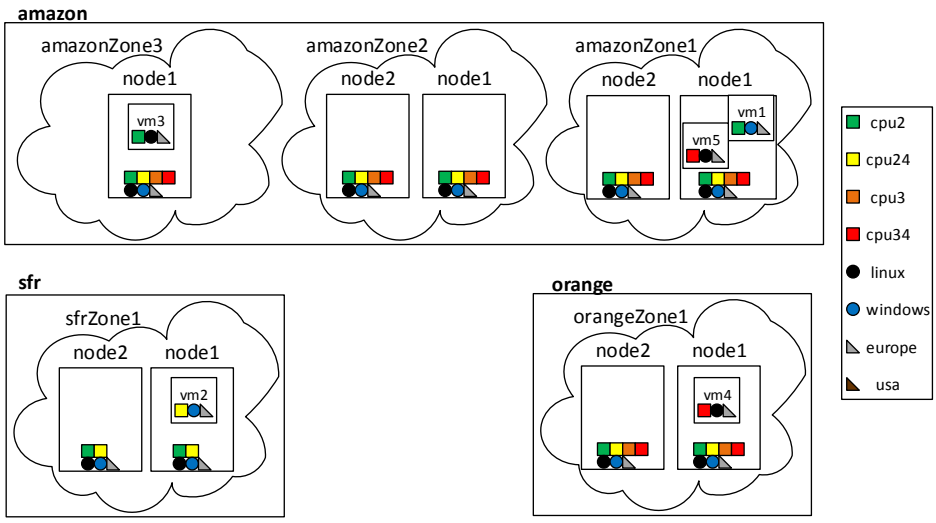

FIGURE 5.6 – *Exemple : l'ensemble* ***firstSelection*** *proposé par le courtier.*

5.4 Mise en correspondance des besoins non-fonctionnels

Dans l'étape précédente, le courtier a sélectionné un ensemble d'offres des fournisseurs qui répondent aux caractéristiques fonctionnelles des MV spécifiées par le client.
Dans cette étape, le courtier analyse le placement généré dans l'ensemble `fisrtSelection` par rapport aux offres non-fonctionnelles des fournisseurs. L'objectif de cette étape est de vérifier que les offres non-fonctionnelles assurent toutes les contraintes non-fonctionnelles du système client et ne violent aucune propriété exigée.
Dans cette seconde étape ; phase 2 sur la figure 5.1 (page 77), le courtier place les MV sur les nœuds physiques comme spécifié dans `firstSelection`. Il considère les offres non-fonctionnelles des fournisseurs qui comprennent les interconnexions entre les grappes, les placements des gardiens et les conflits entre grappes ou fournisseurs. Ensuite, il vérifie que cette configuration des interconnexions, gardiens et conflits assure le bon fonctionnement du système client tout en satisfaisant ses besoins non-fonctionnels.
Le courtier vérifie que les MV qui devront échanger des informations puissent communiquer virtuellement à travers des réseaux physiques qui relient les grappes qui les hébergent. Il vérifie aussi la possibilité que les flux indésirables seront bloqués au niveau des gardiens.

Si la configuration générée par le courtier assure la satisfiabilité des besoins fonctionnels et non-fonctionnels du client, le courtier construit un ensemble `secondSelection`. Cet ensemble représente la configuration du système client qu'on va déployer dans le nuage informatique. L'ensemble représente les affectations des MV sur les nœuds physiques, les interconnexions ainsi que les emplacements des gardiens entre les grappes. Le courtier notifie le client et lui retourne la configuration générée. Si le client est satisfait de la solution, il confirme son accord pour cette configuration. Un contrat entre le client, le courtier et les fournisseurs de service sélectionnés est établi.
Si le courtier échoue à générer une configuration de placement qui satisfait les contraintes du client, il génère un autre ensemble `firstSelection`. On l'appelle solution suivante qui doit être différente des solutions `firstSelection` déjà traitées. Si un tel ensemble existe, le courtier réessaye de mettre en correspondance les besoins non-fonctionnels. Si aucune solution n'est trouvée, le courtier retourne des contre-exemples concernant les propriétés non assurées. Par exemple : le nombre de grappes sélectionnées dans l'ensemble `firstSelection` n'est pas suffisant pour héberger les MV séparément ; le client exige que les MV en concurrence ne doivent pas communiquer mais les seules grappes qui peuvent héberger ces MV sont déjà reliées par des liens d'information non-contrôlés.

Offres non-fonctionnelles : Le courtier reçoit des descriptions des offres non-fonctionnelles des fournisseurs de nuage informatique. En reliant les différentes descriptions, le courtier conçoit une vue globale de ces offres qui peuvent représenter :

- Des liens qui relient des grappes leur permettant d'échanger les informations sans aucun contrôle. Ces liens représentent des réseaux physiques installés entre les grappes physiques.
- Des liens contrôlés par des gardiens. Ce sont des liens représentants des réseaux physiques qui relient les grappes. Cependant des gardiens sont placés sur ces réseaux afin de contrôler les flux circulant sur ces liens.
- Des relations de conflit entre des grappes. Ces relations permettent de limiter les collaborations et le partage entre deux grappes. Par exemple, le courtier ne peut pas placer des MV en collaboration sur des grappes en conflit.

Exemple : On reprend l'exemple précédent pour illustrer cette phase de mise en correspondance des besoins non-fonctionnels.
Dans la phase 2, le courtier considère à la fois les besoins non-fonctionnels du client ainsi que les offres non-fonctionnelles des fournisseurs. Les offres sont représentées par des liens d'interconnexion entre les différentes grappes des fournisseurs, ainsi qu'un ensemble de gardiens placés entre les grappes afin de contrôler les flux d'information. La figure 5.7 représente la vue globale que le courtier a, en considérant toutes les offres reçues des fournisseurs. Cette vue représente à la fois les caractéristiques fonctionnelles et non-fonctionnelles des nœuds physiques, des grappes et des interconnexions.
Dans cet exemple, le courtier a connaissance des faits suivants :

- Les grappes `amazonZone1` et `amazonZone2` sont reliées par un lien direct (qui représente un réseau physique).
- Les grappes `amazonZone2` et `amazonZone3` sont reliées par un lien direct.
- Les grappes `amazonZone1` et `orangeZone1` sont reliées par un lien contrôlé par un gardien.
- Les grappes `amazonZone2` et `sfrZone1` sont reliées par un lien contrôlé par un gardien.
- Les grappes `orangeZone1` et `sfrZone1` sont en conflit.

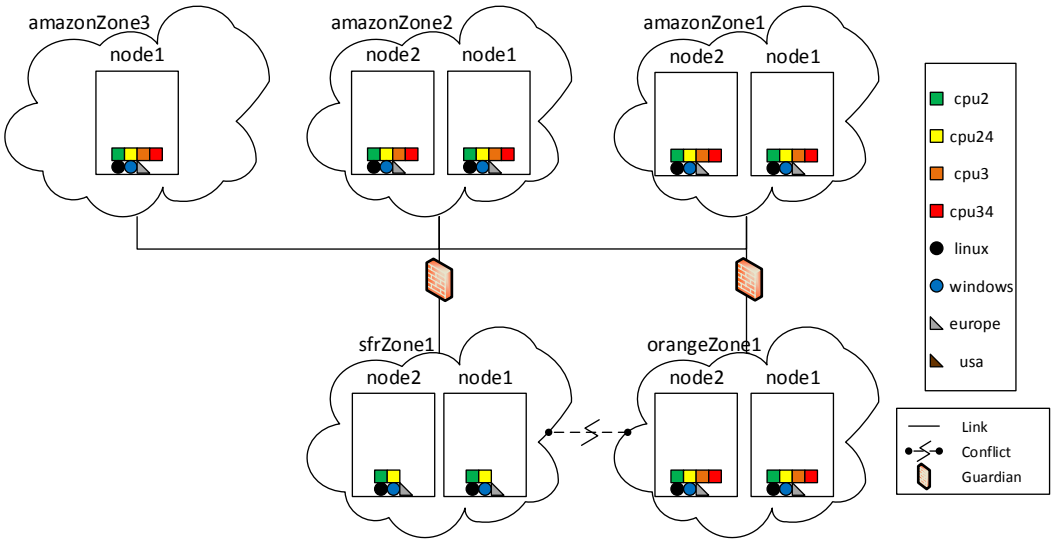

FIGURE 5.7 – *Exemple : les offres fonctionnelles et non-fonctionnelles des fournisseurs.*

Le client définit les besoins non-fonctionnels de son système. C'est un ensemble de propriétés de sécurité et de contraintes de placement et d'interconnexion des MV qui constituent son système. Les types des besoins non-fonctionnels que le client peut exprimer sont présentés précédemment dans la section 4.3.2. Les besoins non-fonctionnels du client de notre exemple sont présentés dans la figure 5.8 et sont constitués de :

- **Propriétés de sécurité :** Le client définit des propriétés de sécurité pour ses MV. Ce sont des relations et des contraintes appliquées sur l'ensemble de ses MV qui auront un impact sur le placement.
  - Collaboration : `vm1` et `vm2` sont définies en collaboration. `vm2` et `vm3` sont aussi définies en collaboration.
  - Concurrence : `vm1` et `vm3` sont définies en concurrence.
  - Flux d'information unidirectionnel : un flux d'information est défini provenant de `vm3` et à destination de `vm5`.
  - Isolation : `vm4` est définie comme une machine isolée.
- **Contraintes de placement et d'interconnexion :** Le client spécifie un ensemble de contraintes de placement de ses MV chez les fournisseurs. Il spécifie aussi les liens d'interconnexion entre MV qui doivent ou ne doivent pas être autorisés.
  - Les MV qui sont en collaboration doivent être allouées soit sur le même nœud physique, soit sur la même grappe ou sur différentes grappes. Dans le cas d'allocation sur différentes grappes, ces dernières doivent être reliées par des liens autorisant les flux d'information. Dans cet exemple, le client n'exige pas de placer les MV en collaboration chez le même fournisseur.
  - Les MV qui sont en concurrence doivent être allouées sur des grappes différentes. Ces dernières ne doivent pas être reliées par des liens directs ou

transitifs. Cependant, elles peuvent être reliées par des liens ou des chemins de liens contrôlés par des gardiens.

– Une MV isolée doit être allouée seule sur un nœud physique.

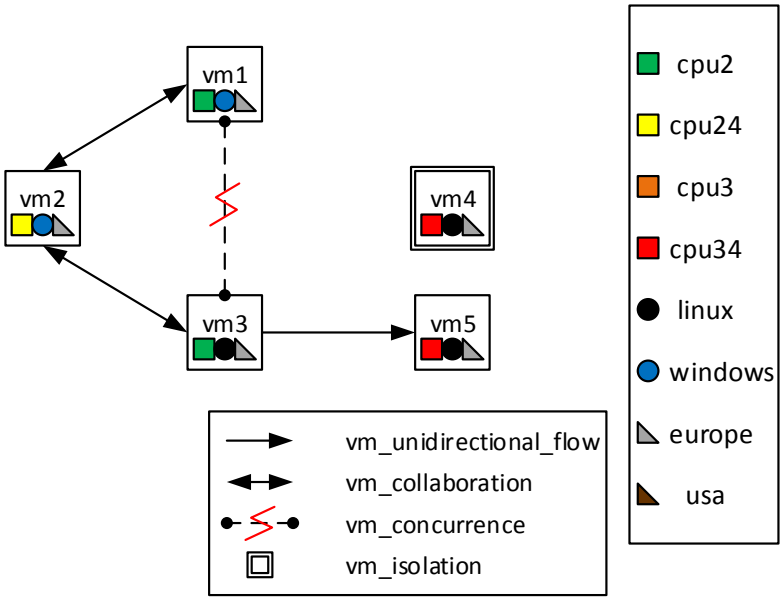

FIGURE 5.8 – *Exemple : les besoins non-fonctionnels.*

Le courtier considère le placement généré dans la phase 1 (l'ensemble `firstSelection` dans la figure 5.6, page 85), et vérifie si ce placement satisfait tous les besoins non-fonctionnels exprimés par le client. Pour cela, le courtier vérifie la validité d'un ensemble d'assertions. Chaque assertion vérifie que le placement proposé satisfait une propriété de sécurité, une contrainte de placement ou la cohérence des liens d'interconnexion. Cela fait l'objet de la sous-section suivante.

Dans la sous-section suivante, nous décrivons les assertions Alloy utilisées par le courtier afin de vérifier que le placement généré assure la satisfiabilité des propriétés de sécurité et des contraintes de placement et d'interconnexion spécifiées par le client.

5.4.1 Vérification de la validité des contraintes de placement du client

Le courtier vérifie d'abord que les contraintes de placement exigées par le client sont respectées par la configuration générée. Ces contraintes peuvent représenter des propriétés de séparations ou d'isolation de MV qui permettent de réduire le risque de fuite d'information. Elle peuvent aussi représenter des propriétés de co-résidence de MV sur le même nœud ou la même grappe physique afin d'optimiser le débit ou permettre le partage des ressources physiques ou virtuelles entre MV.

Grappes en conflit : Le courtier vérifie que les MV d'un client ne sont pas placées sur des grappes en conflit. L'assertion du Listing 5.3 vérifie qu'il n'existe pas deux MV du même client qui sont placées sur deux grappes en conflit.

Listing 5.3 –

```
assert assert_cluster_conflict {
  all ci, cj:Cluster| all c:Customer| no disj vi, vj:VM|
  (vi in c.customer_vms) &&
  (vj in c.customer_vms) &&
  (cj->ci in cluster_relation . conflict ) &&
  (vi in ci.host.vms) &&
  (vj in cj.host.vms)
  }
```

Nous pouvons spécifier différentes variantes de cette propriété. Par exemple, une propriété peut exprimer que les MV d'un même client ne peuvent pas être hébergées par deux fournisseurs différents.

Groupement des MV en collaboration : Les MV en collaboration doivent être hébergées par le même fournisseur de service. L'assertion du Listing 5.4 permet de vérifier qu'il n'existe pas deux MV qui sont en collaboration et sont placées chez des fournisseurs différents.

Listing 5.4 –

```
assert assert_host_collab{
  all p:Provider| no disj vi, vj: VM |
  ( (vj in vi.(vm_relation.vm_collaboration)) ||
    (vj in vi.(vm_relation.vm_sym_collaboration))
  ) &&
  (vi in p.clusters.host.vms ) &&
  (vj !in p.clusters.host.vms)
  }
```

De la même façon, le courtier peut vérifier que les MV en collaboration sont placées sur la même grappe, ou sur le même nœud physique selon l'exigence du client.

Séparation : Les MV en concurrence doivent être placées sur des grappes différentes. Ainsi on réduit tout risque de fuite d'information due au partage de ressources matérielles au sein de la même grappe.
L'assertion du Listing 5.5 permet de vérifier qu'il n'existe pas deux MV qui sont en concurrence et sont placées sur la même grappe.

Listing 5.5 –

```
assert assert_host_concur{
  all c:Cluster| no disj vi, vj: VM |
  ( (vj in vi.(vm_relation.vm_concurrence))||
    (vj in vi.(vm_relation.vm_sym_concurrence))
  ) &&
  (vi in c.host.vms ) &&
  ( vj in c.host.vms)
  }
```

Le courtier vérifie une assertion du même genre afin de s'assurer que deux MV reliées par un flux d'information à sens unique, doivent être placées sur deux grappes différentes, comme spécifié dans Listing 5.6.

Listing 5.6 –

```
assert assert_host_unidirectional_flow{
  all c:Cluster| no disj vi, vj: VM |
  (vj in vi.(vm_relation.vm_unidirectional_flow)) &&
  (vi in c.host.vms ) &&
  ( vj in c.host.vms)
  }
```

Isolation : Une MV isolée doit être allouée seule sur un nœud physique. Cela permet de réduire le risque de fuite d'information entre des MV qui sont allouées sur le même nœud physique et qui partagent des ressources matérielles (mémoire vive ou cache par exemple).
L'assertion du Listing 5.7 permet de vérifier qu'il n'existe pas deux MV qui sont placées sur le même nœud physique et que l'une des deux est définie comme isolée.

Listing 5.7 –

```
assert assert_host_isol{
  all p:PhysicalNode | no disj vi, vj: VM |
  (vi in vm_relation.vm_isolation) &&
  (vi in p.vms) &&
  (vj in p.vms)
  }
```

5.4.2 Vérification de la validité des contraintes d'interconnexion du client

En plus de la vérification des contraintes de placement, le courtier vérifie que les interconnexions entre grappes respectent aussi les propriétés de sécurité exprimées par le client. Le client est capable d'exiger la présence d'un gardien entre toutes les grappes qui hébergent ses MV. Il peut aussi exiger de ne pas autoriser des liens indirects entre ses MV qui passent à travers d'autres grappes n'hébergeant pas une de ses MV. Parce que cela peut altérer l'intégrité ou la confidentialité des flux d'information échangés entre ses MV. Nous illustrons quelques propriétés importantes que le client peut exiger.

Flux entre deux grappes : Nous utilisons le prédicat `exists_link_guard` 5.8 qui vérifie si une grappe ni peut communiquer avec une autre grappe nj, soit à travers des liens de flux soit à travers des liens de flux contrôlés par des gardiens. Nous réutilisons ce prédicat dans d'autres propriétés de ce chapitre.
L'expression (nj in ni.^(cluster_relation.link)) vérifie s'il existe un lien (ou chemin de liens) de flux normaux direct ou transitif entre les deux grappes.
L'expression (nj in ni.^(*(cluster_relation.link).^(cluster_relation.guardian).*(cluster_relation.link)) vérifie s'il existe un chemin de flux contrôlé entre les deux grappes.
Ce dernier est un chemin de flux qui contient au moins un gardien d'où l'utilisation du symbole ^. En effet cela exige qu'il y est au moins un gardien dans le chemin qui relie la grappe ni à nj. Le gardien est précédé et suivi par des liens de flux normaux avec le symbole * pour dire qu'il peut y avoir zéro ou plusieurs liens de flux dans le chemin. L'expression représente tous les types de chemins de flux contrôlés qu'on peut avoir. Le chemin peut commencer et finir par un flux normal ou contrôlé. Au milieu de chemin on peut aussi avoir toutes les combinaisons de chemins de flux normaux et contrôlés possibles.

Listing 5.8 –

```
pred exists_link_guard[ni: Cluster, nj: Cluster]{
  (nj in ni.^(cluster_relation.link)) ||
  (nj in ni.^(*(cluster_relation.link).^(cluster_relation.guardian).*(
    cluster_relation.link))
  )}
```

Flux entre les MV en collaboration : Si deux MV sont en collaboration alors elles doivent être placées soit sur la même grappe, soit sur deux grappes différentes qui peuvent communiquer. Dans ce dernier cas, les deux grappes doivent être reliées par des liens directs ou transitifs qui autorisent la circulation des flux d'information. Elles peuvent donc être reliées par des liens directs ou par des chemins de liens de flux normaux ou contrôlés par des gardiens. Cela fait l'objet de la spécification 5.9.

Listing 5.9 –

```
assert assert_host_flow_collab{
  all disj vi, vj: VM | all ni, nj: Cluster |
  (
  ( (vj in vi.(vm_relation.vm_collaboration)) ||
    (vj in vi.(vm_relation.vm_sym_collaboration))
  ) &&
  (vi in ni.host.vms) &&
  (vj in nj.host.vms)
  ) =>
  ( (ni=nj) ||
    ( exists_link_guard[ni,nj] &&
      exists_link_guard[nj,ni]
    )
  ) }
```

Flux entre les MV en concurrence : Si deux MV sont en concurrence alors les grappes qui les hébergent ne doivent pas être reliées par des liens directs ou transitifs, ou ces grappes peuvent être reliées seulement par des liens (ou chemins de liens) qui contiennent au moins un gardien.
Les gardiens servent à contrôler les flux d'information et bloquer les flux indésirables, tels que les flux entre deux MV en concurrence.
L'assertion du Listing 5.10 vérifie que pour tout couple de MV en concurrence qui sont placées sur deux grappes, il n'existe aucun chemin de liens de flux normaux reliant les deux grappes. Cela sous-entend que l'existence de chemins de flux contrôlés par des gardiens entre les deux grappes est autorisée.

Listing 5.10 –

```
assert assert_host_flow_gard_concur{
  all disj vi, vj: VM | all ni, nj: Cluster |
  (
  ( (vj in vi.(vm_relation.vm_concurrence)) ||
    (vj in vi.(vm_relation.vm_sym_concurrence))
  ) &&
  (vi in ni.host.vms) && (vj in nj.host.vms)
  ) =>
  ( (nj !in ni.^( cluster_relation . link )) &&
    (ni !in nj.^( cluster_relation . link ))
  ) }
```

Flux entre les MV reliées par un flux d'information à sens unique : Ici on considère le cas où le client définit un flux d'information à sens unique entre deux MV et que ces MV sont placées sur deux grappes différentes.
Dans le sens du flux autorisé, il doit y avoir un chemin de liens normaux et/ou contrôlés entre les deux grappes.
Dans le sens contraire, il faut qu'il n'existe pas de chemin de liens normaux entre les deux grappes. Cela sous-entend que l'existence d'un chemin contrôlé est autorisée.
Si on considère que les liens qui relient les grappes sont bidirectionnels, alors les grappes qui hébergent ces MV doivent être reliées par un chemin de liens qui passe obligatoirement par un gardien. Ce chemin de liens contrôlé permet d'autoriser les flux d'information autorisés par le client et de bloquer les flux du sens contraire qui sont interdits par le client.

Listing 5.11 –

```
assert assert_host_flow_gard_unidirectional_flow{
   all disj vi, vj: VM | all ni, nj: Cluster |
  ( (vj in vi.(vm_relation.vm_unidirectional_flow)) &&
  (vi in ni.host.vms) && (vj in nj.host.vms)
  ) =>
  ( exists_link_guard[ni,nj] &&
  (ni !in nj.^( cluster_relation . link ))
  ) }
```

5.4.3 Exemple - Le courtier vérifie la satisfiabilité des besoins non-fonctionnels

Le courtier vérifie que la configuration (le placement) qu'il a proposé satisfait les contraintes de placement et d'interconnexion et les propriétés de sécurité exigées par le client et qui sont décrites dans la figure 5.8 (page 89).
Cela est effectué en vérifiant des assertions Alloy. Chaque assertion exprime une propriété donnée. En la vérifiant, on a deux retours possibles :

- *Counterexample found. assert is invalid* : Alloy retourne un contre-exemple, alors cette propriété est violée et le placement proposé ne satisfait pas la propriété en question.
- *No counterexample found. assert may be valid :* Alloy ne trouve aucun contre-exemple concernant la propriété vérifiée. Dans ce cas elle peut être satisfiable par le placement proposé.

Dans la suite, nous illustrons à l'aide de notre exemple comment vérifier la satisfiabilité des contraintes de placement et d'interconnexion et des propriétés de sécurité en considérant le placement proposé par la phase 1. La figure 5.9 représente le place-

ment `firstSelection` en considérant les besoins fonctionnels et non-fonctionnels du client ainsi que les offres fonctionnelles et non-fonctionnelles des fournisseurs.

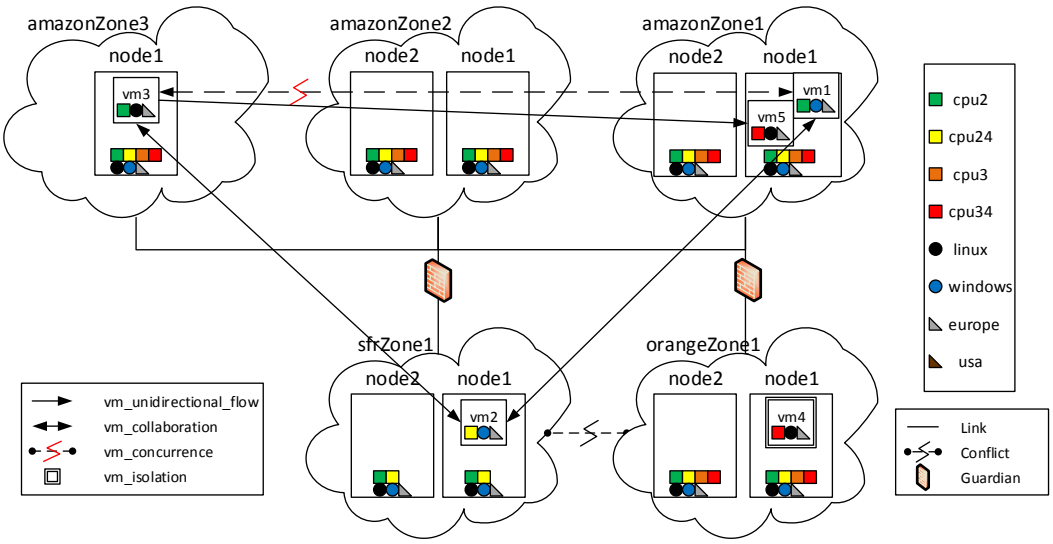

FIGURE 5.9 – *Exemple : placement des* MV *sur les grappes et contraintes non-fonctionnelles.*

A) Contraintes de placement : Le courtier vérifie que le placement qu'il a proposé en phase 1 respecte toutes les contraintes de placement posées par le client.

Grappes en conflit : Le courtier vérifie que les MV d'un client ne sont pas placées sur des grappes en conflit (Listing 5.3, page 90).
Alloy retourne un contre-exemple qui prouve que cette assertion n'est pas satisfiable. En effet, dans notre exemple (Figure 5.9), les deux MV `vm2` et `vm4` du même client sont placées sur deux grappes en conflit `sfrZone1` et `orangeZone1`.

Séparation des MV en concurrence : Le courtier a placé les MV en concurrence `vm1` et `vm3` sur deux grappes différentes donc cette propriété (exprimée dans Listing 5.5, page 91) est satisfiable et Alloy ne retourne pas de contre-exemple.

Isolation : La MV `vm4` qui est définie isolée, est placée seule sur le nœud physique `orangeZone1node1` (Figure 5.9, page 95). En vérifiant l'assertion du Listing 5.7 (page 91), Alloy ne retourne aucun contre-exemple, c'est-à-dire la propriété d'isolation est satisfiable.

B) Contraintes d'interconnexion : Afin d'assurer le bon fonctionnement du système client une fois déployé dans le nuage informatique, la communication entre les différentes MV du client doit être assurée. Les propriétés de sécurité que le client a exigées doivent être respectées. Pour autoriser ou interdire les communications entre les MV du client, les liens de flux doivent être présents entre les

grappes hébergeant les MV. Des gardiens doivent aussi être présents là où il le faut afin de contrôler et bloquer les flux indésirables.

Collaboration : La propriété (Listing 5.9, page 93) qui exprime l'obligation d'assurer des liens de flux entre les grappes hébergeant les MV en collaboration est satisfiable. En effet, `vm1` et `vm2` sont en collaboration et placées respectivement sur les grappes `amazonZone1` et `sfrZone1`. Ces deux grappes sont reliée par au moins un chemin constitué d'un lien de flux normal entre `amazonZone1` et `amazonZone2` et un lien de flux contrôlé entre `amazonZone2` et `sfrZone1`. La communication entre les deux grappes est donc assurée et par conséquent la communication entre les deux MV aussi. C'est le cas aussi pour les deux MV `vm2` et `vm3` qui sont en collaboration et placées respectivement sur `sfrZone1` et `amazonZone3`. Ces deux grappes sont reliées par au moins un chemin de liens assurant ainsi la communication entre les deux MV (voir figure 5.9)

Concurrence : La propriété (Listing 5.10, page 93) qui exprime l'obligation d'absence de chemins de flux non contrôlés entre les MV en concurrence n'est pas satisfiable. La figure 5.10 représente le contre-exemple retourné par Alloy et qui met en évidence la violation de cette propriété. Les deux MV `vm1` et `vm3` sont en concurrence et sont placées respectivement sur les grappes `amazonZone1` et `amazonZone3`. Ces deux grappes sont reliées par un chemin de liens non contrôlé qui passe à travers `amazonZone2`. L'existence d'un chemin de liens non contrôlé entre les deux grappes facilite la fuite d'information entre les MV en concurrence. Comme montré sur la figure 5.10, Alloy met seulement en évidence les entités du contre-exemple mais pas le chemin de liens qui les relie. Il s'agit ici des MV `vm1` et `vm3` et des grappes `amazonZone1` et `amazonZone3`.

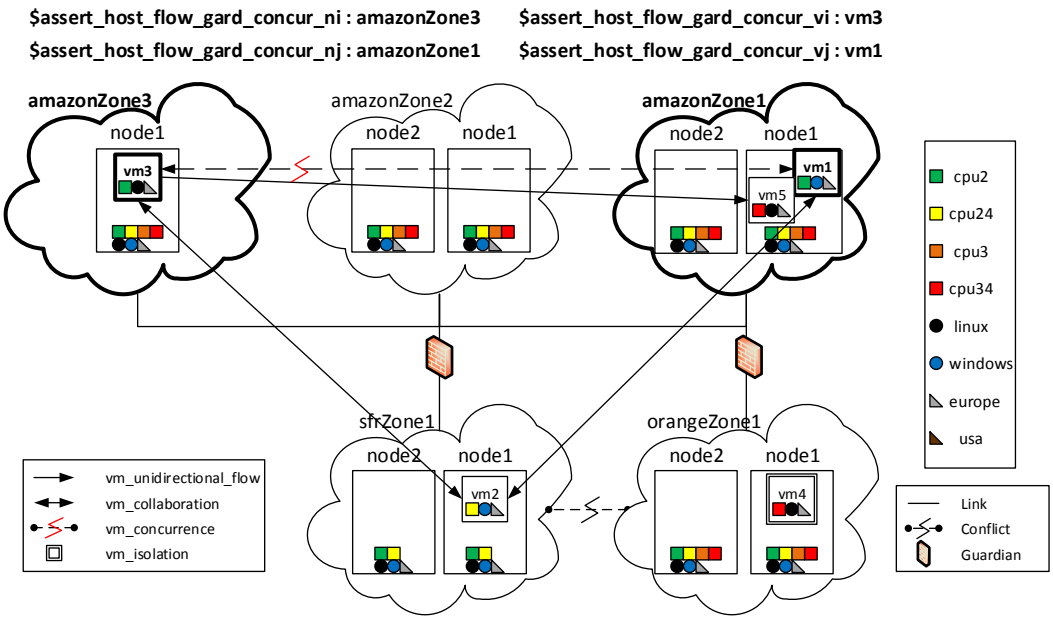

FIGURE 5.10 – *Exemple : contre-exemple Alloy prouve la non-satisfiabilité de la propriété de concurrence.*

Flux à sens unique : Le client a défini un flux à sens unique entre les MV `vm3` et `vm5`. Ces deux MV sont placées respectivement sur les grappes `amazonZone3` et `amazonZone1` (Figure 5.9). Comme dit précédemment, ces deux grappes sont reliées par un chemin de liens non-contrôlé. Cela assure le flux d'information en provenance de `amazonZone3` et à destination de `amazonZone1`, mais autorise aussi le flux dans le sens contraire qui est supposé être interdit. Le contre-exemple retourné par Alloy met en évidence les entités `vm3` et `vm5`, `amazonZone3` et `amazonZone1` et prouve la non-satisfiablité de cette propriété exprimée dans Listing 5.11 (page 94).

Solutions suivantes : Étant donné que la solution de placement précédente ne satisfait pas tous les besoins non-fonctionnels du client, le courtier reprend d'autres solutions possibles obtenues de la phase 1 et re-vérifie à nouveau la satisfiabilité de toutes les contraintes du client. La figure 5.11 représente un placement proposé et vérifié par le courtier. Ce placement satisfait tous les besoins fonctionnels et non-fonctionnels du client. Les contraintes de placement et d'interconnexion et les propriétés de sécurité que le client a exigées sont toutes vérifiées et satisfiables.

Les critères fonctionnels de toutes les MV sont satisfaits. Le courtier propose le placement tel que : `vm1` et `vm5` sont affectées au nœud `orangeZone1node2`. `vm2` est affectée au noeud `amazonZone2node1` et `vm3` au nœud `amazonZone2node2`. `vm4` est placée seule sur le nœud `orangeZone1node1`

Les critères non-fonctionnels sont aussi satisfiables. Cela concerne les propriétés de sécurité, les contraintes de placement et d'interconnexion :

- Conflit entre grappes : aucune MV de ce client n'est placée sur la grappe `sfrZone1` qui est en conflit avec `orangeZone1`, cette propriété est donc respectée.
- MV en collaboration : les MV `vm2` et `vm3` sont placées sur la même grappe `amazonZone2`, donc elles peuvent communiquer via un réseau interne à cette grappe. Les MV `vm1` et `vm2` sont placées sur différentes grappes `orangeZone1` et `amazonZone2`. Ces deux grappes sont reliées par le chemin de liens qui passe via `amazonZone1`, donc la communication entre les deux grappes est assurée et par conséquent la communication entre `vm1` et `vm2` aussi.
- MV en concurrence : les MV en concurrence `vm1` et `vm3` sont placées sur différentes grappes `orangeZone1` et `amazonZone2`. Ces deux grappes sont reliée par un chemin de liens contrôlé. Il est constitué d'un lien normal entre `amazonZone2` et `amazonZone1` et d'un lien contrôlé entre `amazonZone1` et `orangeZone1`. Le gardien placé sur le dernier lien contrôlé permet de bloquer les flux entre les deux MV en concurrence, après avoir ajouté des règles de filtrage dessus.
- MV avec un flux d'information unidirectionnel : `vm3` et `vm5` sont placées sur différentes grappes `amazonZone2` et `orangeZone1`. Le chemin de liens contrôlé qui relient les deux grappes permet d'assurer les communications provenant de `vm3` à destination de `vm5`. En ajoutant des règles de filtrage, les flux du sens inverse sont bloqués au niveau du gardien placé entre `amazonZone1` et `orangeZone1`.
- MV isolée : `vm4` est isolée et est placée seule sur le nœud `orangeZone1node1`. La propriété d'isolation est donc respectée.

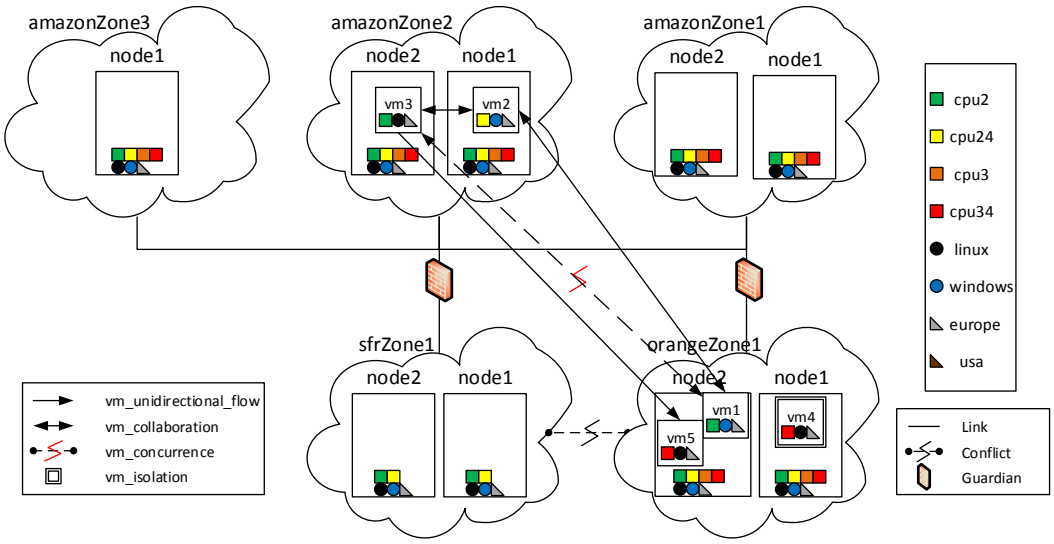

FIGURE 5.11 – *Exemple : placement qui satisfait tous les besoins fonctionnels et non-fonctionnels.*

Ce placement est nommé l'ensemble `secondSelection`. L'ensemble contient les affectations des MV aux nœuds physiques/grappes/fournisseurs et décrit globalement la configuration de placement et les emplacements des gardiens qui peuvent assurer les besoins du client.
L'ensemble `secondSelection` est proposé au client. Ce dernier l'accepte s'il est satisfait du placement. Le courtier procède à l'établissement des contrats SLA entre les fournisseurs qui vont héberger les MV du client, le courtier et le client. Ensuite, le courtier coopère avec les fournisseurs pour le déploiement des MV et la mise en place des mécanismes de sécurité nécessaires comme exigé par le client. Ces derniers peuvent être des règles de filtrage configurées sur les gardiens. Par exemple, des règles peuvent être configurées afin de bloquer les flux d'information indésirables entre les MV en concurrence.

Synthèse : On peut avoir un cas où après avoir testé toutes les solutions possibles (cela est possible car nous considérons que l'ensemble des offres est fini), le courtier ne trouve pas de solution qui satisfait tous les besoins fonctionnels et non-fonctionnels du client. Dans ce cas, en prenant en compte une solution qui ne satisfait pas tous les besoins non-fonctionnels et les contre-exemples retournés par le courtier, le client peut changer ses contraintes ou peut accepter cette solution telle qu'elle est.
Par exemple le client peut accepter une solution où la contrainte « héberger toutes les MV chez le même fournisseur » n'est pas respectée.
Le client peut aussi demander au courtier de lui proposer un placement de gardiens virtuels entre les MV en concurrence ou avec un lien unidirectionnel afin de bloquer les flux indésirables.

5.5 Conclusion

Dans ce chapitre, nous avons présenté notre mécanisme de placement des ressources client chez les fournisseurs de nuage informatique. Ce placement est constitué de plusieurs étapes. La première consiste à affecter les MV du client à des nœuds physiques qui peuvent satisfaire les besoins fonctionnels des MV. La deuxième consiste à vérifier que les affectations proposées par l'étape précédente ainsi que la configuration des réseaux et gardiens reliant les grappes des fournisseurs satisfont les besoins non-fonctionnels du client. Ce mécanisme est effectué en utilisant l'analyseur Alloy qui retourne à chaque étape des contre-exemples dans le cas où les besoins du client ne sont pas satisfiables.
Si le courtier trouve un placement qui satisfait tous les besoins du client, et avant de procéder au déploiement, une étape supplémentaire est nécessaire. Il s'agit de la vérification de la politique de contrôle d'accès des utilisateurs finaux des ressources du client. Le courtier analyse les spécifications de la politique du client et lui signale tout conflit de spécifications. Cette étape fait l'objet du chapitre 6.

6 Contrôle d'accès et propriétés de sécurité

Sommaire

6.1 Introduction 101
6.2 Modèle de base du ACSP-RSM 102
6.2.1 Cas d'usage : système hospitalier 104
6.2.2 Les besoins de contrôle d'accès 106
6.2.3 Les besoins de propriétés de sécurité 107
6.2.4 Conflits et redondances des besoins de CA et de PS 109
6.3 Spécification et analyse formelles des politiques ACSP 110
6.3.1 Spécification du modèle de base de ACSP-RSM 111
6.3.2 Spécification des règles d'autorisation en ABAC 113
6.3.3 Spécification des relations entre les objets et entre les sujets 115
6.3.4 Spécification des propriétés de sécurité requises 116
6.3.5 Analyse formelle des politiques ACSP 117
6.4 Conclusion 124

6.1 Introduction

Une politique de contrôle d'accès est un des éléments essentiels de la sécurisation d'un système. Elle est constituée de plusieurs règles d'autorisation (règles de contrôle d'accès ou de privilèges) dont le nombre dépend de la taille du système. Plus le système est grand, notamment dans le cas des systèmes nuagiques, plus le risque d'incohérence de la politique est important. Les règles de contrôle d'accès peuvent contenir des contradictions, c'est-à-dire, permettre et interdire à la fois un accès à une ressource dans les mêmes conditions. C'est pour cela qu'un moyen de vérification automatique des politiques de contrôle d'accès dans le nuage informatique demeure nécessaire. Nous proposons dans ce chapitre d'analyser les politiques de sécurité du client avant même le déploiement des ressources demandées dans le nuage informatique. L'objectif est de montrer au client tous les conflits de la politique avant de la déployer sur le système. Le client doit proposer une nouvelle politique exempte de conflits.

Quand la phase 2 du processus de placement dans le mécanisme de courtage est finie, le courtier retourne une solution de placement des ressources au client. Après que le client ait donné son accord sur ce placement au courtier, il peut définir des propriétés de sécurité et des politiques de contrôle d'accès spécifiques au système à déployer dans le nuage informatique. Dans ce chapitre, nous présentons un modèle de spécification semi-formel permettant d'exprimer ces propriétés avec un niveau élevé de granularité.
Nous définissons d'abord le modèle de base qui décrit les systèmes clients ACSP-RSM (*Access Control and Security Properties Requirements Specification Model*). Ce modèle peut décrire une politique de sécurité et de contrôle d'accès basée sur les attributs (ABAC).
Le client utilise ce modèle pour spécifier les composants de son système et les relations entre ces composants. Les relations peuvent décrire des privilèges d'accès ou des propriétés de sécurité. Par la suite, nous utilisons Alloy pour analyser les spécifications du client. Notre analyse sert en premier lieu à détecter les conflits entre les règles de contrôle d'accès et en second lieu à vérifier la satisfiabilité des propriétés de sécurité exigées par le client si cette politique de contrôle d'accès est appliquée.

Nous présentons, dans la section 6.2, le modèle de base de description des besoins du client en ce qui concerne le contrôle d'accès et les propriétés de sécurité. Le client utilise ce modèle semi-formel pour spécifier ses politiques de sécurité et de contrôle d'accès. Dans la section 6.3, nous traduisons les descriptions des besoins client en spécifications formelles en utilisant le langage Alloy. Les spécifications sont analysées pour détecter les conflits de descriptions. Enfin, dans la section 6.4, nous concluons ce chapitre.

6.2 Modèle de base du ACSP-RSM

Nous présentons le modèle semi-formel ACSP-RSM [71] qui permet de décrire les composants du système client, c'est-à-dire, les ressources et les utilisateurs ainsi que les besoins du client en termes de contrôle d'accès et de propriétés de sécurité.
ACSP-RSM est basé sur le modèle ABAC. Nous avons choisi d'utiliser le modèle ABAC pour spécifier les politiques de contrôle d'accès. ABAC est un modèle très général et peut être vu comme une généralisation de la plupart des modèles traditionnels existants tels que : RBAC, MAC et DAC [87]. Il permet une grande flexibilité et expressivité des politiques de contrôle d'accès, surtout dans le cas des besoins complexes des systèmes dans le nuage informatique [177] ainsi que dans le contexte de santé (*health-care context*) [146]. Il est facilement extensible et modifiable car il est basé sur les attributs. La hiérarchie et la combinaison des attributs permet de définir des contraintes d'autorisation de façon plus souple et dynamique [136, 51].
Un modèle ACSP-RSM d'un système se définit à l'aide des éléments suivants :

Sujet : L'ensemble $\mathcal{S}u$ des entités actives qui utilisent les objets du système. Cela inclut les utilisateurs ainsi que les applications et les services exécutés via des utilisateurs.

Objet : L'ensemble $\mathcal{O}b$ des entités passives qui sont manipulées par les sujets du système. Cela inclut les ressources, les applications et les services du système.

Contexte : L'ensemble $\mathcal{C}x$ désigne les conditions de l'environnement au moment où l'accès à l'objet sera effectué. Par exemple la date du jour, la localisation de l'utilisateur ; l'accès peut être autorisé à partir d'une zone géographique mais pas une autre.

Attribut : L'ensemble $\mathcal{A}t$ représente des caractéristiques des sujets, des objets ou des contextes. Ils sont représentés par des paires (<nom d'attribut>, <valeur d'attribut>). Par exemple,

- L'identifiant est un attribut qui définit l'identité du sujet.
- Le rôle $\mathcal{R}$ est un attribut affecté à un utilisateur selon ses compétences, son autorité et sa responsabilité dans le système.
- Les attributs associés à un objet servent à spécifier des caractéristiques comme l'emplacement où est alloué cet objet ou l'adresse IP d'une MV.
- La date (ou le temps) représente un attribut lié aux conditions d'environnement et sert à spécifier le temps exact quand une action sur un objet est autorisée.
- Niveau de nuage informatique (Cloud Level) $\mathcal{CL}$ qui consiste en IaaS, PaaS ou SaaS. Cet attribut est utilisé pour séparer les privilèges entre les différents niveaux du nuage informatique.

Action : L'ensemble $\mathcal{A}c$ des opérations que les sujets peuvent effectuer sur les objets. Cet ensemble inclut les opérations élémentaires telles que : `lire`, `écrire` et `exécuter`, les opérations de gestion comme : `créer`, `supprimer` et `partager`. Les opérations de gestion peuvent être exprimées au moyen des opérations élémentaires. Par exemple : les opérations de création et de suppression d'un fichier peuvent être considérées comme une opération d'écriture du fichier.

Le **partage** est une opération où le sujet donne un de ses droits d'accès à un objet (lecture, écriture ou exécution) à un autre sujet. Autrement dit, l'opération de partage donne le droit à des sujets d'accéder à des objets. Ce droit, ils ne l'avaient pas auparavant. L'objectif de l'action de partage des privilèges entre des sujets est de contribuer ensemble aux même tâches.

On distingue aussi un ensemble $\mathcal{PS}$ de **propriétés de sécurité** que le client peut exiger. Il s'agit des privilèges qui peuvent être autorisés ou interdits dans un système. Ces propriétés peuvent concerner un simple privilège d'accès. Par exemple, interdire l'accès en lecture à un objet consiste en la propriété de `confidentialité`. La

propriété de `séparation de privilèges` consiste en la séparation des privilèges d'accès sur les sujets selon leurs rôles dans le système.

Politique : $\mathcal{P}ol$ est un ensemble de règles de contrôle d'accès (d'autorisation) et/ou des règles de sécurité que le client spécifie dans le but de contrôler les accès dans son système. Une règle d'autorisation ou de sécurité est donnée par une relation entre les attributs de sujets, les attributs d'objets et les attributs de contexte afin de déterminer si l'accès est autorisé ou pas. Pour cela, on définit l'ensemble $\mathcal{P}$ des **permissions** qui inclut la permission (`permis`) et l'interdiction (`interdit`) d'accès.

On peut distinguer trois types de politiques de contrôle d'accès ou de sécurité. Le premier type où par défaut tous les accès sont interdits, et on spécifie des règles d'autorisation (permission). Le deuxième type où par défaut tous les accès sont autorisés et on spécifie des règles d'interdiction. Le troisième type où on spécifie à la fois les règles d'interdiction et d'autorisation.
Dans nos travaux, on considère le premier type de politique, c'est-à-dire, par défaut tous les accès dans le système sont interdits. Le client spécifie seulement les règles pour permettre quelques accès dans le système.

Dans la suite, nous présentons la méthode suivant laquelle le client spécifie ses politiques de contrôle d'accès et politiques de sécurité, en utilisant les éléments du ACSP-RSM. Avant cela, nous présentons un cas d'usage qui va servir d'exemple pour illustrer notre démarche dans ce chapitre.

6.2.1 Cas d'usage : système hospitalier

On considère un système d'information (SI) d'un système hospitalier (*health-care system*) qui est largement utilisé dans la littérature [177, 120]. Les systèmes hospitaliers migrent de plus en plus dans le nuage informatique car il leur offre de grandes capacités de calculs, de traitement de données médicales (radios, analyses et autres) et de stockage. Il permet à plusieurs entités médicales (centres de recherches, laboratoires, hôpitaux, etc.) de collaborer afin de prendre des décisions et de suivre l'évolution des soins et visualiser et évaluer les résultats.
Un tel système est exposé à des problèmes de sécurité étant donné que les données personnelles et médicales sont partagées entre plusieurs entités. La gestion des accès à ces données est primordiale.
Dans cet exemple, on considère un service de consultation générale, un service de radiologie et un laboratoire d'analyse. Chaque service a un ensemble de ressources et des utilisateurs (sujets). Des règles d'autorisation doivent être définies pour gérer les accès des sujets aux ressources. Il s'agit de ressources sensibles. La perte, la divulgation ou la modification non légitime des données ne sont pas tolérées. De plus, toute erreur commise sur un dossier médical peut mener à des situations compliquées. Par

exemple, le fait de perdre des résultats d'analyse empêche le diagnostic urgent et peut entrainer du danger sur la santé d'un patient. Le client (la direction des ressources numériques du système hospitalier) détaille les composants de son système. La figure 6.1 présente un exemple simple d'une partie du système hospitalier. Ce dernier est réellement constitué d'un système d'information compliqué. Il rassemble les données des patients, les données des employés, les services médicaux, le service des ressources humaines, etc.
Nous considérons dans ce cas d'usage que le client peut utiliser différents services de différents niveaux de nuage informatique. Il peut louer des MV au niveau IAAS, déployer des applications au niveau PAAS ou utiliser des services au niveau SAAS. Dans ce cas, le système hospitalier du client est déployé sur différents niveaux du nuage informatique.
Les MV que le client a demandées dans les deux chapitres précédents servent à héberger des applications, sauvegarder et traiter des données. L'application `appAnalyse` est utilisée dans le service des analyses médicales. L'application `appRadio` est utilisée dans le service de radiologie. D'autres applications sont utilisées dans le service des consultations générales. Par exemple, une application peut être utilisée pour classer et gérer les dossiers médicaux (*patient data management*). Le client utilise des espaces de stockage pour sauvegarder des données internes du système hospitalier. On les appelle des EMR (E*lectronic* M*edical* R*ecords*). On ne considère dans cet exemple que les EMR dossierMedical, dossierAnalyse et dossierRadio. Les données des dossiers médicaux des consultations générales sont stockées dans l'EMR dossierMedical. Les données des analyses (resp. des radios) sont stockées dans l'EMR dossierAnalyse (resp. dossierRadio). Les ressources du système sont caractérisées par le niveau de sensibilité (faible, critique) selon si la ressource contient des informations critiques ou pas.

Le client définit les sujets qui vont utiliser les ressources du système. Dans cet exemple, on attribue aux sujets des caractéristiques. Les rôles docteur, radiologue, analyste, administrateur système et d'autres sont attribués aux sujets selon le poste qu'ils occupent. L'expertise (débutant, intermédiaire, expert) leur est aussi attribuée selon leur niveau d'expertise dans ce même système hospitalier. L'écriture, la lecture et l'exécution sont définies comme des opérations autorisées dans le système. Les flèches dans la figure 6.1 représentent les privilèges définis par le client. Dans ce chapitre, nous n'allons pas illustrer tout cet exemple, plus de détails sont présentés dans [71].

Le client doit définir les règles d'accès des sujets aux ressources et des propriétés de sécurité comme définies dans les sous-sections 6.2.2 et 6.2.3 .

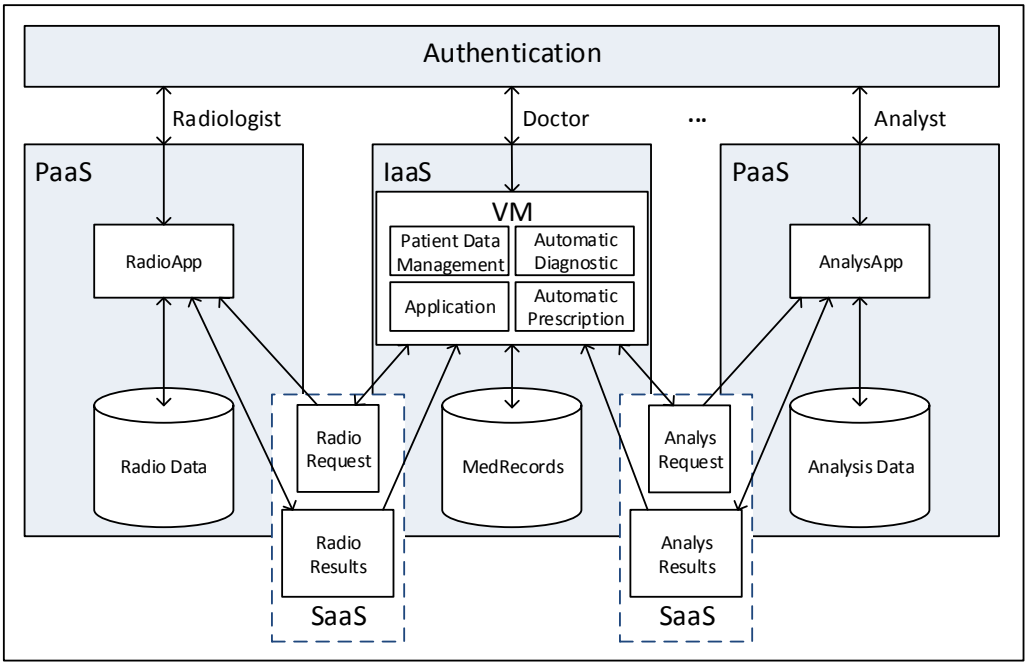

FIGURE 6.1 – *Architecture du SI d'un « système hospitalier »*

6.2.2 Les besoins de contrôle d'accès

Une politique est constituée d'un ensemble de règles de contrôle d'accès et/ou de sécurité. Une règle de contrôle d'accès est spécifiée par l'une des deux formules suivantes :

`ca(sujAtt,objAtt,action,ctxtAtt,permission)`

`caPartage(sujAtt,objAtt,ctxtAtt,sujAttCible,actionCible,permission)`, tels que :

- La règle `ca` concerne une règle de contrôle d'accès et la règle `caPartage` concerne une règle de partage de privilège.
- `sujAtt, objAtt, ctxtAtt, sujAttCible` $\subset \mathcal{A}tt$
  - `sujAtt, sujAttCible` sont des ensembles d'attributs de sujets
  - `objAtt` est un ensemble d'attributs d'objets
  - `ctxtAtt` est un ensemble d'attributs de contexte,
- `action, actionCible` $\subset \mathcal{A}ct$,
- `permission` $\subset \mathcal{P}$.

Ces règles ont les significations suivantes :

- La règle `ca` : permettre à un utilisateur d'effectuer une action sur un objet dans un contexte donné si l'utilisateur vérifie les couples {att, valeur} de l'ensemble sujAtt, l'objet vérifie les couples {att, valeur} de l'ensemble objAtt et le contexte vérifie les couples {att, valeur} de l'ensemble ctxtAtt.
- La règle `caPartage` : permettre à un utilisateur de partager un objet avec un

utilisateur cible en mode actionCible (lecture, écriture ou exécution), dans un contexte donné si l'utilisateur vérifie les couples {att, valeur} de l'ensemble sujAtt, l'objet vérifie les couples {att, valeur} de l'ensemble objAtt, le contexte vérifie les couples {att, valeur} de l'ensemble ctxtAtt et l'utilisateur cible vérifie les couple {att, valeur} de l'ensemble sujAttCible.

Quand un élément n'est pas utilisé pour exprimer une règle, on le remplace par le symbole « _ ». On utilise le symbole « / » pour lister plusieurs choix possibles d'un élément et le symbole « * » pour rassembler toutes les valeurs possibles d'un élément. Par exemple, la règle :

- `ca([role:docteur],[objId:dossierMedical],lire/ecrire,[08:00,17:00],permis)` exprime qu'il est autorisé aux utilisateurs ayant le rôle « docteur » de lire et d'écrire les objets identifiés par « dossierMedical ». Cela est autorisé durant la journée entre 08h00 et 17h00.
- `caPartage([role:docteur],[objId:dossierMedical],*, [role:docteur],lire,permis)` exprime qu'il est autorisé aux utilisateurs ayant le rôle « docteur » de partager les objets identifiés par « dossierMedical » en mode lecture avec d'autres utilisateurs ayant le rôle « docteur », quelque soit le contexte.

Quand le symbole « * » est appliqué à la place des sujets, la règle est applicable pour tous les sujets du système client (pareil pour les autres champs).

Il peut être difficile d'exprimer la totalité d'une politique avec une granularité aussi fine que celle des règles de contrôle d'accès. Par exemple, il peut être compliqué d'interdire des sujets appartenant à un même projet de faire toutes les opérations possibles sur un objet (propriété de sécurité : séparation des privilèges). Dans ce cas, nous proposons une alternative qui consiste à exprimer des besoins en termes de propriétés de sécurité ($\mathcal{PS}$). Les règles de contrôle d'accès et les règles de sécurité se complètent afin d'assurer la protection d'un système.

6.2.3 Les besoins de propriétés de sécurité

Les propriétés de sécurité peuvent remplacer plusieurs règles de contrôle d'accès et peuvent décrire des permissions d'accès directs et indirects. Une règle de propriété de sécurité s'exprime de la façon suivante :

`sec(sujAtt,objAtt,propSec,ctxtAtt,appliquer/lever)`

- `sujAtt, objAtt` $\subset \mathcal{Att}$. `sujAtt` est un ensemble d'attributs de sujets et `objAtt` est un ensemble d'attributs d'objets.
- `propSec` $\subset \mathcal{PS}$ est une propriété de sécurité.
- `ctxtAtt` est un ensemble d'attributs de contexte.

Cette règle applique ou lève la propriété de sécurité « propSec » sur les objets vérifiant objAtt, quand l'accès est demandé par un sujet vérifiant sujAtt. Contrairement aux

permissions des règles de contrôle d'accès (permis, interdit) qui permettent ou interdisent un accès, une règle de propriété de sécurité applique une propriété ou la lève (c'est-à-dire, appliquer son contraire). Cela revient en général à autoriser ou interdire des accès comme nous expliquons dans la suite.
Nous pouvons considérer que le client applique par défaut une règle de sécurité sur l'ensemble des objets sensibles concernant tous les sujets du système. Ensuite, il autorise seulement quelques sujets à lire ces objets en utilisant des règles qui lèvent cette propriété de sécurité.

Les propriétés que nous traitons sont :

- **Confidentialité :** Pour protéger les informations contre la divulgation aux sujets non-autorisés, le client peut appliquer une règle de confidentialité sur ces informations. Cette règle peut être appliquée concernant l'ensemble des sujets du système. Si le client souhaite autoriser un sujet à lire un objet, il peut lever la règle de confidentialité sur cet objet concernant ce sujet. Par exemple :

  ```
  sec([role:docteur],[objId:dossierAnalyseRes],confidentialite,
      [niveauNuage:SaaS],lever)
  ```

  Cette règle lève la confidentialité pour les sujets ayant le rôle « docteur » sur l'objet identifié par « dossierAnalyseRes », c'est-à-dire, les sujets ayant ce rôle peuvent lire « dossierAnalyseRes ». Cette action est autorisée au niveau de nuage informatique SaaS.
- **Intégrité :** Pour protéger l'information contre toute modification par des sujets non autorisés, le client peut appliquer une règle d'intégrité sur cette information. Ensuite, il peut aussi autoriser des sujets à modifier cette information en utilisant une règle qui lève cette propriété d'intégrité. Par exemple :

  ```
  sec([role:docteur],[objId:dossierAnalyse],integrite,
      [niveauNuage:SaaS],lever)
  ```

  Cette règle lève l'intégrité sur l'objet identifié par « dossierAnalyse » pour les sujets ayant le rôle « docteur », c'est-à-dire, les sujets ayant le rôle « docteur » ont le droit de modifier (écrire) « dossierAnalyse ».
- **Séparation de privilèges :** La séparation de privilèges est le fait de distribuer les droits d'accès (privilèges) à un objet sensible sur différents sujets. Cela réduit le risque de violation de confidentialité et d'intégrité. Par exemple séparer les droits d'écriture, de lecture et d'exécution de `appAnalyse`, au niveau de nuage informatique PaaS.

  ```
  sec(*,[objId:appAnalyse],separation_privilege,[niveauNuage:PaaS],appliquer)
  ```

  Cette règle applique la séparation de privilège sur l'application `appAnalyse` pour tous les sujets du système. Cela veut dire que quelque soit le sujet, il ne peut pas avoir à la fois les droits de modifier, lire et d'exécuter cette application.
- **Trusted path execution (TPE) :** Les utilisateurs ne peuvent utiliser que les applications de confiance. Cette propriété est représentée dans la règle suivante par « tpe ».

```
sec([role:analyste],[objId:appAnalyse],tpe,[niveauNuage:PaaS],appliquer)
sec([role:radiologue],[objId:appRadio],tpe,[niveauNuage:PaaS],appliquer)
```

Ces deux règles expriment des propriétés de TPE. L'analyste peut seulement utiliser `appAnalyse` et le radiologue peut seulement utiliser `appRadio`. L'utilisation de ces applications se fait au niveau PaaS.

6.2.4 Conflits et redondances des besoins de CA et de PS

Les règles de contrôle d'accès et de propriétés de sécurité peuvent être analysées afin de détecter les conflits de consistance ou de redondance [118, 90] ou d'autres types de conflit. Ces conflits devront être signalés au client qui a exprimé la politique. Voici quelques types de conflits :

Conflits de modalité : Un conflit de modalité survient quand plusieurs décisions contradictoires sont attribuées au même sujet, objet, action et contexte [129]. Par exemple interdire un sujet ayant un rôle donné l'accès à un dossier et autoriser ce même sujet l'accès au même dossier.
Par exemple, la première règle des deux règles suivantes autorise et la deuxième interdit aux utilisateurs ayant le rôle docteur de lire l'objet dossierAnalyseRes.

```
ca([role:docteur],[objId:dossierAnalyseRes],lire,[niveauNuage:SaaS],permis)
sec([role:docteur],[objId:dossierAnalyseRes],confidentialite,[niveauNuage:SaaS],
    appliquer)
```

Conflits de redondance : Quand le client définit des règles redondantes, le temps de prise de décision d'accès peut être important. S'il assigne des priorités à sa politique de contrôle d'accès, cela peut mener à des règles qui ne sont jamais appliquées [129]. Ces cas peuvent être considérés comme une redondance plus qu'un conflit. Cependant, il est important de détecter cette redondance, la corriger et l'éviter pour optimiser la politique.

Par exemple : le client définit les deux règles suivantes :

```
ca([role:analyste],[objId:appAnalyse],exec,*,_,_,permis)
ca([role:analyste],[objId:appAnalyse],exec,[12:00,14:00],_,_,interdit)
```

On peut considérer des politiques avec des priorités en ajoutant dans les règles d'autorisations un attribut qui signifie la priorité d'appliquer une règle.
Si une demande d'accès a lieu et que plusieurs règles de contrôle d'accès et/ou de sécurité sont applicables, on applique la règle qui est la plus prioritaire.
Si le client définit une priorité pour les règles autorisant l'accès, la première règle devient plus prioritaire que la deuxième. Cette dernière qui interdit à l'analyste d'exécuter `appAnalyse` entre 12h00 et 14h00 ne sera jamais appliquée.
Les propriétés de sécurité peuvent être factorisées en plusieurs règles de contrôle d'accès, ainsi la redondance est plus complexe car on doit considérer les PS et les règles de contrôle d'accès ensemble.

Conflits de contexte : (temporel, spatial ou historique). Dans une politique, le client peut aussi introduire des confusions dans la description du contexte entre les règles qui s'appliquent sur les mêmes sujets et objets.
L'exemple suivant montre le conflit temporel : on interdit au docteur de lire les dossiersMedicaux entre 09h00 et 11h00, et dans une autre règle on l'autorise à lire ces objets quotidiennement entre 08h00 et 11h00.

```
ca([role:docteur],[objId:dossierMedical],lire,
    [niveauNuage:SaaS]daily:[09:00, 11:00],_,_,interdit)
ca([role:docteur],[objId:dossierMedical],lire,
    [niveauNuage:SaaS]daily:[08:00, 11:00],_,_,permis)
```

Les règles de l'exemple suivant permettent au sysAdmin de modifier l'application `appAnalyse` en même temps que l'utilisateur ayant le rôle testeur est autorisé à l'exécuter. Cela peut introduire des incohérences soit dans le développement, soit dans le test de cette application.

```
ca([role:sysAdmin],[objId:appAnalyse],ecrire,daily:[08:00, 10:00],_,_,permis)
sec([role:testeur],[objId:appAnalyse],tpe,[niveauNuage:PaaS],appliquer)
```

Conflits issus du partage : Quand le partage des privilèges est autorisé, des conflits avec d'autres règles peuvent apparaître. Dans l'exemple suivant le docteur peut partager dossierRadio avec l'analyste en mode lecture. Cela lui donne le droit de lecture et crée un conflit avec la deuxième règle qui lui interdit la lecture de dossierRadio.

```
caPartage([role:docteur],[objId:dossierRadio],[niveauNuage:SaaS],
   [role:analyste],lire,permis)
sec([role:analyste],[objId:dossierRadio],confidentialite,[niveauNuage:PaaS],
     appliquer)
```

Une des solutions pour lever les confusions et conflits est de considérer un mécanisme de décision basé sur une stratégie de priorité des règles bien définie. Cela s'applique au moment où une requête d'accès fait appel à plusieurs règles qui peuvent être conflictuelles. Nous nous intéressons seulement à la détection de ces conflits. La résolution des conflits sort du cadre de notre travail.

6.3 Spécification et analyse formelles des politiques ACSP

Dans cette section nous présentons une méthode de spécification formelle des politiques de sécurité et d'analyse de ces spécifications. Nous utilisons le langage Alloy pour la description des spécifications et l'analyseur Alloy pour les analyser.

Dans le chapitre 4, le client décrit le système qu'il souhaite migrer dans le nuage informatique. Dans le chapitre 5, le courtier lui propose un placement des ressources (MV) chez les fournisseurs de nuage informatique. Quand le client accepte ce placement, il définit une politique de sécurité fine pour sécuriser les différents composants

de ses ressources.
Il décrit d'abord les ressources de chaque MV, par exemple s'il va installer des applications et stocker des données sur la MV, il doit les spécifier et leur attribuer des valeurs d'attributs. Le client décrit aussi les sujets qui vont utiliser les ressources et leur affecte des valeurs d'attributs. Par exemple, il affecte des rôles aux utilisateurs (sujets). Ensuite, il décrit les autorisations d'accès en fonction des attributs des sujets, des attributs des ressources et des attributs de contexte.
Ces descriptions sont faites en utilisant le modèle semi-formel ACSP-RSM défini dans la section 6.2 qui est plus facile à utiliser que Alloy. Ensuite, les spécifications du client sont traduites automatiquement en spécifications Alloy. Cette étape de traduction n'est pas encore réalisée.

Dans la suite nous présentons la méthode de spécification formelle des composants et caractéristiques du système client, ainsi que les politiques de contrôle d'accès et de sécurité. Ensuite nous présentons la méthode d'analyse de ces politiques et de détection des conflits. Nous illustrons toutes les étapes en utilisant l'exemple du système hospitalier.

6.3.1 Spécification du modèle de base de ACSP-RSM

Les éléments de base du modèle ACSP-RSM sont les sujets (Sujet), les objets (Objet), les conditions d'environnement (Contexte), les actions (Action) et les attributs (Attribut). On les décrit en Alloy comme présenté dans Listing 6.1.

Listing 6.1 –

```
abstract sig Sujet{}
abstract sig Objet{}
abstract sig Contexte{}
abstract sig Action{}
abstract sig  Attribut {}
```

Spécifications d'un cas d'usage : Dans le cas d'usage du système hospitalier, le client définit un ensemble de ressources (applications, bases de données, documents) et un ensemble de sujets.

Les données de ce système médical sont de nature sensible. On ne tolère pas la perte de données, la divulgation de données à des sujets non-autorisés ou la modification des données par des sujets non-autorisés.
Le client spécifie ses besoins en terme de politique de contrôle d'accès et de sécurité, c'est-à-dire qu'il spécifie les règles d'accès aux ressources du système hospitalier. L'objectif est d'analyser ses besoins (politique de contrôle d'accès et de propriétés de

sécurité) pour détecter les inconsistances et les conflits ainsi que vérifier la satisfiabilité des propriétés de sécurité par rapport à la politique spécifiée pour ce système.

Spécifications des attributs : On définit quelques noms (types) d'attributs ainsi que leurs valeurs possibles dans Listing 6.2. Pour les sujets, on définit le type d'attribut `Role` avec les valeurs possibles `docteur, infirmier, analyste, radiologue, sysAdmin, developpeur`. On définit l'attribut `Expertise` avec `debutant, intermediaire, expert` comme valeurs possibles.

Listing 6.2 –

```
abstract sig Role, Expertise extends Attribut{}
one sig docteur, infirmier , analyste, radiologue, sysAdmin, developpeur extends
    Role{}
one sig debutant, intermediaire, expert extends Expertise{}
```

De la même façon, on définit pour les objets les types d'attributs `ObjId, Objectif, Sensibilite`. L'attribut `ObjId` peut prendre comme valeur `dossierMedical, dossierAnalyse, dossierRadio, dossierChir, appAnalyse, appRadio` ou `secModule`. L'attribut `Objectif` a `consultation, chirurgie, radio, analyse, pediatrie, appMaj, secMaj` comme valeurs possibles. `critique, faible` sont les valeurs que peut prendre l'attribut `Sensibilite`. Les attributs de contexte sont des caractéristiques de l'environnement dans lequel l'opération demandée sera effectuée. On considère le niveau de nuage informatique `CNiveau`, le moment de la journée `DateJour` quand le sujet effectue l'opération, et la localisation où l'opération sera effectuée `Localisation`. `saas, paas, iaas` sont les valeurs de l'attribut `CNiveau`. `am, pm` représentent l'attribut `DateJour`. `centre, nord, ouest, est` sont les valeurs possibles de l'attribut `Localisation`.

Dans Listing 6.3, on étend les types de base du modèle ACSP pour définir des éléments spécifiques au système hospitalier du client. On définit alors `SujetMed, ObjetMed` et `ContexteMed`. On leur attribue les types d'attributs comme suit :

Listing 6.3 –

```
abstract sig SujetMed extends Sujet{
  role: one Role,
  expertise: one Expertise}
abstract sig ObjetMed extends Objet{
  objId: one ObjId,
  objectif : one Objectif,
  sensibilite : one Sensibilite}
abstract sig ContexteMed extends Contexte{
  cNiveau: one CNiveau,
  dateJour: one DateJour,
  localisation : one Localisation}
```

Dans Listing 6.4, on définit les opérations (actions) de lecture, écriture et exécution que les sujets peuvent effectuer dans ce système.

Listing 6.4 –

```
one sig lire , ecrire , executer in Action{}
```

6.3.2 Spécification des règles d'autorisation en ABAC

6.3.2.1 Règles d'accès

La fonction « regle_acces » (Listing 6.5) est une fonction de décision d'accès. La décision d'accès est calculée en fonction des valeurs d'attributs. Dans cette fonction, le client précise que par défaut l'accès est interdit et définit des règles d'autorisation.
La fonction « regle_acces » prend quatre paramètres en entrée : un sujet « sa », un objet « oa », un contexte « ca » et une action « a ». Les valeurs d'attributs du sujet, de l'objet, du contexte et de l'action passés en paramètres, sont mises en correspondance avec les règles spécifiées par la fonction « regle_acces » afin de retourner une permission (permis ou interdit). Si la fonction retourne permis alors le sujet est autorisé à effectuer l'action sur l'objet dans le contexte spécifié sinon l'action en question est interdite.

Dans cet exemple, on a spécifié trois règles. « regle_1 » : les sujets ayant un rôle « docteur » et un niveau d'expertise « expert » sont autorisés à « lire » les objets de type « dossierChir » avec un objectif de « consultation » et dont la sensibilité est « faible », seulement quand « DateJour » est égale à « am ».
La deuxième règle « regle_2 » permet aux sujets ayant le rôle « developpeur » et une expertise différente de « debutant » de modifier l'application `appAnalyse`, avec un objectif de mise à jour « appMaj », quand la date du jour est égale à « pm ».

La troisième règle « regle_3 » interdit par défaut le reste des accès, sur lesquels les deux premières règles ne s'appliquent pas.

Listing 6.5 –

```
fun regle_acces[sa:SujetMed, oa:ObjetMed, ca:ContexteMed,a:Action]: permission {
(    /*regle_1*/
  ( sa.role=docteur && sa.expertise=expert && oa.objId=dossierChir && oa.objectif=
    consultation && oa.sensibilite=faible  && ca.dateJour=am && a=lire)
  ||    /*regle_2*/
  ( sa.role=developpeur && sa.expertise!=debutant && oa.objId=appAnalyse && oa.
    objectif=appMaj && ca.dateJour=pm && a=ecrire)
 )  =>    permis
    /*regle_3*/
    else  interdit
 }
```

Pour cette fonction, on définit les signatures d'autorisation et d'interdiction dans Listing 6.6 (signatures privées : sont utilisées seulement dans le module Alloy où elles sont définies).

Listing 6.6 –

```
private  abstract sig  permission{}
private  abstract one sig permis, interdit  extends permission{}
```

6.3.2.2 Règles de partage

On définit un deuxième format de règle d'accès pour exprimer des autorisations de partage de privilèges. Un sujet partage un objet avec un sujet cible en lui autorisant d'effectuer une action donnée si les conditions du contextes sont réunies.
La fonction (Listing 6.7) prend pour paramètres en entrée : un sujet « sa », un objet « oa », un contexte « ca », un sujet cible « saCible » et une action cible « acCible ». Elle décrit les règles qui autorisent ou interdisent le partage de privilège en fonction des valeurs d'attributs des éléments passés en paramètre.

Listing 6.7 –

```
fun regle_partage[sa:SujetMed, oa:ObjetMed, ca:ContexteMed, saCible:SujetMed,
    acCible:Action]: permission {
  ( sa.role=docteur && oa.objId=dossierMedical && oa.objectif=consultation &&
    saCible.role= infirmier  && acCible=lire && ca.dateJour=pm && ca.cNiveau=saas )
    =>permis
  else  interdit
  }
```

Cette fonction contient une seule règle qui autorise un partage en lecture seule d'un objet identifié par dossierMedical avec un objectif de consultation. Ce partage est autorisé s'il est fait par un sujet ayant le rôle docteur avec un sujet cible ayant le rôle infirmier. Cette règle de partage est équivalente dans le ACSP à la règle ACSP-RSM suivante :

```
caPartage([role:docteur],[objId:dossierMedical,objectif:consultation],
          [role:infirmier],lire,[niveauNuage:SaaS,dateJour:pm],permis).
```

6.3.3 Spécification des relations entre les objets et entre les sujets

Le concept de relation d'Alloy nous a permis de simplifier l'expression des règles de contrôle d'accès et de sécurité. Dans cette partie, nous introduisons des relations entre les sujets et entre les objets. Cela nous permet d'exprimer des règles de façon simple. En plus, cela aide à spécifier et à vérifier des propriétés de sécurité complexes comme la séparation de privilèges.
On définit des relations entre les objets. Ces relations peuvent être des collaborations ou des concurrences entre les ressources (par exemple, les applications). De la même façon, on définit des relations de concurrence et de collaboration entre les sujets.
La séparation des objets et des sujets en groupes de collaboration et de concurrence permet de définir des propriétés de sécurité de façon plus souple. En effet, une seule spécification suffit pour autoriser un sujet à accéder aux objets en collaboration. Par contre, il faut plusieurs règles pour autoriser au même sujet l'accès à tous les objets, si on ne considère pas la relation de collaboration entre les objets.
L'objectif est aussi de séparer les objets sensibles des autres et de séparer les privilèges en fonction des relations entre les objets et des relations entre les sujets. C'est plus simple de ne pas autoriser à un utilisateur l'accès aux objets concurrents que lui interdire l'accès à ces objets en spécifiant une règle pour chaque objet. L'ensemble « relation » (défini dans Listing 6.8) rassemble les couples des objets en collaboration « ObjetCollaboration » et en concurrence « ObjetConcurrence » ainsi que les couples des sujets en collaboration « SujetCollaboration » et en concurrence « SujetConcurrence ».

L'ensemble « relation.ObjetCollaboration » (resp. « relation.ObjetConcurrence ») représente les objets définis en collaboration (resp. en concurrence). L'ensemble « relation.SujetCollaboration » (resp. « relation.SujetConcurrence ») représente les sujets définis en collaboration (resp. en concurrence).

Listing 6.8 –

```
one sig relation {
  ObjetCollaboration: set ObjetMed->ObjetMed,
  ObjetConcurrence: set ObjetMed->ObjetMed,
  SujetCollaboration: set SujetMed->SujetMed,
  SujetConcurrence: set SujetMed->SujetMed
  }
```

6.3.4 Spécification des propriétés de sécurité requises

Le client peut définir des propriétés de sécurité pour son système sans avoir besoin de spécifier des règles de contrôle d'accès. Pour cela, on définit des propriétés de sécurité ; par exemple, la confidentialité ou l'intégrité d'un objet par rapport à un sujet. On définit aussi un ensemble d'objets sur lesquels on applique la séparation des privilèges. Ces propriétés sont groupées dans un ensemble qu'on appelle politique de sécurité « SPpolitique » présentée dans Listing 6.9.
L'ensemble « SPpolitique.confidentialite » (resp. « SPpolitique.integrite ») contient les propriétés de confidentialité (resp. d'intégrité) à garantir. Si la relation $o \rightarrow s$ appartient à l'ensemble « SPpolitique.confidentialite », cela veut dire que le sujet s n'a pas le droit de lire l'objet o. Le sujet s n'a pas le droit de modifier l'objet o si la relation $o \rightarrow s$ appartient à l'ensemble « SPpolitique.integrite ».
L'objectif de ces propriétés est de compléter la politique de contrôle d'accès. Cela permet de définir des propriétés de façon plus souple entre les objets et les sujets grâce aux relations de collaboration et de concurrence. Il est aussi possible de définir la SPpolitique comme plus prioritaire que la politique de contrôle d'accès. Autrement dit, quand on a une demande d'accès, on consulte d'abord les règles de la SPpolitique. Si on trouve une règle de confidentialité (resp. d'intégrité) qui correspond aux sujet et objet de la requête d'accès, c'est-à-dire, l'accès en lecture (resp. écriture) est interdit. Si on ne trouve pas, on calcule la décision d'accès à partir des fonctions « regle_acces » et « regle_partage ».

Listing 6.9 –

```
one sig SPpolitique {
   confidentialite : set ObjetMed->SujetMed,
   integrite : set ObjetMed->SujetMed,
  separationPrivilege : set ObjetMed
  }
```

Par exemple tous les objets o de type « dossierChir » doivent être confidentiels pour les sujets s ayant « radiologue » comme rôle. Dans ce cas la relation $o \rightarrow s$ est ajoutée dans l'ensemble « SPpolitique.confidentialite ». Cette propriété est spécifiée dans Listing 6.10.

Listing 6.10 –

```
fact {
  all o: ObjetMed| all s:SujetMed|
  o.objId=dossierChir && s.role=radiologue =>
  o->s in SPpolitique. confidentialite
  }
```

Dans Listing 6.11, on ajoute les objets de types `appAnalyse` et `appRadio` dans l'ensemble « SPpolitique.separationPrivilege » des objets sur lesquels les privilèges vont être séparés. Aucun sujet dans le système ne peut effectuer plus d'une opération sur chacun de ces objets. Si un sujet a le droit d'exécuter l'application `appAnalyse`, la propriété de séparation de privilège lui interdit d'écrire la même application.

Listing 6.11 –

```
fact {
  all o: ObjetMed|
  o.objId=appAnalyse || o.objId=appRadio =>
  o in SPpolitique.separationPrivilege
  }
```

6.3.5 Analyse formelle des politiques ACSP

Dans cette sous-section nous commençons par détecter les conflits qu'un client peut introduire quand il spécifie sa politique de contrôle d'accès et de sécurité.

6.3.5.1 Confidentialité des objets

Le client ne peut pas demander en même temps et dans les mêmes conditions la confidentialité d'un objet par rapport à un sujet et autoriser la lecture du même objet par le même sujet. Cela peut être spécifié dans l'assertion 6.12 « conflit_confidentialite ». Il ne doit pas y avoir deux règles telles que : une première règle de confidentialité interdit à un sujet donné la lecture d'un objet « o $\rightarrow$ s $\in$ SPpolitique.confidentialite » et une deuxième règle de contrôle d'accès autorise la même opération pour un contexte quelconque « regle_acces[s,o,c,lire]=permis ».

Listing 6.12 –

```
assert conflit_confidentialite {
  all o: ObjetMed|all s:SujetMed|all c: ContexteMed|
  !(o->s in SPpolitique.confidentialite &&
    regle_acces[s,o,c, lire ]=permis)
  }
```

6.3.5.2 Intégrité des objets

Le client ne peut pas demander l'intégrité d'une ressource par rapport à un sujet donné et en même temps autoriser ce sujet à modifier (écrire) le même objet. L'assertion 6.13 « conflit_integrite » spécifie que dans le système du client, on ne permet pas l'existence des deux spécifications suivantes : la première qui exige l'intégrité d'un objet par rapport à un sujet « o $\rightarrow$ s $\in$ SPpolitique.integrite », c'est-à-dire, ce sujet n'a pas le droit de modifier cet objet et une deuxième règle permet la même opération pour les mêmes sujet et objet « regle_acces[s,o,c,ecrire]=permis ».

Listing 6.13 –

```
assert conflit_integrite {
  all o: ObjetMed|all s:SujetMed|all c: ContexteMed|
  !(o->s in SPpolitique.integrite &&
    regle_acces[s,o,c, ecrire ]=permis)
  }
```

6.3.5.3 Séparation des privilèges

Séparer les privilèges revient à séparer les droits de lecture, d'écriture et d'exécution d'un objet sur différents types d'attributs des sujets. Par exemple, le sujet qui développe `appAnalyse` doit être différent de celui qui la teste. Ces propriétés sont inspirées des travaux de [148] qui proposent un ensemble de propriétés de séparation de privilèges qui sont formalisées dans [65]. L'objectif de la séparation de privilèges

est de distribuer les droits d'accès (privilèges) à un objet sensible sur différents sujets. Cela réduit le risque de violation de confidentialité et d'intégrité. Ainsi, une violation apparue dans une partie du système ne se propage pas au reste du système, parce que cela ne peut pas dépasser les privilèges autorisés sur l'objet concerné. Le client définit lui même les objets qui contiennent ou traitent des données sensibles et exprime des propriétés de séparation de privilèges sur ces objets. Les privilèges sont distribués sur différents sujets en considérant les relations (collaboration ou concurrence) entre les sujets et les objets définies dans la sous-section 6.3.3. Nous spécifions dans la suite quelques propriétés de séparation de privilèges en utilisant Alloy. Le client peut définir de telles propriétés sur des ressources critiques ou sur des parties sensibles de son système.

1. **Séparation simple des privilèges :** Il n'existe aucun utilisateur qui peut effectuer plus d'une opération sur un objet. Motivation : un utilisateur ne doit pas pouvoir créer un exécutable, et l'exécuter, pour faire une attaque. Ceci empêche les attaques par concurrence d'accès par exemple. Cette propriété est exprimée dans l'assertion présentée dans Listing 6.14.

Listing 6.14 –

```
assert privilege_separation_1{
  all o: ObjetMed| all disj a1,a2:Action| no s:SujetMed|
  regle_acces[s,o,ContexteMed,a1]=permis &&
  regle_acces[s,o,ContexteMed,a2]=permis }
```

2. **Séparation des privilèges entre sujets collaboratifs :** Il n'existe pas deux utilisateurs en collaboration (appartiennent au même projet par exemple) qui peuvent effectuer toutes les opérations du système (trois opérations : lecture, écriture et exécution) sur un objet. Motivation : deux utilisateurs en collaboration ne doivent pas pouvoir collaborer pour effectuer différentes opérations qui mèneraient vers une attaque. Par exemple, si un utilisateur peut créer un exécutable, il ne faut pas qu'un second utilisateur en collaboration avec lui puisse l'exécuter, et vice versa. Cette propriété est exprimée dans l'assertion présentée dans Listing 6.15.

Listing 6.15 –

```
assert privilege_separation_2{
  all o: ObjetMed| all disj si, sj :SujetMed| all disj a1,a2,a3:Action|
  !(  si->sj in relation.SujetCollaboration &&
      regle_acces[si,o,ContexteMed,a1]=permis &&
      regle_acces[si,o,ContexteMed,a2]=permis &&
      regle_acces[sj,o,ContexteMed,a3]=permis
    ) }
```

3. **Séparation des privilèges entre sujets concurrents :** Deux utilisateurs en concurrence ne doivent pas avoir des privilèges sur le même objet (ou ne doivent pas avoir accès au même objet). Motivation : un utilisateur peut effectuer une opération (écrire du contenu) qui sera annulée (supprimer le contenu) par l'autre utilisateur. Cette propriété est exprimée dans l'assertion présentée dans Listing 6.16.

Listing 6.16 –

```
assert privilege_separation_3{
  all o: ObjetMed| all disj si,sj:SujetMed| disj a1,a2:Action|
  !(  si->sj in relation.SujetConcurrence &&
    regle_acces[si,o,ContexteMed,a1]=permis &&
    regle_acces[sj,o,ContexteMed,a2]=permis
  ) }
```

4. **Séparation des privilèges sur des objets conflictuels :** Un utilisateur ne doit pas pouvoir accéder à deux objets en conflit. Motivation : l'utilisateur peut accéder à un objet pour récupérer de l'information et la divulguer vers un autre objet qui est en concurrence avec le premier. Listing 6.17 présente une spécification Alloy de cette propriété.

Listing 6.17 –

```
assert privilege_separation_4{
  all disj oi, oj: ObjetMed| no s:SujetMed|
  oi->oj in relation.ObjetConcurrence &&
  regle_acces[s,oi,ContexteMed,Action]=permis &&
  regle_acces[s,oj,ContexteMed,Action]=permis }
```

5. **Séparation des privilèges sur des objets conflictuels entre sujets collaboratifs :** Deux utilisateurs en collaboration ne doivent pas avoir des privilèges sur deux objets en conflit. Motivation : les utilisateurs qui font partie d'une collaboration (c'est-à-dire, un projet ou un groupe) partagent potentiellement un espace RAM ou un disque de stockage. Un utilisateur « si » peut récupérer de l'information d'un objet « oi », l'information est sauvegardée dans le moyen de stockage partagé « ok ». L'autre utilisateur « sj » peut dans ce cas accéder au moyen de stockage partagé « ok », récupérer l'information et la divulguer au deuxième objet « oj » qui est en conflit avec le premier. Listing 6.18 présente une spécification Alloy de cette propriété.

Listing 6.18 –

```
assert privilege_separation_5{
  all disj oi,oj, ok: ObjetMed| all disj si,sj:SujetMed|
  !(  si->sj in relation.SujetCollaboration &&
    oi->oj in relation.ObjetConcurrence &&
    regle_acces[si,oi,ContexteMed,Action]=permis &&
    regle_acces[si,ok,ContexteMed,Action]=permis &&
    regle_acces[sj,ok,ContexteMed,Action]=permis &&
    regle_acces[sj,oj,ContexteMed,Action]=permis
  ) }
```

6. **Séparation des privilèges conflictuels :** On considère deux privilèges en conflits, deux opérations qu'on ne permet pas d'effectuer dans le même contexte ou par le même sujet, par exemple écrire (mettre à jour) une application et exécuter la même application. Il n'est pas autorisé à effectuer deux actions en conflit dans les mêmes conditions d'environnement (contexte). L'assertion présentée dans Listing 6.19 exprime cette propriété.

Listing 6.19 –

```
assert privilege_separation_6{
  all si,sj:SujetMed|all c:ContexteMed|no o:ObjetMed|
  o in SPpolitique.separationPrivilege &&
  regle_acces[si,o,c,executer]=permis &&
  regle_acces[sj,o,c,ecrire]=permis
  }
```

Exemple : Supposons que le client a défini que :

- Les objets ayant un ObjId `dossierChir` et une sensibilité `critique` sont en concurrence entre eux.
- Les utilisateurs ayant un rôle de `docteur` et un niveau d'expertise `debutant` ont le droit d'accéder en lecture et écriture à tous les objets avec un ObjId `dossierChir`.

En analysant l'assertion `privilege_separation_4` (Listing 6.17, page 120), Alloy retourne un contre-exemple dont une partie est présentée dans la figure 6.2.

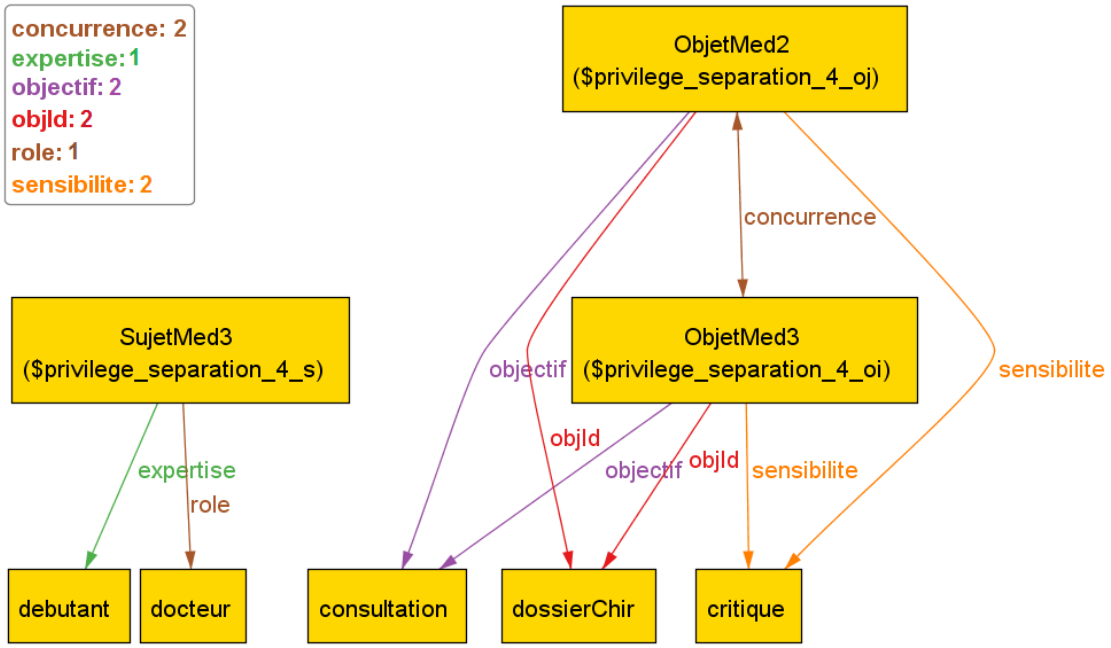

FIGURE 6.2 – *Exemple : contre-exemple, propriété de séparation de privilèges non respectée.*

Ce contre-exemple montre des éléments qui prouvent que la propriété de séparation de privilèges `privilege_separation_4` n'est pas respectée.
Le `SujetMed3` a un rôle `docteur` et un niveau d'expertise `debutant`. Selon les règles précisées par le client, ce sujet a donc accès aux objets `ObjetMed2` et `ObjetMed3` qui ont un ObjId `dossierChir`. Ces deux derniers ont aussi une sensibilité `critique` ce qui les rend concurrents entre eux.
Cela montre que l'assertion `privilege_separation_4` vérifiant qu'il n'existe aucun sujet ayant accès à deux objets concurrents, n'est pas respectée par la politique du client.

6.3.5.4 Détection des conflits et des redondances

Dans cette partie nous utilisons Alloy pour détecter les conflits dans les spécifications du client. Le client peut spécifier des propriétés contradictoires. Nous les détectons et les signalons au client. Ce dernier doit modifier ses spécifications afin d'éliminer les conflits.

Conflits de modalité : Pour tous les objets, sujets, opérations et contextes du système hospitalier, il ne doit pas exister deux règles telles que l'une autorise aux sujets d'effectuer une action sur un objet et l'autre interdit le même privilège. On utilise l'assertion Alloy présentée dans Listing 6.20 pour détecter de tels conflits.

Listing 6.20 –

```
assert conflit_modalite{
  all o: ObjetMed| all s:SujetMed| all a:Action | all c: ContexteMed |
  !( regle_acces[s,o,c,a]=permis &&
    regle_acces[s,o,c,a]= interdit )
  }
```

Problèmes de redondance : Il ne doit pas exister de règles qui couvrent d'autres qui ne sont donc jamais appliquées. Dans l'assertion présentée dans Listing 6.21, on vérifie que pour tous les ObjetMed, SujetMed et ContexteMed, il n'existe pas deux règles telles que : une première règle applique la confidentialité d'un objet pour un sujet donné, et la deuxième règle interdit au même sujet de lire cet objet dans un contexte précis. La première règle couvre la deuxième qui n'est dans ce cas jamais appliquée.

Listing 6.21 –

```
assert conflit_redondance{
  all o: ObjetMed| all s:SujetMed|all c: ContexteMed|
  !( o->s in SPpolitique. confidentialite &&
    regle_acces[s,o,c, lire ]= interdit )
  }
```

Conflits de contexte : Il n'existe pas de règles qui expriment les mêmes privilèges avec des domaines de contexte contradictoires et non disjoints. Par exemple, l'assertion présentée dans Listing 6.22, assure que l'on n'autorise pas le même privilège sur deux niveaux différents de nuage informatique.

Listing 6.22 –

```
assert conflit_contexte{
  all o: ObjetMed| all s:SujetMed| all a:Action | no disj ci, cj: ContexteMed |
  regle_acces[s,o,ci,a]=permis &&
  regle_acces[s,o,cj,a]=permis &&
  ci.cNiveau!=cj.cNiveau
  }
```

Conflits issus du partage : Un utilisateur ne peut pas partager un privilège qu'il ne détient pas.
Ce type de conflit peut être détecté en vérifiant l'assertion présentée dans Listing 6.23.

Listing 6.23 –

```
assert conflit_partage{
  all o: ObjetMed| all s,sCible:SujetMed| all acCible:Action | all c: ContexteMed |
  (regle_acces[s,o,c,aCible]= interdit ) =>
  (regle_partage[s,o,c,sCible,aCible]= interdit )
  }
```

Exemple : Selon la règle de partage 6.7 (page 115) précisée par le client, les sujets ayant le rôle `docteur` ont le droit de partager en mode lecture les objets ayant un ObjId `dossierMedical` et un objectif de `consultation` avec les sujets ayant le rôle `infirmier` dans un certain contexte. Cette règle leur donne le droit de partager des dossiers médicaux avec une sensibilité `critique` parce que cette règle de partage ne fait pas de restriction sur la sensibilité des objets à partager.
Cependant, les sujets ayant le rôle `docteur` n'ont même pas le droit d'accéder en mode lecture aux dossiers médicaux ayant une sensibilité `critique` (selon les règles d'accès 6.5, page 114).
Ce qui va être mis en évidence dans une partie d'un contre-exemple (Figure 6.3) retourné par l'analyseur Alloy, en analysant `conflit_partage` 6.23.

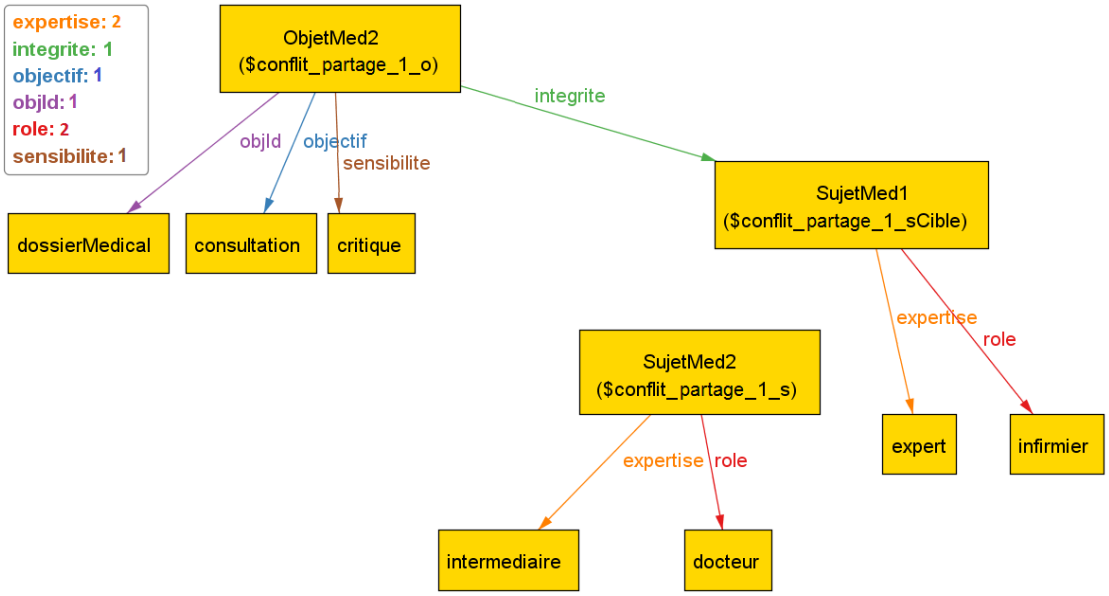

FIGURE 6.3 – *Exemple : contre-exemple, conflit de règle de partage et règle d'accès.*

6.4 CONCLUSION

Ce chapitre est divisé en deux parties. Dans la première partie, nous avons présenté un modèle de spécification des besoins de contrôle d'accès et de sécurité. Ce modèle

est utilisé pour décrire un système qu'un client souhaite déployer dans le nuage informatique. Il permet aussi de spécifier dans un langage semi-formel les besoins de contrôle d'accès et de sécurité qui doivent être assurés une fois le système migré vers le nuage informatique.
Dans la deuxième partie, nous illustrons la méthode de spécification et analyse formelles des politiques de sécurité et de contrôle d'accès que le client a spécifiées dans la première partie. Nous considérons un système hospitalier qui nécessite de fortes exigences de sécurité. Nous présentons un modèle de spécification basé sur les attributs. Notre modèle est spécifié en utilisant le langage formel Alloy. Nous décrivons les systèmes à déployer (hospitalier dans ce cas) et spécifions les règles de contrôle d'accès et les propriétés de sécurité que le client exige. Nous analysons en utilisant l'analyseur Alloy, que les spécifications du client sont consistantes, c'est-à-dire, il n'existe pas de conflits dans les spécifications du client. Nous vérifions ainsi que les propriétés de sécurité exigées par le client sont satisfiables par rapport à la description de son système. Cela donne au client une assurance que si le déploiement se fait en bonnes conditions, la politique de sécurité et de contrôle d'accès qui sera appliquée sur le système dans le nuage informatique satisfait toutes les exigences spécifiées par le client et elle est exempte de conflits et de contradictions.

Le travail de ce chapitre peut être amélioré et complété en considérant la validation de la politique de contrôle d'accès par rapport aux contraintes de sécurité exprimées par le client dans le chapitre 4. Par exemple : ne pas autoriser un utilisateur à accéder aux objets situés dans deux MV en concurrence. Ces vérifications peuvent être basées sur la vérification de l'ensemble des flux d'information entre les utilisateurs et les objets du système client. Pour cela, nous pouvons nous inspirer du PIGA [35] qui est un mécanisme permettant de contrôler les flux d'information entre les objets et les sujets.

Conclusion et perspectives 7

Sommaire

7.1 Conclusion 127
7.2 Perspectives 128

7.1 Conclusion

La contribution majeure de cette thèse est d'intégrer les besoins des clients en termes de sécurité dans un processus de courtage des services de l'informatique en nuage.

L'état de l'art dressé, montre la nécessité de prendre en compte les critères personnalisés de sécurité des clients. Les fournisseurs proposent des offres préalablement configurées, par exemple : une MV avec un IDS pré-installé. Cependant, cela reste insuffisant car chaque client, qu'il soit particulier, une grande ou moyenne entreprise ou une organisation politique ou de défense, a ses besoins spécifiques à son activité, son système d'information, ses données, ses services et ses propres utilisateurs. Il est donc indispensable d'offrir la possibilité aux clients d'exprimer leurs propres contraintes et besoins en termes de sécurité avant même de se procurer les services de l'informatique en nuage. De plus, il faut que le client soit capable de comparer les différentes offres des fournisseurs et d'en sélectionner celles qui répondent le mieux à ses besoins.

La solution qui nous semble la plus adaptée est de considérer les critères de sécurité exprimés par le client dans le processus de courtage. Cela permet au client de comparer les offres des fournisseurs et d'en sélectionner qui assurent ses exigences de sécurité. De plus, toutes les ressources demandées par le client ainsi que les mécanismes de sécurité qui répondent à ses besoins doivent être installés et configurés avant d'être mis en service et utilisés par ce client. Nous proposons un courtier basé sur le langage et l'outil Alloy. Cet outil de vérification formelle nous permet d'analyser les systèmes de l'informatique en nuage et les besoins du client. Il nous permet de détecter les erreurs de spécification assez tôt dans le processus de courtage afin d'éviter des vulnérabilités après le déploiement des ressources. De plus, c'est un moyen d'assurer que les propriétés de sécurité sont valables par rapport au système du client et par rapport aux offres des fournisseurs.

Le client décrit son système à déployer avec des critères de ressources (CPU, RAM, ...) ainsi que des propriétés de sécurité qu'il souhaite appliquer. Nous simplifions l'expression des propriétés de sécurité en définissant des relations de collaboration, de concurrence et d'isolation entre les composants du système client. Le client définit les échanges d'information possibles entre les composants. Il définit des contraintes de placement en fonction des relations de collaboration/concurrence/isolation. Par exemple : les MV concurrentes doivent être placées dans des grappes différentes. Les autorisations d'échange d'information peuvent aussi être exprimées en fonction de ces relations. Par exemple : Les MV en collaboration doivent échanger des informations. Cela veut dire que les flux d'information entre les MV en collaboration sont possibles et doivent être permis au niveau des gardiens (par-feux). Toutes ces relations, contraintes, et propriétés sont faciles à spécifier en utilisant Alloy.

Nous vérifions ensuite que les systèmes client ne contiennent pas de conflit. Nous générons un placement des ressources du client chez les fournisseurs qui assurent les besoins techniques. Nous vérifions aussi que le placement généré assure les contraintes du placement et les propriétés de sécurité que le client a exigées. Cela se fait en analysant les flux d'information directs et transitifs entre les ressources. Alloy permet de vérifier les spécifications du client et de générer des contre-exemples dans le cas d'insatisfiabilité d'une spécification. Si toutes les contraintes sont respectées, le placement peut être adopté par le client.

Nous avons ensuite proposé la vérification des politiques de contrôle d'accès avant le déploiement des ressources du client chez les fournisseurs. Le client décrit les types d'utilisateurs qui vont accéder à ses ressources et services. Nous proposons aussi un moyen pour simplifier l'expression des autorisations d'accès et des propriétés de sécurité en utilisant des relations entre les ressources et entre les utilisateurs. Cela permet de spécifier facilement des propriétés comme la séparation de privilèges. Par exemple : Les utilisateurs définis en concurrence n'ont pas le droit d'accéder aux objets définis en collaboration.

Notre démarche permet de simplifier la spécification des contraintes de placement des ressources, des propriétés de sécurité et des autorisations d'accès. Elle permet aussi de générer un placement qui satisfait ces contraintes et de vérifier que les exigences du client sont respectées et peuvent être appliquées. Ainsi, on évite de déployer un système contenant des vulnérabilités et l'exposer aux attaques. De plus, nous offrons au client une assurance de respecter les contrats de sécurité de ses ressources. Nous pensons que notre démarche peut être approfondie et intégrée par les fournisseurs ou les courtiers du nuage informatique.

7.2 Perspectives

Dans cette section, nous discutons des limitations de notre approche et des perspectives à réaliser afin d'améliorer et d'étendre notre travail.

Améliorations du courtier Plusieurs critères de sécurité peuvent être considérés dans le processus de courtage, par exemple les mécanismes de chiffrement, d'authentification, etc. Pour cela, il est possible de considérer les contrôles de sécurité du registre STAR [45]. La vérification formelle de ce type de critères de sécurité n'est pas évidente. Dans ce cas, il est possible d'utiliser de l'audit par un tiers de confiance ou utiliser des mécanismes de certifications.

D'un autre coté, nous avons considéré dans cette thèse la configuration physique des offres des fournisseurs. Il serait possible de prendre en compte des mécanismes de sécurité au niveau virtuel. Par exemple, il est possible d'assurer l'isolation entre MV en utilisant des pare-feux virtuels.

Le courtier retourne au client des contre-exemples dans le cas d'incohérence de sa description ou de non-satisfaction des propriétés exigées. La limitation est que le courtier renvoie le contre-exemple retourné par Alloy. Ce dernier peut être compliqué à exploiter. Pour améliorer ce point, il serait intéressant de fournir au client des contre-exemples bien expliqués et illustrés. En plus du contre-exemple qui illustre l'invalidité de spécification ou l'incohérence du modèle, il est possible de mettre en évidence les contraintes, les spécifications et les propriétés qui posent problèmes.

Solveur de contraintes Le mécanisme de placement de notre processus de courtage est basé sur la mise en correspondances des besoins du client avec les offres des fournisseurs. La mise en correspondance concerne les critères fonctionnels et les critères non-fonctionnels. Nous avons choisi d'utiliser Alloy pour la mise en correspondance afin de détecter les erreurs de spécification avant le déploiement. Ce choix est adapté aux critères non-fonctionnels, car il permet de vérifier que les spécifications sont satisfiables. Cependant, Alloy est limité quand il s'agit des critères numériques car il n'est pas destiné au calcul numérique. Une solution qui pourrait améliorer ce point est de faire la mise en correspondance des critères fonctionnels nécessitant les calculs numériques en utilisant un solveur de contraintes. Ainsi, on aurait une première configuration de placement que le courtier analyse en utilisant Alloy. Il vérifie alors que les critères non-fonctionnels sont assurés. Dans le cas contraire, on réitère la mise en correspondance jusqu'à trouver une solution ou épuiser les solutions possibles. Cela sous-entend qu'il est aussi nécessaire de développer la traduction des résultats et la communication entre le solveur de contraintes et Alloy.

Déploiement et établissement des contrats Le travail effectué dans cette thèse pourrait être poursuivi en mettant en œuvre l'étape de déploiement. Après avoir trouver un placement des ressources client qui satisfait tous ses besoins, il faut déployer les ressources comme les MV, chez les fournisseurs sélectionnés. Cela peut être réalisé en s'inspirant du projet CompatibleOne [174]. Il est nécessaire de faire les configurations indispensables pour que le système du client soit fonctionnel (communications réseaux).

Les propriétés de sécurité exigées par le client ainsi que sa politique de contrôle d'accès doivent être appliquées une fois ses ressources déployées dans le nuage informatique. Cela peut être effectuée en configurant des mécanismes de sécurité adaptés aux besoins. Dans [34] plusieurs mécanismes de sécurité peuvent être configurés afin de répondre à des propriétés de sécurité. Par exemple, les propriétés de contrôle d'accès peuvent être assurées en utilisant IP Tables.

Il faut d'abord analyser la possibilité d'établir des contrats SecLA (*Security SLA*) en utilisant des langages existants, tout en considérant les contraintes de sécurité. Le langage doit être supporté par des mécanismes de gestions des SLA. Cela facilite le suivi et l'audit tout au long du cycle de vie d'un système dans l'informatique en nuage. En plus des tarifs, de la qualité de service et de sécurité, ces contrats doivent contenir toutes les actions à effectuer en cas de rupture du contrat, de violation des propriétés de sécurité ou d'autres contraintes.

Évolution dynamique des besoins Les besoins du clients pourraient évoluer de façon dynamique. Il peut ajouter ou supprimer des ressources ou des utilisateurs. Il peut modifier sa politique de sécurité. La migration des ressources entre des fournisseurs devrait être considérée. Tout cela devrait être possible dans le processus de courtage. Donc, il serait intéressant d'ajouter des mécanismes de re-placement (mise en correspondance). Cela permettrait de trouver un nouveau placement qui répond aux nouveaux besoins du client sans interrompre ses services, sans perdre des données et tout en préservant la sécurité de ses services et données. De telles solutions ont été proposées dans le projet mOSAIC [163], mais ne concernent que la re-négociation des SLA en cas d'évolution de la consommation des ressources. L'évolution et le changement dynamique des besoins non-fonctionnels et de sécurité ne sont pas encore traités. Cela est aussi valable pour l'évolution dynamique des architectures et des offres des fournisseurs de l'informatique en nuage.

Bibliographie

[1] NuSMV: a new symbolic model checker. http://nusmv.fbk.eu/.

[2] Available Role Based Access Control Permissions for XenServer, 2012. http://support.citrix.com/article/CTX126441.

[3] Managing projects and users, 2014. http://docs.openstack.org/openstack-ops/content/projects_users.html.

[4] Amazon Elastic Compute Cloud (Amazon EC2), 2015. https://aws.amazon.com/fr/ec2/.

[5] Amazon Simple Storage Service (Amazon S3), 2015. https://aws.amazon.com/fr/s3/.

[6] Amazon SimpleDB, 2015. https://aws.amazon.com/fr/simpledb/.

[7] Cloud Foundry documentation, 2015. http://docs.cloudfoundry.org/.

[8] Google App Engine: Platform as a service, 2015. https://cloud.google.com/appengine/.

[9] Google Apps for work, 2015. https://www.google.fr/intx/fr/work/apps/business/.

[10] Microsoft Azure: Cloud computing platform & services, 2015. https://azure.microsoft.com/.

[11] Office 365, 2015. http://office.microsoft.com/.

[12] OpenNebula : The cloud data center management solution., 2015. http://opennebula.org/.

[13] Openshift: The open hybrid cloud application platform by red hat, 2015. https://www.openshift.com/.

[14] OpenStack: The open source cloud computing software, 2015. http://www.openstack.org/.

[15] E. K. Achampong and C. Dzidonu. Managing access to electronic health records in a cloud computing environment. *Journal of Information Sciences and Computing Technologies*, 3(2):191–195, 2015.

[16] M. Ali, S. U. Khan, and A. V. Vasilakos. Security in cloud computing: Opportunities and challenges. *Information Sciences*, 305:357–383, 2015.

[17] M. Almorsy, J. Grundy, and A. S. Ibrahim. Collaboration-based cloud computing security management framework. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 364–371. IEEE, 2011.

[18] S. Alshehri. *Toward Effective Access Control Using Attributes and Pseudoroles*. PhD thesis, Rochester Institute of Technology, 2014.

[19] J. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Company, Fort Washington, Pennsylvania, April 1980.

[20] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification (ws-agreement). In *Open Grid Forum*, volume 128, page 216, 2007.

[21] J. Arshad, P. Townend, and J. Xu. An abstract model for integrated intrusion detection and severity analysis for clouds. *Cloud Computing Advancements in Design, Implementation, and Technologies*, 2012.

[22] R. Ausanka-Crues. Methods for access control: advances and limitations. *Harvey Mudd College*, 301, 2001.

[23] E. Badidi. A Framework for Software-as-a-Service Selection and Provisioning. *International Journal Of Computer Networks & Communications*, 2013.

[24] K. Bernsmed, M. G. Jaatun, P. H. Meland, and A. Undheim. Security SLAs for federated cloud services. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 202–209. IEEE, 2011.

[25] E. Bertino, B. Catania, M. L. Damiani, and P. Perlasca. GEO-RBAC: a spatially aware RBAC. In *Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 29–37. ACM, 2005.

[26] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 321–334. IEEE, 2007.

[27] S. Biedermann and S. Katzenbeisser. Detecting computer worms in the cloud. In *Open Problems in Network Security*, pages 43–54. Springer, 2012.

[28] K. Z. Bijon, R. Krishman, and R. Sandhu. Constraints specification in attribute based access control. *Science*, 2(3):131–144, 2013.

[29] K. Z. Bijon, R. Krishnan, and R. Sandhu. A framework for risk-aware role based access control. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 462–469. IEEE, 2013.

[30] K. Z. Bijon, R. Krishnan, and R. Sandhu. Towards an attribute based constraints specification language. In *Social Computing (SocialCom), 2013 International Conference on*, pages 108–113. IEEE, 2013.

[31] K. Z. Bijon, R. Krishnan, and R. Sandhu. A formal model for isolation management in cloud infrastructure-as-a-service. In *Network and System Security*, pages 41–53. Springer, 2014.

[32] S. Bleikertz and T. Groß. A virtualization assurance language for isolation and deployment. In *Policies for Distributed Systems and Networks (POLICY), 2011 IEEE International Symposium on*, pages 33–40. IEEE, 2011.

[33] S. Bleikertz, T. Groß, and S. Mödersheim. Automated verification of virtualized infrastructures. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 47–58. ACM, 2011.

[34] A. Bousquet, J. Briffaut, and C. Toinard. An autonomous cloud management system for in-depth security. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 368–374. IEEE, 2014.

[35] J. Briffaut. *Formalization and guaranty of system security properties: application to the detection of intrusions*. PhD thesis, Thèse de doctorat, Université d'Orléans. SDS, 2007.

[36] J. Brunel, D. Chemouil, L. Rioux, M. Bakkali, and F. Vallée. A viewpoint-based approach for formal safety & security assessment of system architectures. In *11th Workshop on Model-Driven Engineering, Verification and Validation*, volume 1235, pages 39–48, 2014.

[37] J. M. A. Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray. Toward a multi-tenancy authorization system for cloud services. *IEEE Security & Privacy*, (6):48–55, 2010.

[38] R. G. Cascella. Contrail: Bringing trust in clouds. 2011.

[39] V. Casola, R. Preziosi, M. Rak, and L. Troiano. A reference model for security level evaluation: Policy and fuzzy techniques. 2005.

[40] A. Cavoukian, M. Chibba, G. Williamson, and A. Ferguson. The importance of ABAC: Attribute-based access control to big data: Privacy and context. 2015.

[41] S. Chakraborty and I. Ray. TrustBAC: integrating trust relationships into the RBAC model for access control in open systems. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 49–58. ACM, 2006.

[42] S. Chari, L. Koved, M. E. Zurko, and M. Westford. Using recommenders for Discretionary Access Control. In *Proceedings of the Web*, volume 2, 2009.

[43] L. Chen and J. Crampton. Risk-aware Role-Based Access Control. In *Security and Trust Management*, pages 140–156. Springer, 2012.

[44] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Computer Aided Verification*, pages 359–364. Springer, 2002.

[45] Cloud Security Alliance. CSA Security, Trust & Assurance Registry (STAR). `https://cloudsecurityalliance.org/star/`, accessed on January 2016.

[46] Cloud Security Alliance. Consensus Assessments Initiative Questionnaire, 2011. `https://cloudsecurityalliance.org/research/cai/`, accessed on March 15, 2013.

[47] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing v3.0. 2011. `https://cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf`.

[48] Cloud Security Alliance. STAR Certification, 2011. https://cloudsecurityalliance.org/star/certification/, accessed on January 2015.

[49] Cloud Security Alliance. Cloud Controls Matrix (CCM), 2014. https://cloudsecurityalliance.org/research/ccm/, accessed on January 2015.

[50] Cloud Security Alliance. Cloud Adoption Practices and Priorities Survey Report, 2015. https://cloudsecurityalliance.org/download/cloud-adoption-practices-priorities-survey-report/, accessed on April 2016.

[51] E. Coyne and T. R. Weil. ABAC and RBAC: Scalable, flexible, and auditable access management. *IT Professional*, 15(3):0014–16, 2013.

[52] A. V. Dastjerdi, K. A. Bakar, and S. G. H. Tabatabaei. Distributed intrusion detection in clouds using mobile agents. In *Advanced Engineering Computing and Applications in Sciences, 2009. ADVCOMP'09. Third International Conference on*, pages 175–180. IEEE, 2009.

[53] S. A. de Chaves, C. B. Westphall, and F. R. Lamin. SLA perspective in security management for cloud computing. In *Networking and Services (ICNS), 2010 Sixth International Conference on*, pages 212–217. IEEE, 2010.

[54] M. Dekker and G. Hogben. Survey and analysis of security parameters in cloud SLAs across the European public sector. *Technical Report TR-2011-12-19, European Network and Information Security Agency*, 2011.

[55] A. A. El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, and G. Trouessin. Or-BAC: un modele de contrôle d'acces basé sur les organisations. *Cahiers francophones de la recherche en sécurité de l'information*, 1:30–43, 2003.

[56] D. A. Fernandes, L. F. Soares, J. V. Gomes, M. M. Freire, and P. R. Inácio. Security issues in cloud environments: a survey. *International Journal of Information Security*, 13(2):113–170, 2014.

[57] D. F. Ferraiolo and D. R. Kuhn. Role-Based Access Control. In *15th National Computer Security Conference*, 1992.

[58] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for Role-Based Access Control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[59] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, et al. OPTIMIS: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66–77, 2012.

[60] M. Frappier, B. Fraikin, R. Chossart, R. Chane-Yack-Fa, and M. Ouenzar. Comparison of model checking tools for information systems. In *Formal Methods and Software Engineering*, pages 581–596. Springer, 2010.

[61] M. Frtunic, F. Jovanovic, M. Gligorijvic, L. Dordevic, S. Janicijevic, P. H. Meland, K. Bernsmed, and H. N. Castejón. CloudSurfer- a cloud broker application for security concerns. In *CLOSER*, pages 199–206, 2013.

[62] J. L. Garcia, T. Vateva-Gurova, N. Suri, M. Rak, and L. Liccardo. Negotiating and brokering cloud resources based on security level agreements. In *CLOSER*, pages 533–541, 2013.

[63] T. Garfinkel, M. Rosenblum, et al. A virtual machine introspection based architecture for intrusion detection. In *NDSS*, volume 3, pages 191–206, 2003.

[64] S. K. Garg, S. Versteeg, and R. Buyya. SMICloud: A framework for comparing and ranking cloud services. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 210–218. IEEE, 2011.

[65] V. D. Gligor, S. I. Gavrila, and D. Ferraiolo. On the formal definition of separation-of-duty policies and their composition. In *Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on*, pages 172–183. IEEE, 1998.

[66] E. Goettelmann, K. Dahman, B. Gateau, E. Dubois, and C. Godart. A security risk assessment model for business process deployment in the cloud. In *Services Computing (SCC), 2014 IEEE International Conference on*, pages 307–314. IEEE, 2014.

[67] A. Gouglidis, I. Mavridis, and V. C. Hu. Verification of secure inter-operation properties in multi-domain RBAC systems. In *Software Security and Reliability-Companion (SERE-C), 2013 IEEE 7th International Conference on*, pages 35–44. IEEE, 2013.

[68] A. Gouglidis, I. Mavridis, and V. C. Hu. Security policy verification for multi-domains in cloud systems. *International Journal of Information Security*, 13(2):97–111, 2014.

[69] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm, 2006.

[70] T. T. W. Group et al. The notorious nine: cloud computing top threats in 2013. *Cloud Security Alliance*, 2013.

[71] A. Guesmi and P. Clemente. Access control and security properties requirements specification for clouds' seclas. In *IEEE 5th International Conference on Cloud Computing Technology and Science, CloudCom 2013, Bristol, United Kingdom, December 2-5, 2013, Volume 1*, pages 723–729, 2013.

[72] A. Guesmi, P. Clemente, F. Loulergue, and P. Berthomé. Cloud resources placement based on functional and non-functional requirements. In *SECRYPT 2015 - Proceedings of the 12th International Conference on Security and Cryptography, Colmar, Alsace, France, 20-22 July, 2015.*, pages 335–342, 2015.

[73] S. M. Habib, S. Ries, M. Mühlhäuser, and P. Varikkattu. Towards a trust management system for cloud computing marketplaces: using CAIQ as a trust information source. *Security and Communication Networks*, 7(11):2185–2200, 2014.

[74] H. Hamad and M. Al-Hoby. Managing intrusion detection as a service in cloud networks. *International Journal of Computer Applications*, 41(1):35–40, 2012.

[75] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, Aug. 1976.

[76] M. Hogan, F. Liu, A. Sokol, and J. Tong. Nist cloud computing standards roadmap. *NIST Special Publication*, 35, 2011.

[77] G. Hogben and A. Pannetrat. Mutant apples: a critical examination of cloud SLA availability definitions. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 379–386. IEEE, 2013.

[78] G. J. Holzmann. *The SPIN model checker: Primer and reference manual*, volume 1003. Addison-Wesley Reading, 2004.

[79] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. Guide to attribute based access control (ABAC) definition and considerations. *NIST Special Publication*, 800:162, 2014.

[80] F. K. Hussain, O. K. Hussain, et al. Towards multi-criteria cloud service selection. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, pages 44–48. IEEE, 2011.

[81] A. S. Ibrahim, J. Hamlyn-Harris, J. Grundy, and M. Almorsy. Cloudsec: a security monitoring appliance for virtual machines in the iaas cloud model. In *Network and System Security (NSS), 2011 5th International Conference on*, pages 113–120. IEEE, 2011.

[82] International Organization for Standardization. ISO/IEC JTC 1/SC 38 - Cloud Computing and Distributed Platforms. `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_tc_browse.htm?commid=601355`.

[83] D. Jackson. Automating first-order relational logic. In *ACM SIGSOFT Software Engineering Notes*, volume 25, pages 130–139. ACM, 2000.

[84] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290, 2002.

[85] D. Jackson. *Software Abstractions: logic, language, and analysis*. MIT Press, 2012.

[86] P. Jamkhedkar, J. Szefer, D. Perez-Botero, T. Zhang, G. Triolo, and R. B. Lee. A framework for realizing security on demand in cloud computing. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 371–378. IEEE, 2013.

[87] X. Jin, R. Krishnan, and R. S. Sandhu. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. *DBSec*, 12:41–55, 2012.

[88] A. Joshi, S. T. King, G. W. Dunlap, and P. M. Chen. Detecting past and present intrusions through vulnerability-specific predicates. In *ACM SIGOPS Operating Systems Review*, volume 39, pages 91–104. ACM, 2005.

[89] E. Kamateri, N. Loutas, D. Zeginis, J. Ahtes, F. D'Andria, S. Bocconi, P. Gouvas, G. Ledakis, F. Ravagli, O. Lobunets, et al. Cloud4SOA: A semantic-interoperability PaaS solution for multi-cloud platform management and portability. In *Service-Oriented and Cloud Computing*, pages 64–78. Springer, 2013.

[90] H. Kamoda, M. Yamaoka, S. Matsuda, K. Broda, and M. Sloman. Policy conflict analysis using free variable tableaux for access control in web services environments. In *Proceedings of the Policy Management for the Web Workshop at the 14th International World Wide Web Conference (WWW)*, pages 121–126. Japan, 2005.

[91] E. Keller, J. Szefer, J. Rexford, and R. B. Lee. NoHype: virtualized cloud infrastructure without the virtualization. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 350–361. ACM, 2010.

[92] B. Keltoum, B. Mohamed, and B. Mohamed. A Novel Access Control Model for Securing Cloud API. *on Networking and Advanced Systems*, page 67, 2015.

[93] D. R. Kuhn, E. J. Coyne, and T. R. Weil. Adding attributes to Role-Based Access Control. *Computer*, (6):79–81, 2010.

[94] H. Kwon, T. Kim, S. J. Yu, and H. K. Kim. Self-similarity based lightweight intrusion detection method for cloud computing. In *Intelligent Information and Database Systems*, pages 353–362. Springer, 2011.

[95] B. W. Lampson. Dynamic protection structures. In *Proceedings of the November 18-20, 1969, fall joint computer conference*, pages 27–38. ACM, 1969.

[96] C. Langaliya and R. Aluvalu. Enhancing cloud security through access control models: A survey. *International Journal of Computer Applications*, 112(7), 2015.

[97] M. Laureano, C. Maziero, and E. Jamhour. Intrusion detection in virtual machine environments. In *Euromicro Conference, 2004. Proceedings. 30th*, pages 520–525. IEEE, 2004.

[98] J.-H. Lee, M.-W. Park, J.-H. Eom, and T.-M. Chung. Multi-level intrusion detection system and log management in cloud computing. In *Advanced Communication Technology (ICACT), 2011 13th International Conference on*, pages 552–555. IEEE, 2011.

[99] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 1–14. ACM, 2010.

[100] J. Li, B. Li, T. Wo, C. Hu, J. Huai, L. Liu, and K. Lam. Cyberguarder: A virtualization security assurance architecture for green cloud computing. *Future Generation Computer Systems*, 28(2):379–390, 2012.

[101] J. Li, G. Zhao, X. Chen, D. Xie, C. Rong, W. Li, L. Tang, and Y. Tang. Fine-grained data access control systems with user accountability in cloud computing. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 89–96. IEEE, 2010.

[102] M. Li, S. Yu, K. Ren, and W. Lou. Securing personal health records in cloud computing: Patient-centric and fine-grained data access control in multi-owner settings. In *Security and Privacy in Communication Networks*, pages 89–106. Springer, 2010.

[103] M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. *Parallel and Distributed Systems, IEEE Transactions on*, 24(1):131–143, 2013.

[104] W. Li, H. Wan, X. Ren, and S. Li. A refined RBAC model for cloud computing. In *Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on*, pages 43–48. IEEE, 2012.

[105] Y. Li, N. Cuppens-Boulahia, J.-M. Crom, F. Cuppens, V. Frey, and X. Ji. Similarity measure for security policies in service provider selection. In *Information Systems Security*, pages 227–242. Springer, 2015.

[106] L. Liccardo, M. Rak, G. Di Modica, and O. Tomarchio. Ontology-based negotiation of security requirements in cloud. In *Computational Aspects of Social Networks (CASoN), 2012 Fourth International Conference on*, pages 192–197. IEEE, 2012.

[107] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf. NIST cloud computing reference architecture. *NIST special publication*, 500:292, 2011.

[108] C.-C. Lo, C.-C. Huang, and J. Ku. A cooperative intrusion detection system framework for cloud computing networks. In *Parallel processing workshops (ICPPW), 2010 39th international conference on*, pages 280–284. IEEE, 2010.

[109] F. Lombardi and R. Di Pietro. Secure virtualization for cloud computing. *Journal of Network and Computer Applications*, 34(4):1113–1122, 2011.

[110] A. Lounis, A. Hadjidj, A. Bouabdallah, and Y. Challal. Secure and scalable cloud-based architecture for e-health wireless sensor networks. In *Computer communications and networks (ICCCN), 2012 21st international conference on*, pages 1–7. IEEE, 2012.

[111] A. Lounis, A. Hadjidj, A. Bouabdallah, and Y. Challal. Healing on the cloud: Secure cloud architecture for medical wireless sensor networks. *Future Generation Computer Systems*, 2015.

[112] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck. Web service level agreement (WSLA) language specification. *IBM Corporation*, pages 815–824, 2003.

[113] J. Luna, H. Ghani, D. Germanus, and N. Suri. A security metrics framework for the cloud. In *Security and Cryptography (SECRYPT), 2011 Proceedings of the International Conference on*, pages 245–250. IEEE, 2011.

[114] J. Luna, H. Ghani, T. Vateva, and N. Suri. Quantitative assessment of cloud security level agreements: a case study. *Proceedings of Security and Cryptography*, pages 64–73, 2012.

[115] J. Luna, N. Suri, M. Iorga, and A. Karmel. Leveraging the potential of cloud security service-level agreements through standards. *Cloud Computing, IEEE*, 2(3):32–40, 2015.

[116] J. Luna Garcia, R. Langenberg, and N. Suri. Benchmarking cloud security level agreements using quantitative policy trees. In *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*, pages 103–112. ACM, 2012.

[117] S. Malik and L. Zhang. Boolean satisfiability from theoretical hardness to practical success. *Communications of the ACM*, 52(8):76–82, 2009.

[118] S. Mankovskii. Typing for conflict detection in access control policies. *E-Technologies: Innovation in an Open World*, 26:212, 2009.

[119] C. Mazzariello, R. Bifulco, and R. Canonico. Integrating a network IDS into an open source cloud computing environment. In *Information Assurance and Security (IAS), 2010 Sixth International Conference on*, pages 265–270. IEEE, 2010.

[120] M. Meingast, T. Roosta, and S. Sastry. Security and privacy issues with health care information technology. In *Engineering in Medicine and Biology Society, 2006. EMBS'06. 28th Annual International Conference of the IEEE*, pages 5453–5458. IEEE, 2006.

[121] P. H. Meland, K. Bernsmed, M. G. Jaatun, A. Undheim, and H. N. Castejón. Expressing cloud security requirements in deontic contract languages. In *CLOSER*, pages 638–646, 2012.

[122] P. Mell and T. Grance. The NIST definition of cloud computing. *Reports on Computer Systems Technology, NIST*, 2011.

[123] C. Modi, D. Patel, B. Borisaniya, A. Patel, and M. Rajarajan. A survey on security issues and solutions at different layers of cloud computing. *The Journal of Supercomputing*, 63(2):561–592, 2013.

[124] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan. A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications*, 36(1):42–57, 2013.

[125] H. Moraes, R. V. Nunes, and D. Guedes. Dcportalsng: Efficient isolation of tenant networks in virtualized datacenters. *Proc. 13th ICN*, 2014.

[126] J. Morris. sVirt: Hardening linux virtualization with mandatory access control. In *Linux. conf. au Conference*, 2009.

[127] F. Moscato, R. Aversa, B. D. Martino, T.-F. Fortis, and V. Munteanu. An analysis of mosaic ontology for cloud resources annotation. In *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, pages 973–980. IEEE, 2011.

[128] F. Mostefaoui. *Un cadre formel pour le développement orienté aspect: modélisation et vérification des interactions dues aux aspects*. Universite de Montreal, 2008.

[129] J. Moura. Characterising access control conflicts. In *NMR 2012 - 14th International Workshop on Non-Monotonic Reasoning*, 2012.

[130] K. Munir and S. Palaniappan. Security threats/attacks present in cloud environment. *IJCSNS*, 12(12):107, 2012.

[131] Y. Ou et al. The concept of cloud computing and the main security issues in it. 2015.

[132] A. Patel, M. Taghavi, K. Bakhtiyari, and J. C. Júnior. An intrusion detection and prevention system in cloud computing: A systematic review. *Journal of Network and Computer Applications*, 36(1):25–41, 2013.

[133] D. Petcu and C. Craciun. Towards a security sla-based cloud monitoring service. In *2014 4th International Conference on Cloud Computing and Services Science (CLOSER)*, pages 598–603, 2014.

[134] D. Petcu, C. Craciun, and M. Rak. Towards a cross platform cloud API. In *1st International Conference on Cloud Computing and Services Science*, pages 166–169, 2011.

[135] D. Power, M. Slaymaker, and A. Simpson. Automatic conformance checking of role-based access control policies via Alloy. In *Engineering Secure Software and Systems*, pages 15–28. Springer, 2011.

[136] K. Punithasurya and S. Jeba Priya. Analysis of different access control mechanism in cloud. *International Journal of Applied Information Systems (IJAIS), Foundation of Computer Science FCS*, 4(2), 2012.

[137] T. Rajendran, P. Balasubramanie, and R. Cherian. An efficient WS-QoS broker based architecture for web services selection. *International Journal of Computer Applications*, 1(9):79–84, 2010.

[138] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212. ACM, 2009.

[139] C. Rong, S. T. Nguyen, and M. G. Jaatun. Beyond lightning: A survey on security challenges in cloud computing. *Computers & Electrical Engineering*, 39(1):47–54, 2013.

[140] S. K. Sah, S. Shakya, and H. Dhungana. A security management for cloud based applications and services with diameter-aaa. In *Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014 International Conference on*, pages 6–11. IEEE, 2014.

[141] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *Advances in Cryptology–EUROCRYPT 2005*, pages 457–473. Springer, 2005.

[142] R. Sailer, E. Valdez, T. Jaeger, R. Perez, L. Van Doorn, J. L. Griffin, S. Berger, R. Sailer, E. Valdez, T. Jaeger, et al. shype: Secure hypervisor approach to trusted virtualized systems. *Techn. Rep. RC23511*, 2005.

[143] R. S. Sandhu. The Typed Access Matrix Model. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 122–136, Oakland, CA, USA, May 1992. IEEE.

[144] A. Schaad. A framework for organisational control principles. In *PhD thesis, The University of York, York, England*, 2003.

[145] A. Schaad and J. D. Moffett. A lightweight approach to specification and analysis of role-based access control extensions. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 13–22. ACM, 2002.

[146] M. Scholl, K. Stine, K. Lin, and D. Steinberg. Draft security architecture design process for health information exchanges (HIEs). *Report, NIST*, 2009.

[147] S. Sharma, A. Gupta, and S. Agrawal. A survey of intrusion detection system for denial of service attack in cloud. *International Journal of Computer Applications*, 120(19), 2015.

[148] R. T. Simon and M. E. Zurko. Separation of duty in role-based environments. In *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, pages 183–194. IEEE, 1997.

[149] A. Sirisha and G. G. Kumari. API access control in cloud using the Role Based Access Control Model. In *Trendz in Information Sciences & Computing (TISC), 2010*, pages 135–137. IEEE, 2010.

[150] D. Slamanig. Dynamic accumulator based discretionary access control for outsourced storage with unlinkable access. In *Financial Cryptography and Data Security*, pages 215–222. Springer, 2012.

[151] J. M. Spivey and J. Abrial. *The Z notation*. Prentice Hall Hemel Hempstead, 1992.

[152] G. Stoneburner, A. Goguen, and A. Feringa. Risk management guide for information technology systems. *Reports on Computer Systems Technology, NIST*, 2002.

[153] S. Subashini and V. Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications*, 34(1):1–11, 2011.

[154] J. Szefer, E. Keller, R. B. Lee, and J. Rexford. Eliminating the hypervisor attack surface for a more secure cloud. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 401–412. ACM, 2011.

[155] A. Taha, R. Trapero, J. Luna, and N. Suri. AHP-Based Quantitative Approach for Assessing and Comparing Cloud Security. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*, pages 284–291. IEEE, 2014.

[156] B. Tang, Q. Li, and R. Sandhu. A multi-tenant RBAC model for collaborative cloud services. In *Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on*, pages 229–238. IEEE, 2013.

[157] B. Tang, R. Sandhu, and Q. Li. Multi-tenancy authorization models for collaborative cloud services. *Concurrency and Computation: Practice and Experience*, 2014.

[158] Z. Tang, J. Wei, A. Sallam, K. Li, and R. Li. A new RBAC based access control model for cloud computing. In *Advances in Grid and Pervasive Computing*, pages 279–288. Springer, 2012.

[159] M. Toahchoodee and I. Ray. On the formal analysis of a spatio-temporal role-based access control model. In *Data and Applications Security XXII*, pages 17–32. Springer, 2008.

[160] M. Toahchoodee and I. Ray. Using Alloy to analyse a spatio-temporal access control model supporting delegation. *IET Information Security*, 3(3):75–113, 2009.

[161] W.-T. Tsai and Q. Shao. Role-Based Access Control using reference ontology in clouds. In *Autonomous Decentralized Systems (ISADS), 2011 10th International Symposium on*, pages 121–128. IEEE, 2011.

[162] C. Vecchiola, X. Chu, and R. Buyya. Aneka: a software platform for .NET-based cloud computing. *High Speed and Large Scale Scientific Computing*, 18:267–295, 2009.

[163] S. Venticinque, R. Aversa, B. Di Martino, M. Rak, and D. Petcu. A cloud agency for SLA negotiation and management. In *Euro-Par 2010 Parallel Processing Workshops*, pages 587–594. Springer, 2010.

[164] Z. Wan, J. E. Liu, and R. H. Deng. HASBE: a hierarchical attribute-based solution for flexible and scalable access control in cloud computing. *Information Forensics and Security, IEEE Transactions on*, 7(2):743–754, 2012.

[165] G. Wang, Q. Liu, and J. Wu. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 735–737. ACM, 2010.

[166] Q. Wang and H. Jin. Data leakage mitigation for discretionary access control in collaboration clouds. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 103–112. ACM, 2011.

[167] H. A. Weber. Role-Based Access Control: the nist solution. *SANS Institute InfoSec Reading Room*, 2003.

[168] Y. Weinsberg, S. Tzur-David, D. Dolev, and T. Anker. High performance string matching algorithm for a network intrusion prevention system (nips). In *High Performance Switching and Routing, 2006 Workshop on*, pages 7–pp. IEEE, 2006.

[169] H. Wu, Y. Ding, C. Winer, and L. Yao. Network security for virtual machine in cloud computing. In *Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on*, pages 18–21. IEEE, 2010.

[170] M.-Y. Wu and T.-H. Lee. Design and implementation of cloud API access control based on OAuth. In *TENCON Spring Conference, 2013 IEEE*, pages 485–489. IEEE, 2013.

[171] Y. Xia, Y. Liu, and H. Chen. Architecture support for guest-transparent vm protection from untrusted hypervisor and physical attacks. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pages 246–257. IEEE, 2013.

[172] Z. Xiao and Y. Xiao. Security and privacy in cloud computing. *Communications Surveys & Tutorials, IEEE*, 15(2):843–859, 2013.

[173] T. Xing, D. Huang, L. Xu, C.-J. Chung, and P. Khatkar. Snortflow: A openflow-based intrusion prevention system in cloud environment. In *Research and Educational Experiment Workshop (GREE), 2013 Second GENI*, pages 89–92. IEEE, 2013.

[174] S. Yangui, I.-J. Marshall, J.-P. Laisne, and S. Tata. CompatibleOne: The open source cloud broker. *Journal of Grid Computing*, 12(1):93–109, 2014.

[175] E. Yuan and J. Tong. Attributed based access control (ABAC) for web services. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005.

[176] F. Zhang, J. Chen, H. Chen, and B. Zang. Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 203–216. ACM, 2011.

[177] R. Zhang and L. Liu. Security models and requirements for healthcare application clouds. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 268–275. IEEE, 2010.

[178] J. Zhu and W. W. Smari. Attribute based access control and security for collaboration environments. In *Aerospace and Electronics Conference, 2008. NAECON 2008. IEEE National*, pages 31–35. IEEE, 2008.

Asma GUESMI

Spécification et analyse formelles des politiques de sécurité dans un processus de courtage de l'informatique en nuage

Les offres de l'informatique en nuage augmentent de plus en plus et les clients ne sont pas capables de les comparer afin de choisir la plus adaptée à leurs besoins. De plus, les garanties de sécurité proposées par les fournisseurs restent incompréhensibles pour les clients. Cela représente un frein pour l'adoption des solutions de l'informatique en nuage.
Dans cette thèse, nous proposons un mécanisme de courtage des services de l'informatique en nuage qui prend en compte les besoins du client en termes de sécurité.
Les besoins exprimés par le client sont de deux natures. Les besoins fonctionnels représentent les ressources et leurs performances. Les besoins non-fonctionnels représentent les propriétés de sécurité et les contraintes de placement des ressources dans le nuage informatique. Nous utilisons le langage Alloy pour décrire les offres et les besoins. Nous utilisons l'analyseur Alloy pour l'analyse et la vérification des spécifications du client. Le courtier sélectionne les fournisseurs qui satisfont les besoins fonctionnels et non-fonctionnels du client. Il vérifie ensuite, que la configuration du placement des ressources chez les fournisseurs respecte toutes les propriétés de sécurité exigées par le client.
Toutes ces démarches sont effectuées avant le déploiement des ressources dans le nuage informatique. Cela permet de détecter les erreurs et conflits des besoins du client tôt. Ainsi, on réduit les vulnérabilités des ressources du client une fois déployées.
Mots clés : courtage informatique, nuage informatique, sécurité, Alloy, spécification, méthode formelle.

Formal specification and analysis of security policies in a cloud brokerage process

The number of cloud offerings increases rapidly. Therefore, it is difficult for clients to select the adequate cloud providers which fit their needs.
In this thesis, we introduce a cloud service brokerage mechanism that considers the client security requirements.
We consider two types of the client requirements. The amount of resources is represented by the functional requirements. The non-functional requirements consist on security properties and placement constraints. The requirements and the offers are specified using the Alloy language.
To eliminate inner conflicts within customers requirements, and to match the cloud providers offers with these customers requirements, we use a formal analysis tool: Alloy.
The broker uses a matching algorithm to place the required resources in the adequate cloud providers, in a way that fulfills all customer requirements, including security properties. The broker checks that the placement configuration ensures all the security requirements.
All these steps are done before the resources deployment in the cloud computing. This allows to detect the conflicts and errors in the clients requirements, thus resources vulnerabilities can be avoided after the deployment.
Keywords : cloud brokerage, cloud computing, security, Alloy, specification, formal method.

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

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

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE