Modélisation des Systèmes Élastiques Cloud : vers la Vérification Formelle de leur Comportement

Hamza Sahli

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Constantine 2 - Abdelhamid Mehri
Faculté des Nouvelles Technologies de l'Information et la Communication
Département de Technologies des Logiciels et des Systèmes d'Information

Année : 2017
N° d'ordre : 2017/010/DR.LMD/UC2
Série : 2017/010/DR.LMD/NTIC

T H È S E

Pour l'obtention du diplôme de Docteur en Informatique

Option : Ingénierie des systèmes d'information

Présentée et soutenue par : Hamza SAHLI

Modélisation des systèmes élastiques cloud : vers la vérification formelle de leur comportement

Dirigée par : Pr. Faiza BELALA et Dr. Nabil Hameurlain

Devant le jury composé de :

Pr. Mahmoud BOUFAIDA	Université Constantine 2 - Abdelhamid Mehri	Président
Pr. Faiza BELALA	Université Constantine 2 - Abdelhamid Mehri	Directrice
Dr. Nabil HAMEURLAIN	Université de Pau et des Pays de l'Adour	Co-Directeur
Pr. Djamel MESLATI	Université Badji Mokhtar - Annaba	Examinateur
Dr. Lakhdar DERDOURI	Université Oum El Bouaghi	Examinateur
Dr. Nabil BELALA	Université Constantine 2 - Abdelhamid Mehri	Examinateur

Soutenue le 21 Février 2017

I do so dearly believe,that no half-heartedness and no worldly fear
must turn us aside from following the light unflinchingly

J. R. R. Tolkien

Remerciements

Au terme de ce long parcours qu'est le doctorat, je tiens à adresser mes remerciements à l'ensemble des personnes qui par leurs contributions ont permis l'aboutissement de ce travail.

Je tiens, tout d'abord, à remercier mes deux directeurs de thèse pour la confiance qu'ils m'ont témoignée, et pour le climat de sérénité qu'ils ont entretenu durant ces années. Ma reconnaissance à Madame Faïza BELALA, professeur à l'Université Constantine 2 - Abdelhamid Mehri, pour avoir accepté de diriger mes travaux de recherche. Ses conseils, sa disponibilité et sa patience depuis mes premiers pas jusqu'à l'intégration totale dans la recherche m'ont été très précieux. J'adresse également mes vifs remerciements à Monsieur Nabil HAMEURLAIN, maître de conférences habilité à diriger des recherches à l'Université de Pau et des Pays de l'Adour (UPPA) pour sa contribution au sein de l'encadrement de cette recherche dont il a largement participé à son évolution.

Je tiens à remercier Monsieur Mahmoud BOUFAIDA, professeur à l'Université Constantine 2 - Abdelhamid Mehri, pour m'avoir fait l'honneur d'accepter de présider le jury de ma thèse. Je suis particulièrement sensible, par ailleurs, au grand honneur dont m'ont gratifié Professeur Djamel MESLATI de l'Université Badji Mokhtar - Annaba, Monsieur Lakhdar DERDOURI, maître de conférences à l'Université d'Oum El Bouaghi et Monsieur Nabil BELALA, maître de conférences à l'Université Constantine 2 - Abdelhamid Mehri, pour l'intérêt qu'ils ont bien voulu porter à mes travaux en acceptant d'en être les rapporteurs, qu'ils trouvent ici l'expression de ma profonde reconnaissance.

Ma gratitude infinie va aussi à ma famille et surtout mes parents qui m'ont soutenu tout au long de mes études. Je les remercie pour leurs encouragements, leur dévouement et leur soutien inconditionnel durant toutes ces années.

Résumé :

Les dernières années ont vu l'émergence d'un nouveau type de systèmes autonomiques appelés les systèmes élastiques cloud. La caractéristique qui a rendu ces systèmes très populaires dans les secteurs académiques et industriels est leur capacité de gérer et planifier la consommation des ressources pour maintenir une bonne qualité de service (QoS) dans un système cloud, tout en réduisant les coûts de fonctionnement à travers l'utilisation des solutions de contrôle d'élasticité.

L'approvisionnement des ressources suffisantes pour le bon fonctionnement d'un système cloud, sans aucun gaspillage n'est pas une tâche triviale. Une bonne solution d'élasticité doit se baser sur un modèle précis et complet qui permet de décrire les architectures des systèmes cloud et leur comportement élastique, tout en capturant les complexités internes. Dans ce contexte, les méthodes formelles caractérisées par leur efficacité, rigueur et précision présentent une solution efficace pour répondre aux exigences de modélisation et d'analyse de ce type de systèmes. L'état de l'art montre qu'aucune des quelques approches formelles actuelles, ne fournit une méthodologie générique et complète qui couvre tous les aspects relatifs à la modélisation et l'analyse des systèmes élastiques basés cloud. La majorité des propositions existantes se focalisent principalement sur le niveau infrastructure du cloud et l'élasticité horizontale sans prendre en considération des aspects structurels.

Pour pallier à ce manque, le travail présenté dans cette thèse s'intéresse à la proposition d'une approche générique et exhaustive basée sur un cadre formel permettant de spécifier les architectures des systèmes basés cloud, ainsi que la modélisation et la vérification de leur comportement élastique. En premier lieu, nous adoptons les systèmes réactifs bigraphiques et la logique de typage pour définir un modèle structurel qui permet de décrire les éléments architecturaux dans un système cloud, les relations et les dépendances entre ces éléments et aussi les contraintes structurelles et relationnelles qui en régissent le comportement élastique. Ensuite, nous proposons deux catégories principales de règles de réaction bigraphiques afin de fournir les mécanismes nécessaires pour modéliser tous les aspects comportementaux des systèmes élastiques basés cloud, tout en tenant compte des points de vue client et cloud. Enfin, pour valider l'approche proposée, nous avons développé le prototype MoveElastic, qui est un Framework basé sur un couplage judicieux entre les systèmes réactifs bigraphiques et le langage Maude pour la spécification des architectures des systèmes cloud et la vérification de leurs propriétés d'élasticité.

Mots clés : Méthodes formelles, modélisation, systèmes réactifs bigraphiques, élasticité, systèmes élastiques cloud, comportement, Maude, vérification.

Abstract :

The last few years have witnessed the appearance of a new kind of autonomic systems, called elastic cloud systems. The characteristic that has made these systems very popular in both academic and industrial sectors is their ability to manage and plan resources consumption in order to maintain a decent quality of service (QoS) in a cloud system, while reducing the operating costs by employing a solution for controlling elasticity.

The provision of just enough resources for the proper functioning of a cloud system without any waste is not a trivial task. A good elasticity solution must be based on a precise and complete model which allows describing the architectures of cloud-based systems and their elastic behaviour, while capturing the internal complexities. In this context, formal methods characterized by their efficiency, rigor and precision present an effective solution to meet the requirements for modelling and analysing this sort of systems. The state of the art shows that none of the few current formal approaches provide a generic and comprehensive methodology that covers all aspects related to modelling and analysing cloud-based elastic systems. The majority of existing initiatives focuses mainly on the infrastructure level of the cloud model and the horizontal elasticity without taking into account the structural aspects.

To overcome this lack, the work presented in this thesis is interested in proposing an exhaustive and generic approach based on a formal framework for the specification of cloud systems architectures, as well as modelling and verifying their elastic behaviour. First, we adopt bigraphical reactive systems and their sorting logic to define a structural model which allows to describe the architectural elements in a cloud system, the relations and dependencies between these elements and also the structural and relational constraints that governs the elastic behaviour. Then, we propose two main categories of bigraphical reaction rules providing the necessary mechanisms to model all behavioural aspects of cloud-based elastic systems, while taking into account both client and cloud perspectives. Finally, to validate the proposed approach, we implement the MoveElastic prototype, a formal framework based on a judicious coupling between bigraphical reactive systems and Maude language for the specification of cloud systems architectures and the verification of their elasticity properties.

Keywords : Formal methods, modelling, bigraphical reactive systems, elasticity, cloud elastic systems, behaviour, Maude, formal verification.

Liste des abréviations

Ajax	Asynchronous javascript and XML
Amazon EC2	Amazon elastic compute cloud
AP	Atomic propositions
API	Application programming interface
AWS	Amazon web services
BigMC	Bigraphical model checker
BigraphER	Bigraph evaluator and rewriting
BiLog	Bigraphical logics
BPEL	Business process execution language
BPL	Bigraphical programming language
BPL-Tool	Bigraphical programming language project tool
BPMN	Business process model and notation
BRS	Bigraphical reactive systems
CLTLt (d)	Timed constraint linear temporal logic
CPN	Colored Petri nets
CPU	Central processing unit
CSS	Cascading style sheets
ElaaS	Elasticity as-a-service
EMF	Eclipse modeling framework
Graphviz	Graph visualization software
HTML	HyperText markup language
IaaS	Infrastructure as-a-service
JSON	Javascript object notation
LTS	Labeled transition system
MAPE	Monitor, analyse, plan, execute
MDP	Markov decision process
OCaml	Objective categorical abstract machine language
OCCI	Open cloud computing interface
PaaS	Platform as-a-service
PN	Petri nets
QoS	Quality of service
QRE	QoS-aware resource elasticity
REST	Renationalisation state transfer
RSS	Rich site summary
SaaS	Software as-a-service
SAT	Satisfiability
SBP	Service-based business process
SLA	Service-level agreements
SM	Surrogate models
SML	Standard meta language
SMT	Satisfiability modulo theories
SOA	Service oriented architecture

SOAP	Simple object access protocol
TRE	Timed regular expressions
UDDI	Universal description discovery integration
UML	Unified modeling language
ViePEP	Vienna platform for elastic processes
WSDL	Web service description language
XHTML	Extensible hyperText markup language
XML	Extensible markup language

Table des matières

1 Introduction générale **1**
- 1.1 Contexte . . . 1
- 1.2 Problématique . . . 3
- 1.3 Objectifs et contributions . . . 4
- 1.4 Organisation du manuscrit . . . 6
- 1.5 Diffusion scientifique . . . 7

2 Le Paradigme cloud computing **8**
- 2.1 Introduction . . . 8
- 2.2 Définitions . . . 9
- 2.3 Caractéristiques du cloud computing . . . 11
- 2.4 Modèles de services . . . 12
- 2.5 Modèles de déploiement . . . 14
- 2.6 Acteurs du cloud computing . . . 16
- 2.7 Technologies connexes . . . 17
  - 2.7.1 Virtualisation . . . 17
  - 2.7.2 Orchestration du workflow . . . 19
  - 2.7.3 Architecture orientée services . . . 20
  - 2.7.4 Processus métier basé-service . . . 22
  - 2.7.5 Web 2.0 . . . 22
- 2.8 Conclusion . . . 23

3 Gestion autonomique de l'élasticité dans le cloud **25**
- 3.1 Introduction . . . 25
- 3.2 Principe de l'informatique autonomique . . . 26
  - 3.2.1 Propriétés autonomiques . . . 26
  - 3.2.2 Boucle de contrôle autonomique . . . 28
- 3.3 Élasticité dans le cloud . . . 30
- 3.4 Solutions de gestion autonomique de l'élasticité . . . 32
- 3.5 Modèles formels pour la spécification et l'analyse de l'élasticité . . . 39
- 3.6 Conclusion . . . 45

4 Fondements formels **46**
- 4.1 Introduction . . . 46
- 4.2 Les systèmes réactifs bigraphiques . . . 47
  - 4.2.1 Anatomie des bigraphes et forme graphique . . . 47
  - 4.2.2 Notation . . . 49
  - 4.2.3 Définitions formelles . . . 49
  - 4.2.4 Opérations sur les bigraphes . . . 51
  - 4.2.5 Forme algébrique . . . 54

4.2.6 Logique de typage . . . 56
4.2.7 Aspect dynamique des bigraphes . . . 57
4.2.8 Logique spatiale BiLog . . . 59
4.2.9 Outils autour des BRS . . . 59
4.3 Langage Maude . . . 60
4.3.1 Syntaxe et notations . . . 61
4.3.2 Langage de stratégies . . . 63
4.3.3 Analyse formelle dans Maude . . . 65
4.4 Conclusion . . . 67

5 Une architecture basée BRS pour la modélisation des systèmes élastiques cloud 68
5.1 Introduction . . . 68
5.2 Principe de modélisation . . . 69
5.2.1 Éléments architecturaux d'un système élastique cloud . . . 69
5.2.2 Méta-modèle d'un système élastique cloud . . . 70
5.3 Sémantique basée BRS d'un système élastique cloud . . . 72
5.3.1 Bigraphe front-end . . . 76
5.3.2 Bigraphe back-end . . . 78
5.3.3 Bigraphe contrôleur d'élasticité . . . 79
5.4 Conclusion . . . 82

6 Gestion et planification d'élasticité : approche formelle 83
6.1 Introduction . . . 83
6.2 Modélisation des interactions front-end/back-end . . . 84
6.3 Modélisation des méthodes d'élasticité . . . 86
6.3.1 Actions d'élasticité . . . 86
6.3.2 Transitions d'états . . . 90
6.4 Stratégie de l'élasticité . . . 91
6.5 Étude de cas . . . 93
6.5.1 Architecture du système élastique cloud . . . 93
6.5.2 Planification de l'élasticité du système élastique cloud . . . 94
6.6 Conclusion . . . 97

7 Implémentation et vérification des systèmes élastiques basés cloud 98
7.1 Introduction . . . 98
7.2 Le Framework MoVeElastic . . . 99
7.3 Implémentation de MoVeElastic . . . 101
7.3.1 Aspects structurels . . . 101
7.3.2 Aspects comportementaux . . . 104
7.3.3 Validation par simulation . . . 107
7.4 Stratégies d'élasticité dans MoVeElastic . . . 109
7.5 Vérification formelle de l'élasticité . . . 111
7.5.1 Vérification par invariants . . . 112
7.5.2 Vérification via le model-checker LTL . . . 116

7.6 Conclusion 118

8 Conclusion générale **120**
8.1 Contributions 121
8.2 Perspectives 123

Bibliographie **125**

A Module comportement Maude **133**

Table des figures

2.1 Modèles de services cloud 13
2.2 Modèles de déploiement 15
2.3 Acteurs du cloud computing 17
2.4 Machine virtuelle 18
2.5 Style architectural SOA 21

3.1 Boucle de contrôle autonomique MAPE-K [IBM Group 2005] 29
3.2 Quadruplet de l'élasticité [Galante 2012] 31
3.3 Fonctionnement global de Vulcan, un gestionnaire de planification de l'élasticité [Letondeur 2014] 33
3.4 Architecture pour l'approvisionnement de l'élasticité [Hwang 2014] . 34
3.5 Architecture ViePEP [Schulte 2012] 35
3.6 Architecture du Framework Vadara [Loff 2014] 36
3.7 Le Framework QoS-Aware Resource Elasticity (QRE) [Kaur 2014] . . 37
3.8 Les composants de ElaaS [Kranas 2012] 38
3.9 Vue générale sur le modèle MDP [Naskos 2014] 40
3.10 Aperçu le Framework basé modèle de génération de tests [Gambi 2013a] 42

4.1 Anatomie des bigraphes 48
4.2 Composition des bigraphes $A = G \circ B$ 52
4.3 Produit tensoriel des bigraphes : $A = G \otimes B$ 53
4.4 Bigraphe H modélisant un hôpital 57
4.5 Exemple d'une règle de réaction bigraphique 58

5.1 L'architecture d'un système élastique cloud [Bersani 2014] 70
5.2 Méta-modèle pour les systèmes élastiques cloud 71
5.3 Un exemple d'un bigraphe front-end F 77
5.4 Un exemple d'un bigraphe back-end B 77
5.5 Un exemple d'un bigraphe contrôleur d'élasticité E 80
5.6 Le bigraphe CS résultant du produit parallèle de F, B et E 81

6.1 Règle de réaction pour l'envoie d'une requête (a), Règle de réaction pour le déploiement de service (b) 85
6.2 Méthode d'élasticité dans un système élastique cloud 86
6.3 Réplication d'une instance de VM (a), Consolidation d'une instance d'équilibreur de charge (b) 88
6.4 Augmentation/diminution des ressources allouées à une VM (a), Migration de machine virtuelle (b) 89
6.5 Marquage d'un serveur chargé (a), Marquage d'un équilibreur de charge sous-utilisé (b) 91
6.6 Le bigraphe back-end modélisant l'architecture cloud 94

Table des figures

7.1 Étapes de MoVeElastic 100
7.2 Hiérarchie des modules Maude implémentant MoVeElastic 101
7.3 Résultats de la simulation du système cloud 108
7.4 Dépendance entre les modules d'implémentation et les modules de vérification 112
7.5 Résultats de la vérification de l'élasticité horizontale 115
7.6 Résultat de vérification d'élasticité verticale sous LTL Maude 117
7.7 Partie des résultats de vérification d'élasticité verticale par la négation 118

Liste des tableaux

2.1 Définitions du cloud computing 11

3.1 Modèles formels pour la spécification et l'analyse de l'élasticité . . . 44

4.1 Langage de termes bigraphiques 55
4.2 Langage de stratégies de Maude 64

5.1 Sémantique bigraphique des éléments d'un système élastique basé cloud 72
5.2 Contrôles et sortes du bigraphe CS 74
5.3 Règles de typage Φ_{CS} pour le bigraphe CS 75

6.1 Règles de réaction modélisant les interactions front-end/back-end . . 84
6.2 Règles de réaction modélisant des actions d'élasticité 87
6.3 Règles de réaction pour le mécanisme de marquage 90

Listings

4.1 Module système RIVER-CROSSING 65
4.2 Module de strategies RIVER-CROSSING-STRAT 65
7.1 Module fonctionnel Elastic_Front_End 102
7.2 Module fonctionnel Elastic_Back_End 103
7.3 Module fonctionnel Elastic_Cloud_System 104
7.4 Module système Elastic_Cloud_Behaviour 105
7.5 Configuration initiale du système élastique cloud 108
7.6 Module de strategies Elastic_Cloud_Controller 109
7.7 Module système Elastic_Cloud_Intial_State pour la vérification par invariants 113
7.8 Module système Elastic_Cloud_Preds pour la vérification par invariants 113
7.9 Requêtes search pour la vérification des propriétés d'élasticité 114
7.10 Module système Elastic_Cloud_Preds pour LTL Maude 116
7.11 Formule LTL exprimant des propriétés d'élasticité 117
A.1 Module Elastic_Cloud_Behaviour 133

Chapitre 1

Introduction générale

Sommaire

1.1 Contexte 1
1.2 Problématique 3
1.3 Objectifs et contributions 4
1.4 Organisation du manuscrit 6
1.5 Diffusion scientifique 7

1.1 Contexte

Au cours de ces dernières années, avec l'évolution rapide des technologies de l'information, de nombreuses organisations et entreprises ont cherché la meilleure façon pour réduire les coûts de fonctionnement, assurer la mise à l'échelle de leurs systèmes informatiques, fournir une bonne performance à leurs applications tout en optimisant l'utilisation des ressources informatiques [Soundararajan 2010]. Plusieurs technologies sont apparues au fil des années, tels que, les systèmes distribués, le traitement parallèle, le grid computing, la virtualisation, et d'autres qui abordent la question de l'approvisionnement efficace des ressources et le déploiement des applications.

Avec la croissance de la demande du marché et l'apparition régulière de nouvelles exigences métiers, ces technologies sont moins efficaces en raison de leur manque de flexibilité, d'évolutivité et leur coût parfois excessif. Par conséquent, et afin de satisfaire les exigences du marché, le paradigme "Cloud Computing" ou l'informatique en nuage a émergé cette dernière décennie, il se base sur ces technologies tout en rajoutant de nouvelles caractéristiques à l'approvisionnement des ressources informatiques. Ainsi, les entreprises adoptent de plus en plus le nouveau modèle économique proposé par le cloud computing. Un sondage impliquant 300 entreprises, mené par le [Cloud Industry Forum 2012] a montré que 53% des entreprises ont déjà adopté le cloud. Le même sondage a montré que 73% d'entre elles envisagent d'augmenter l'adoption des services cloud dans les 12 prochains mois.

Actuellement, le paradigme cloud computing est devenu massivement populaire non seulement dans le secteur industriel, mais aussi dans le monde académique. Les entreprises, les particuliers et les chercheurs pourraient louer des ressources informatiques en fonction de leurs besoins spécifiques, en ne payant que leur consommation réelle. Les clients du cloud computing deviennent donc plus diversifiés, allant

des développeurs, chercheurs, informaticiens, aux utilisateurs finaux (non informaticiens). Afin de satisfaire ce large éventail de clients, en particulier les clients sans connaissance préalable en matière de gestion d'infrastructure, les fournisseurs cloud émergents offrent différents types de services. Les services cloud computing peuvent être classifiés en trois couches ou modèles. En partant du modèle "infrastructure as-a-service" (IaaS), les fournisseurs cloud ont présenté un autre modèle de service cloud appelé "platform as-a-service" (PaaS) qui permet de développer différents types de systèmes et d'applications directement dans le cloud. En outre, le modèle "software as-a-service" (SaaS) s'est avéré utile pour les différents types de clients, surtout ceux qui n'ont aucune expérience dans les technologies de l'information et qui veulent seulement utiliser des services bien particuliers. Chacun de ces modèles de services présente des avantages et des inconvénients. Par exemple, la souplesse d'utilisation des services cloud diminue du niveau IaaS au niveau SaaS, ainsi que les connaissances requises pour utiliser chaque modèle de service. Un service IaaS propose une ressource virtualisée (une machine virtuelle) que l'utilisateur doit configurer, ce qui pourrait être compliqué. Pour un utilisateur d'un service SaaS, il suffit d'interagir avec une interface de programmation d'application (API) fournie.

Le cloud computing est basé sur une simple idée qui consiste à mettre à disposition de ses utilisateurs un ensemble de ressources informatiques virtualisées permettant d'effectuer des calculs, de stocker des données, ou de faire transiter des informations. Ces ressources sont accessibles sous la forme de services, généralement via Internet et parfois via un réseau intranet ou extranet. Les utilisateurs cloud peuvent solliciter ces services à la demande, selon leurs besoins et en ne payant ni plus ni moins que ce qu'ils consomment. Il y a plusieurs caractéristiques qui rendent le modèle de prestation de services cloud computing très populaire et attractif pour les entreprises/utilisateurs, tels que : son modèle économique fondé sur la notion de facturation "pay-as-you-go", le déploiement plus facile et plus rapide des applications au sein des infrastructures cloud, la libération des utilisateurs des préoccupations liées à la gestion du matériel, sa capacité d'évoluer en termes de ressources et l'élasticité des ressources informatiques.

Toutefois, la caractéristique la plus attrayante pour les utilisateurs cloud est la dernière citée, elle permet de distinguer le paradigme cloud computing des autres paradigmes [Dustdar 2011]. L'origine du concept d'élasticité se retrouve dans le domaine de la physique, où un objet ou un matériau solide est dit élastique, s'il est capable de retrouver sa forme d'origine après avoir été déformé. Dans le cloud computing, l'élasticité est une qualité clé qui impose un contrôle efficace sur l'approvisionnement des ressources informatiques en augmentant et diminuant leurs taux d'utilisation selon les fluctuations de la charge de travail, afin d'assurer que ces ressources limitées puissent être utilisées d'une manière illimitée.

Le principe de l'élasticité consiste à maintenir une infrastructure cloud à la taille nécessaire, à chaque instant, en assurant l'approvisionnement des ressources nécessaires et suffisantes pour le bon fonctionnement d'un système cloud même lorsque le taux d'utilisation augmente ou diminue, évitant ainsi, la sur-exploitation et la sous-exploitation de ressources. Cette notion d'adaptation dynamique de la consom-

mation de ressources en termes d'élasticité joue un rôle important pour garantir le respect des accords de niveaux de services ou "service-level agreements" (SLA) établis entre les utilisateurs du service et son fournisseur. Du point de vue utilisateur, l'élasticité assure un approvisionnement efficace de ressources qui garantit un maintien de la qualité de service (QoS) sans dépasser un budget donné. Du point de vue fournisseur, l'élasticité maximise le profit financier en assurant une meilleure exploitation des ressources informatiques et en permettant à plusieurs clients d'être servis simultanément tout en les gardant satisfaits [Vaquero 2011]. L'élasticité est fournie dans le cloud selon une stratégie réactive ou proactive, à l'aide de trois méthodes [Galante 2012] : la mise à l'échelle horizontale (l'élasticité horizontale), la mise à l'échelle verticale (l'élasticité verticale) et la migration. La méthode de la mise à l'échelle horizontale consiste à répliquer ou supprimer des instances virtuelles des services cloud. La mise à l'échelle verticale augmente ou réduit la quantité de ressources allouées à une instance de service cloud. Enfin, la méthode de migration consiste à déplacer ou redéployer une instance d'un hôte vers un autre hôte.

1.2 Problématique

L'émergence du paradigme cloud computing a donné naissance à un nouveau type de systèmes autonomiques, où l'élasticité est un principe de conception clé. Ces systèmes appelés "systèmes élastiques cloud" sont distribués et déployés sur plusieurs instances ayant différents services cloud. Le grand avantage de ce type de systèmes est leur capacité d'évoluer en termes de quantité de ressources informatiques utilisées lorsque la charge de travail est élevée, et de rétrograder dans le cas contraire (la charge de travail est faible) quand cela est possible, en réduisant le coût, et tout en maintenant la performance et la qualité de service [Dustdar 2011]. Un système élastique cloud est composé de trois entités principales, le front-end, le back-end et le contrôleur d'élasticité interagissant entre eux via différents moyens.

Ces dernières années, les systèmes élastiques cloud ont attiré une attention particulière dans les domaines industriels et académiques. Les fournisseurs de services cloud visent l'attraction de nouveaux utilisateurs afin d'augmenter leur adoption, en essayant de garantir continuellement la qualité de service (QoS) avec un coût compétitif. Ils proposent des solutions pour la gestion et la planification de l'élasticité.

Dans le monde de l'industrie, il y a plusieurs propositions pour gérer et planifier l'élasticité horizontale et verticale de manière automatique dans les environnements cloud, tels que : Rackspace, HPCloud, Amazon EC2, Google AppEngine et Microsoft Azure. Ces solutions commerciales existantes sur le marché ne sont pas encore complètement prêtes, elles sont relativement nouvelles et présentent des limites en termes de contrôle d'élasticité et d'adaptation dynamique de la consommation de ressources, ce qui influe sur la disponibilité, la performance et aussi sur la facturation finale [Gambi 2016]. De plus, les mécanismes proposés par ces solutions industrielles se limitent souvent à la spécification des services cloud requis, sans tenir compte de leurs comportements élastiques.

D'autre part, les solutions académiques ont souvent recours à des contrôleurs

d'élasticité qui surveillent le comportement des systèmes cloud en permanence pour prendre des décisions concernant l'élasticité en adoptant des stratégies réactives, proactives ou hybrides (combinaison entre des stratégies réactives et proactives). Une approche commune adoptée par les contrôleurs d'élasticité proposés pour gérer et planifier l'approvisionnement des ressources informatiques dans les environnements cloud, consiste à utiliser une boucle de contrôle fermée appelée : MAPE-K (Monitor, Analyse, Plan, Execute), introduite par IBM dans l'informatique autonomique [Jacob 2004]. La première étape de la boucle de contrôle MAPE-K, consiste à recueillir des informations de surveillance sur l'utilisation et la performance des ressources informatiques allouées à un système cloud en utilisant un ensemble de capteurs (sondes). Ensuite, ces informations sont analysées pour décider si une ou plusieurs actions doivent être déclenchées. Enfin, la dernière étape consiste à établir un plan d'actions et l'exécuter à travers des actionneurs.

Actuellement, les différentes solutions d'élasticité disponibles dans la littérature peuvent être classées en deux catégories. La première catégorie propose des contrôleurs d'élasticité qui adoptent des stratégies réactives, telles que les stratégies à base de seuils. Ce type de stratégies est très fiable dans le cas où il n'y a pas beaucoup de pics de charge de travail potentiels ou des changements de comportementaux inattendus. Cependant, elles deviennent moins efficaces lorsque la charge de travail réelle ou la performance du système différent de celles prédites initialement lors de la conception du système élastique cloud. La deuxième catégorie de solutions existantes se base sur des contrôleurs d'élasticité utilisant des méthodes proactives. Ces solutions qui sont basées souvent sur des techniques d'apprentissage automatique ou la théorie de contrôle, possèdent généralement des fondements mathématiques et statistiques solides, ce qui leur permet d'être rapides, robustes et optimales. Néanmoins, ces stratégies sont coûteuses et très difficiles à mettre en œuvre dans les environnements cloud. En outre, elles peuvent provoquer des comportements indésirables, si elles ne sont pas correctement conçues. Dernièrement, certaines initiatives ont essayé de combiner les deux types de stratégies réactives et proactives [Ali-Eldin 2012].

1.3 Objectifs et contributions

L'approvisionnement autonomique d'un immense parc de ressources logicielles et matérielles dans les systèmes cloud en termes d'élasticité n'est pas une tâche triviale. Pour être efficace, une bonne solution pour la planification et la gestion de l'élasticité doit fournir des mécanismes pour décrire les architectures de ce genre de systèmes et leur comportement élastique. Cette description sert à déterminer et comprendre les changements structurels et les dépendances comportementales dans le système. La description doit (1) considérer tous les éléments architecturaux dans un système élastique cloud, (2) capturer la complexité des interactions internes dans une architecture cloud, (3) prendre en charge des évolutions perpétuelles de ressources virtualisées, de stockage, de calcul, (4) ternir compte du comportement dans un système élastique cloud des points de vue client et cloud.

Il est clair que la maturité du paradigme cloud est loin d'être atteinte et nécessite

encore plus de labour. La définition d'une méthodologie de développement de systèmes élastiques cloud qui aidera les différents types d'utilisateurs à mieux maîtriser les différents aspects de leurs systèmes cloud, contribuera à cacher les contraintes liées à la mise à des ressources matérielles et logicielles dont ils s'en servira. L'idée est de disposer, au plus tôt dans la phase de conception, d'un modèle du système élastique à développer. Ce modèle doit s'appuyer sur une base mathématique rigoureuse prenant en compte une sémantique comportementale afin de décrire sans ambiguïté d'une part, tous les éléments architecturaux d'un tel système et leurs comportements internes, et d'autre part les aspects interaction, coordination et synchronisation entre ces éléments.

C'est dans ce contexte que s'inscrit notre travail de thèse qui a comme objectif principal de développer une approche générique et exhaustive qui aide à réduire la complexité de modélisation et d'analyse des systèmes élastiques basés cloud et leur comportement. Nous nous intéressons particulièrement à la proposition d'une approche formelle à base des systèmes réactifs bigraphiques typés (BRS with sorting) permettant de modéliser et vérifier les aspects structurels et comportementaux de ces systèmes. Nos contributions apportent des solutions aux défis suscités de la manière suivante :

- Le cadre sémantique proposé et le formalisme sous jacent (BRS) permettent de capturer la complexité architecturale d'un système élastique cloud. Nous définissons alors trois bigraphes indépendants modélisant les trois parties qui le constituent : le front-end, le back-end et le contrôleur d'élasticité. L'utilisation des bigraphes a permis de prendre en charge la mobilité des ressources impliquées, les relations et les dépendances intra éléments constitutifs d'un système cloud. Les contraintes structurelles et relationnelles qui peuvent exister sont aussi définies au moyen de la logique de typage que nous avons associé aux différents éléments des bigraphes conçus.
- Une attention particulière est accordée à la modélisation du contrôleur d'élasticité dans ce type de système. Sa structure s'inspire de la boucle de contrôle des systèmes autonomiques : MAPE-K (Monitor, Analyse, Plan, Execute). La gestion de l'élasticité et l'évolution des ressources sont prises en charge par le bigraphe interprétant cette structure.
- Le modèle proposé à l'avantage d'être dynamique, il prévoit des règles de réaction paramétriques pour modéliser les divers aspects comportementaux dans les systèmes élastiques basés cloud. D'une part, les interactions entre le front-end et back-end d'un système cloud sont représentées par une première catégorie de règles de réaction qui régissent le comportement des différents types de clients vis-à-vis des applications et des services cloud. D'autre part, la prise en charge du comportement élastique des systèmes basés cloud est aussi gérée par une autre catégorie de règles de réaction. Ces dernières règles sont classées selon les différentes méthodes d'élasticité utilisées dans les environnements cloud computing (horizontale, verticale et migration) et aussi selon les quatre niveaux cloud considérés pour la mise en œuvre de l'approvision-

nement des ressources (application, plate-forme, infrastructure et répartition de charge).

- Le comportement complexe des systèmes élastiques cloud ainsi défini, est modélisé par l'exécution des différentes classes de règles de réaction décorées avec des prédicats de la logique spatiale BiLog. Des stratégies d'élasticité sont définies pour contrôler l'approvisionnement des ressources dans le cloud, évitant ainsi les comportements indésirables, tels que l'instabilité dans l'allocation des ressources et la dégradation sévère de la qualité de service, etc.
- Dans le but de consolider les résultats théoriques de ce travail de thèse, un prototype baptisé MoVeElastic (pour Modeling and Verifying Elastic Cloud Systems) est développé pour exécuter et vérifier formellement les modèles formels proposés sous l'environnement Maude. L'intégration des modèles bigraphiques dans Maude permet de profiter des avantages des deux formalismes utilisés.

1.4 Organisation du manuscrit

Ce document est principalement structuré en deux grandes parties en plus d'une introduction et d'une conclusion. Une première partie constituée des trois premiers chapitres est consacrée à la présentation d'un état de l'art sur le paradigme cloud, l'élasticité et le cadre formel adopté. Une deuxième partie comportant les trois chapitres suivants est consacrée à la description de nos contributions.

Le chapitre 2 est consacré à la présentation du paradigme cloud computing qui constitue le contexte général dans lequel s'inscrivent les travaux présentés dans cette thèse. Il donne une description du cloud computing à travers ses définitions, ses concepts fondamentaux et ses technologies connexes.

Le chapitre 3 propose une description des différents concepts nécessaires pour comprendre les contributions proposées dans la suite de cette thèse. Il introduit tout d'abord des définitions générales sur l'informatique autonomique et l'élasticité dans le cloud. Ensuite, il explore des travaux de la littérature concernant la gestion de l'élasticité dans ce type de systèmes. Enfin, il présente et discute des approches formelles pour la modélisation et l'analyse des systèmes élastiques basés cloud. Une synthèse de ces travaux est dégagée afin de bien situer les contributions de cette thèse.

Le chapitre 4 détaille les formalismes adoptés dans cette thèse pour modéliser et vérifier les systèmes élastiques cloud. Il présente dans un premier temps les concepts de base liés aux systèmes réactifs bigraphiques, avant d'aborder le langage Maude.

Le chapitre 5 introduit la première contribution de cette thèse, il présente une approche de modélisation des systèmes élastiques cloud à base des systèmes réactifs bigraphiques et la logique de typage. Cette approche offre un cadre formel et exhaustif qui aide à réduire la complexité de conception des architectures des systèmes élastiques cloud.

Le chapitre 6 aborde la modélisation des comportements dynamiques des systèmes élastiques basés cloud en utilisant des règles de réaction bigraphiques, combi-

nées avec des prédicats de BiLog. L'exécution de ces stratégies simule le comportement de ce type de systèmes en utilisant plusieurs méthodes d'élasticité possibles, appliquées aux différents niveaux cloud.

Le chapitre 7 présente en détail l'implémentation et les expérimentations qui ont permis la validation de l'approche théorique proposée. Le prototype MoVeElastic est mis en œuvre en intégrant les systèmes réactifs bigraphiques dans le système Maude. Les outils de réécriture et d'analyse formelle autour du système Maude sont utilisés pour d'une part, exécuter et prototyper les modèles bigraphiques élaborés et d'autre part, vérifier le comportement complexe des systèmes élastiques cloud, à travers deux techniques de vérification formelle : vérification par modèles (model-checking) et recherche d'invariants.

1.5 Diffusion scientifique

Ce sujet de recherche a fait l'objet de diverses publications listées ci-dessous.

Revue scientifique avec comité de lecture

- Sahli, H., Belala, F. and Bouanaka, C. (2016) 'Formal verification of cloud systems elasticity', International Journal of Critical Computer-Based Systems, Vol. 6, No. 4, pp.364-384.
- Sahli, H., Hameurlain, N. and Belala, F. (2016). 'A bigraphical model for specifying cloud-based elastic systems and their behavior', published online (1 June 2016), The International Journal of Parallel, Emergent and Distributed Systems, pp.1-24.

Conférence internationale avec comité de lecture

- Sahli, H., Bouanaka, C., Dib, A.T.E (2014) 'Towards a formal model for cloud computing elasticity', in proceeedings of the 23rd IEEE International WETICE Conference (WETICE), Parma, Italy, pp. 359-364.
- Sahli, H., Belala, F. and Bouanaka, C. (2014) 'Model-checking cloud systems using BigMC', in proceedings of the 8th International Workshop on Verification and Evaluation of Computer and Communication Systems (VECOS 2014), Bejaïa, Algeria, pp. 25-33.
- Sahli, H., Belala, F. and Bouanaka, C. (2015) 'A BRS-based approach to model and verify cloud systems elasticity', Procedia Computer Science, vol 68, Elsevier, Pisa, Italy, pp. 29-41.
- Sahli, H., Seghir, B., Khebbeb, K. and Belala, F. (2015) 'Modeling elastic cloud services with bigraphs and agents', to appear in proceeding of the 4th International Conference on Software Engineering and New Technologies, Istanbul, Turkey.

Chapitre 2

Le Paradigme cloud computing

Sommaire

2.1 Introduction 8
2.2 Définitions 9
2.3 Caractéristiques du cloud computing 11
2.4 Modèles de services 12
2.5 Modèles de déploiement 14
2.6 Acteurs du cloud computing 16
2.7 Technologies connexes 17
2.7.1 Virtualisation 17
2.7.2 Orchestration du workflow 19
2.7.3 Architecture orientée services 20
2.7.4 Processus métier basé-service 22
2.7.5 Web 2.0 22
2.8 Conclusion 23

2.1 Introduction

Le "Cloud Computing" ou l'informatique en nuage est un nouveau paradigme de la technologie de l'information qui a été construit sur des technologies existantes et d'autres jugées nouvelles. Le principe du cloud consiste à mettre à disposition un ensemble de ressources informatiques sous forme de services que l'utilisateur peut solliciter et consommer à la demande, via Internet ou via un intranet/extranet, selon ses besoins et en ne payant que sa consommation réelle.

L'idée de fournir un service informatique centralisé remonte aux années 1960. Ce concept est apparu notamment avec McCarthy [Garfinkel 1999] ou encore Kleinrock [Kleinrock 2005] sous le nom d'informatique utilitaire. En effet, John McCarthy proposera d'employer la technologie "time-sharing" pour partager la puissance de calcul informatique et des applications sous la forme d'un service public pour différents utilisateurs. En 1966, l'ingénieur canadien Douglas F. Parkhill a publié son livre "The Challenge of the Computer Utility" [Parkhill 1966], dans lequel il décrit l'idée du calcul informatique comme un service public avec une installation informatique centralisée à laquelle nombreux utilisateurs distants peuvent se connecter via des réseaux informatiques. Parkhill introduit dans son livre certains aspects fondamentaux de cloud computing tels que : l'approvisionnement élastique des ressources,

les services publics fournis sur un réseau, l'illusion de ressources infinies. Cette idée a été très populaire dans les années 1960, mais a disparu au milieu des années 1970. Il est devenu clair que les technologies matérielles, logicielles et réseaux à l'époque n'étaient tout simplement pas prêtes. C'est ensuite vers la fin des années 90 que ce concept a réellement pris de l'importance tout d'abord avec l'émergence du "Grid Computing".

Tel que défini par le CERN (European Organization for Nuclear Research) [CERN 2006], le grid computing est un service de partage de puissance informatique et de capacité de stockage de données à travers des réseaux informatiques. Ce service peut être centralisé ou distribué sur plusieurs clusters interconnectés dans différents emplacements. Le grid computing a été principalement destiné aux initiatives scientifiques à grande échelle dans le cadre du calcul de haute performance, néanmoins son utilisation reste assez complexe pour les non spécialistes. Il a fallu ensuite attendre les années 2000 pour voir l'apparition véritable du paradigme cloud computing, s'adressant à un large public, avec Amazon EC2 [Amazon EC2 2006] ou encore la collaboration d'IBM et Google [IBM 2007]. Par la suite, de nombreuses solutions cloud open source ont vu le jour, à titre d'exemple OpenStack [OpenStack 2010] et OpenShift [OpenShift 2011].

Dans ce chapitre, nous introduisons les différents aspects relatifs au paradigme "cloud computing". Nous abordons en particulier les diverses définitions du cloud computing, les concepts clés de ce paradigme, y compris ses caractéristiques, modèles de service, de déploiements et acteurs, et enfin les technologies connexes.

2.2 Définitions

Il y a eu de nombreuses tentatives pour définir le paradigme cloud, chacune d'elles se concentre sur un aspect particulier de ce paradigme. Dans un ordre chronologique, nous résumons dans Tableau 2.1 quelques définitions pertinentes dans la littérature pour le cloud computing.

Auteurs & Année	Définition originale	Définition en français
K. Sheynkman [Geelan 2008]	Clouds focused on making the hardware layer consumable as on-demand compute and storage capacity. This is an important first step, but for companies to harness the power of the cloud, complete application infrastructure needs to be easily configured, deployed, dynamically scaled and managed in these virtualized hardware environments	Le cloud s'est concentré sur le principe de rendre la couche matérielle consommable à la demande sous forme de calcul et de capacité de stockage. Cependant, pour une exploitation optimale du cloud, les infrastructures sous-jacentes doivent être facilement configurées, déployées, mises à l'échelle de manière dynamique et gérées dans ces environnements matériels virtualisés

[Buyya 2008]	Cloud computing can be defined as a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified resources based on service-level agreements established through negotiation between the service provider and consumers	Le cloud computing peut être défini comme un type de système parallèle et distribué constitué d'un ensemble d'ordinateurs interconnectés et virtualisés qui sont dynamiquement approvisionnés à la demande comme un ensemble de ressources unifiées en fonction des SLAs établis entre le fournisseur et le consommateur
[Foley 2008]	Cloud computing is on-demand access to virtualized IT resources that are housed outside of your own data center, shared by others, simple to use, paid for via subscription, and accessed over the web	Le cloud peut être vu comme l'accès à la demande aux ressources virtualisées qui sont déployées dans des centres de données partagées entre plusieurs utilisateurs. Ces ressources sont simples à utiliser, payées par abonnement, et accessibles sur le web
[Gartner 2008]	A style of computing where massively scalable IT-enabled capabilities are delivered 'as a service' to external customers using Internet technologies	Un style informatique qui offre à des clients externes, des capacités informatiques évolutives en tant que service à l'aide de l'Internet
[Geelan 2008]	Cloud is one of those catch all buzz words that tries to encompass a variety of aspects raging from deployment, load balancing, provisioning, business model and architecture (like Web2.0). It's the n logical step in software (software 10.0)	Le cloud est un terme reflétant une solution informatique qui englobe une variété d'aspects tels que l'approvisionnement des ressources informatiques, le déploiement, l'équilibrage de charge, le modèle métier et l'architecture web 2.0
[Armbrust 2009]	Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services. The services themselves have long been referred to as Software as a Service (SaaS). The data center hardware and software is what we call a Cloud	Cloud Computing désigne à la fois les applications livrées comme des services sur Internet et le matériel et la plate-forme hébergés dans les centres de données qui fournissent ces services. Les services sont appelés "Software as a Service (SaaS)". Le matériel et la plate-forme constituent ce que nous appelons un Cloud
[Amrhein 2009]	All-inclusive solution in which all computing resources (hardware, software, networking, storage, and so on) are provided rapidly to users as demand dictates	Toute solution inclusive dans laquelle toutes les ressources informatiques (matérielle, logicielle, réseau, stockage, etc.) sont fournies rapidement aux utilisateurs à la demande
[Vaquero 2009]	Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. The pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs	Le cloud est un ensemble de ressources virtualisées, facilement utilisables et accessibles (matériels, plate-formes et/ou services). Ces ressources peuvent être configurées dynamiquement selon la charge de travail, ce qui garantit une utilisation optimale des ressources. Le pool de ressources est généralement exploité par un modèle de facturation à l'utilisation dans lequel les garanties sont offertes par le fournisseur à l'aide des SLAs
[Wang 2010]	A computing cloud is a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing infrastructures on demand, which could be accessed in a simple and pervasive way	Cloud computing est un ensemble de services accessibles de partout via des réseaux informatiques, qui fournit des ressources évolutives, garantit la qualité de service et basé sur un modèle économique simple et attractif

[Blake 2011]	Cloud computing can be seen as the use of fast, high-bandwidth Internet connections to deploy services that are centrally maintained, often by third parties, and thus minimize the cost and difficulty of IT administration and support for the organizations that consume those services	Le cloud computing peut être considéré comme l'utilisation de connexions Internet à haut débit pour déployer des services qui sont maintenus par des tiers, et ainsi minimiser le coût d'administration pour les organisations consommant ces services
[Mell 2011]	Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models	Le cloud computing est un modèle informatique ubiquitaire qui permet un accès facile, à la demande, via un réseau, à un ensemble partagé de ressources configurables (serveurs, stockage, applications et services) qui peuvent être rapidement approvisionnées et libérées par un minimum d'effort de gestion ou d'interaction avec le fournisseur de services. Le modèle favorise la disponibilité et il est composé de cinq caractéristiques essentielles, trois modèles de services et quatre modèles de déploiement

TABLE 2.1 – Définitions du cloud computing

Les premières tentatives pour définir le paradigme cloud computing à partir de l'année 2008 ont été naturellement incomplètes. De plus en plus, le cloud computing est devenu largement adopté et par conséquent sa définition est plus mature. Actuellement, la définition la plus complète et la plus adoptée du paradigme cloud dans les milieux académiques et industriels est celle introduite par le NIST (National Institute of Standards and Technology) [Mell 2011]. La suite de ce chapitre est consacrée à la présentation de cette définition, à partir des cinq caractéristiques essentielles du modèle cloud computing, ses trois modèles de services et ses quatre modèles de déploiement.

2.3 Caractéristiques du cloud computing

Le paradigme cloud computing représente un nouveau modèle pour le partage des ressources informatiques, et non pas une nouvelle technologie. Il se distingue par les cinq caractéristiques essentielles suivantes :

Libre-service à la demande

Le paradigme cloud est basé sur la notion de libre-service à la demande. Elle permettra à l'utilisateur d'interagir avec le cloud pour effectuer des tâches comme la construction, le déploiement, la gestion et la planification. L'utilisateur doit être en mesure d'accéder à des capacités de calcul et des ressources informatiques (CPU, stockage, bande passante, etc.) en fonction de ses besoins, sans la nécessité de l'intervention du fournisseur de services cloud. Cette notion de libre-service offre aux utilisateurs cloud un certain niveau d'autonomie et d'indépendance, une agilité dans

leur travail et leur permet de prendre les meilleures décisions sur les besoins actuels et futurs.

Accès universel via le réseau

Les ressources informatiques dans le cloud doivent être accessibles et gérées universellement et rapidement, en utilisant des protocoles Internet standard (généralement via des services web). Cela permet aux utilisateurs d'accéder à leurs ressources cloud à travers tout type de dispositifs et terminaux (navigateur, smartphone, tablette, etc.) à condition qu'ils aient une connexion Internet. L'accès universel est un élément clé derrière l'adoption massive du cloud, non seulement par des acteurs professionnels, mais aussi par le grand public qui est aujourd'hui familier avec les solutions basées sur le cloud tels que le stockage cloud ou la diffusion multimédia.

Mise en commun (Pooling) de ressources

Les ressources informatiques sont partagées entre plusieurs utilisateurs cloud à l'aide d'un modèle de multi location, avec assignation et redéploiement dynamiques des ressources physiques et virtuelles en fonction de la demande. Ce partage rend l'emplacement exact des données des utilisateurs impossible à déterminer. Cependant, il est possible de connaître l'emplacement des données et des ressources à un niveau d'abstraction plus élevé (pays, état, ou centre de données).

Mise à l'échelle rapide (élasticité)

Une des caractéristiques les plus importantes du paradigme cloud computing est celle de l'élasticité ou la mise à l'échelle rapide des ressources. Cette caractéristique permet aux systèmes basés cloud d'allouer ou de libérer rapidement et d'une manière automatique des ressources informatiques pour être en mesure de répondre à une montée ou à une descente en charge du travail. Pour l'utilisateur cloud, les ressources disponibles semblent souvent être infinies et peuvent être acquises avec n'importe quelle quantité et à tout moment.

Facturation à l'usage

Les systèmes basés cloud doivent être capables de se gérer pour permettre l'optimisation interne du système. Pour cela, ils s'appuient sur des mesures de référence obtenues grâce à divers mécanismes de supervision. Ces mesures permettent une facturation précise pour les utilisateurs. La facturation est calculée en fonction de la durée et de la quantité de ressources utilisées. Une unité de traitement stoppée n'est pas facturée. En outre, Il n'y a pas généralement un coût pour la mise en service.

2.4 Modèles de services

Selon le NIST, les services cloud computing peuvent être classifiés en trois couches, infrastructure as-a-service (IaaS), platform as-a-service (PaaS), et software

as-a-service (SaaS), comme indiqué dans la Figure 2.1. Les services des couches supérieures s'appuient généralement sur ceux des couches sous-jacentes, mais peuvent également être fournis en tant que services autonomes. Les utilisateurs cloud peuvent acquérir des ressources de différentes natures de divers fournisseurs. Un utilisateur peut louer des ressources infrastructures, ressources plate-formes, ressources logicielles, ou les trois types simultanément.

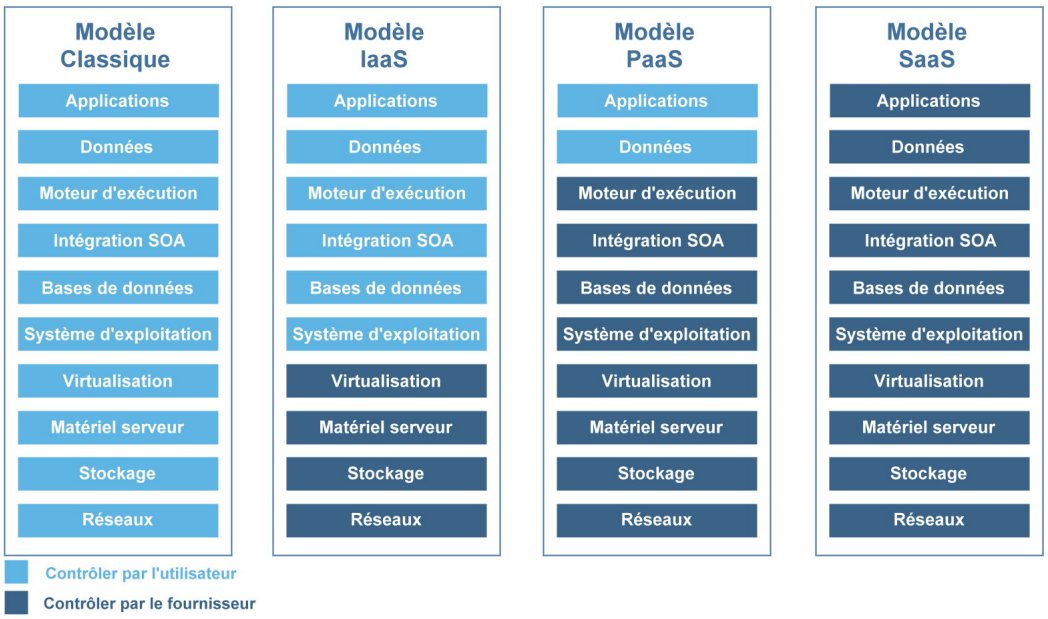


FIGURE 2.1 – Modèles de services cloud

Infrastructure as a Service (IaaS)

La couche IaaS est le niveau d'abstraction le plus bas dans le cloud. IaaS représente un modèle de service qui permet aux utilisateurs cloud d'acquérir des ressources informatiques virtualisées tels que la capacité de calcul, le stockage, la bande passante, ainsi que d'autres types de services tels que l'équilibrage de charge et les réseaux de diffusion de contenu. Le fournisseur IaaS possède et entretient l'équipement alors que l'utilisateur loue les services spécifiques dont il a besoin. En général dans IaaS, les ressources informatiques sont fournies aux utilisateurs cloud en tant qu'instances de machines virtuelles qui peuvent différer en termes de système d'exploitation en cours d'exécution, logiciels installés et ressources allouées (nombre de cœur et fréquence CPU, mémoire, bande passante et capacité de stockage). Le fournisseur IaaS permet également aux utilisateurs de gérer la mise en réseau de leurs machines virtuelles (VM), en définissant des restrictions sur leur accès ou en mettant en place des réseaux virtuels. En outre, les utilisateurs peuvent créer des copies de leurs instances virtuelles comme un moyen de sauvegarde, ou utiliser une copie d'une instance VM en cours d'exécution pour créer un modèle VM qui peut être

utilisé ultérieurement pour générer de nouvelles instances clonées. Parmi les principaux fournisseurs IaaS existants on peut citer, Amazon Elastic Compute Cloud (EC2), Amazon Web Services (AWS), Microsoft Azure, Google Compute Engine, Rackspace Open Cloud et IBM SmartCloud Enterprise.

Platform as a Service (PaaS)

La couche PaaS représente le deuxième niveau dans le modèle cloud, destiné principalement aux développeurs et aux entreprises de développement de logiciels. Le modèle PaaS propose à ce type d'utilisateurs, en plus d'un service d'utilisation de logiciel à distance, l'accès à une véritable plate-forme de développement, équipée d'un langage de programmation, d'outils de développements et des API. L'utilisateur bénéficie donc d'un environnement de développement géré, hébergé, maintenu par un prestataire et basé sur une infrastructure externe. Il aura donc la possibilité de développer des outils spécifiques pour ses activités. En général, le modèle PaaS ne remplace pas complètement toutes les ressources de l'utilisateur. Plutôt, ces ressources s'appuieront sur le modèle PaaS pour certains services clés, comme l'hébergement d'applications et le déploiement de nouveaux systèmes d'exploitation. Dans PaaS, le fournisseur prend en charge l'infrastructure et le logiciel cloud sous-jacent, alors que les utilisateurs cloud doivent seulement se connecter pour utiliser la plate-forme, généralement via un navigateur et une console web. Comme exemples de solutions PaaS, on y trouve, Amazon Web Services (Elastic Beanstalk et DynamoDB), Force.com, Google App Engine, Red Hat OpenShift, IBM Bluemix.

Software as a Service (SaaS)

La troisième couche SaaS est le niveau d'abstraction le plus haut dans le cloud. Le SaaS est un modèle de distribution de services web et de logiciels dans lequel les différents types d'applications sont hébergés et fournis par un prestataire de services, et mis à la disposition des utilisateurs via Internet (souvent via un navigateur web). Les utilisateurs cloud sont alors complètement libérés des charges de développement et de maintenance. Cependant, le fournisseur de services est responsable de la configuration des applications et la gestion des infrastructures sous-jacentes. Le modèle de prestation SaaS est de plus en plus populaire parmi les utilisateurs finaux parce qu'il ne nécessite aucune expérience dans les technologies de l'information. Nous pouvons citer comme exemple de SaaS, Gmail pour la gestion de courrier électronique, Office 365 et Google Docs pour l'édition des documents, DropBox pour le stockage.

2.5 Modèles de déploiement

Le NIST définit quatre modèles de déploiement pour le cloud computing (voir Figure 2.2) :

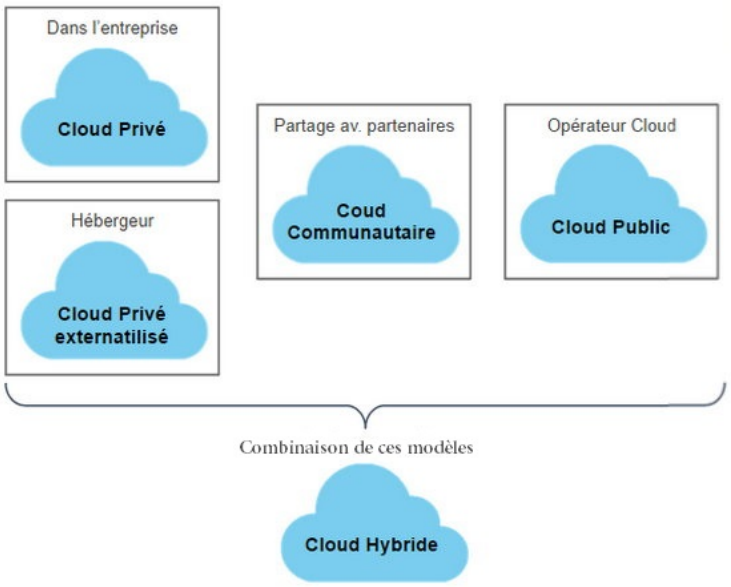


FIGURE 2.2 – Modèles de déploiement

Cloud public

Dans ce modèle, les services cloud sont fournis dans un environnement complètement détenu par un prestataire de services. Les utilisateurs d'un cloud public louent simplement les ressources informatiques nécessaires qui sont délivrées par le biais d'Internet. Cependant, ces utilisateurs n'ont aucun droit sur les infrastructures, le matériel, et les applications fournies sur le cloud. Le fournisseur de cloud met à disposition pour ces utilisateurs des ressources conçues. Il les gère, maintient et les fait évoluer en fonction des besoins des utilisateurs au fil du temps. Actuellement, le cloud public est le modèle de déploiement le plus adopté, avec des principaux fournisseurs de services cloud tels que Amazon, Microsoft et Google qui offrent des services efficaces et fiables à un large public d'utilisateurs à faible coût.

Cloud privé

Un cloud privé est une infrastructure ou une plate-forme construite spécifiquement pour servir une seule entreprise. L'infrastructure cloud pourrait être construite en interne par la même entreprise ou par un tiers. Ainsi, le cloud privé est détenu par l'entreprise utilisatrice, où elle est obligée de construire, de maintenir et de gérer l'ensemble de ses constituants, ce qui peut être à la fois compliqué et coûteux. Les clouds privés diffèrent des clouds publics du fait que le pool de ressources qui leur est associé (réseaux, serveurs, et infrastructures de stockage) est dédié à une seule entreprise et n'est pas accessible depuis l'extérieur de cette entreprise. Ce haut degré de contrôle et de transparence permet au propriétaire d'un cloud privé de se conformer plus facilement à des normes, politiques de sécurité réglementaires qui peuvent être requises dans certains domaines.

Cloud communautaire

Le cloud communautaire est un modèle de déploiement qui peut être utilisé par plusieurs entreprises ou organisations ayant des besoins communs. Un cloud communautaire partagé entre plusieurs entreprises est construit, géré, et sécurisé par l'ensemble des participants ou par un tiers (un fournisseur de services). Ces participants ont des exigences et des intérêts similaires, ils réunissent leurs moyens humains et financiers pour atteindre leurs objectifs communs. L'infrastructure commune est spécifiquement conçue pour répondre aux exigences de la communauté. À titre d'exemple, des organismes gouvernementaux, des hôtels, des hôpitaux ou des entreprises de télécommunications qui auraient des contraintes de réseau, de sécurité, de stockage, de calcul ou d'automatisation similaires pourraient trouver des intérêts communs à déployer collectivement un cloud communautaire.

Cloud hybride

Un cloud hybride comme son nom l'indique est la combinaison et la fédération de deux ou plusieurs clouds de nature différente (public, privé et communautaire). Les entreprises peuvent exiger des ressources supplémentaires en périodes de pointe. Dans ce cas, ils peuvent solliciter des ressources supplémentaires auprès des fournisseurs de cloud public. Avec un cloud hybride, une entreprise peut tirer parti de la simplicité et du faible coût d'un cloud public pour héberger des services classiques qui n'exigent aucune précaution particulière, tout en créant son propre cloud privé pour des applications étroitement intégrées aux systèmes existants ou pour le stockage de données sensibles. Elle a également la possibilité de privilégier l'utilisation de son cloud privé tout en gardant la possibilité de déborder sur une offre de cloud public en cas de besoin temporaire. Dans un cloud hybride, les clouds publics, privés ou communautaires sont des entités indépendantes, connectées entre elles par une technologie normalisée ou un propriétaire qui permet la portabilité des données et des applications.

2.6 Acteurs du cloud computing

Le NIST définit cinq acteurs majeurs dans le paradigme cloud computing : le consommateur de cloud, le fournisseur de cloud, l'auditeur de cloud, le courtier de cloud et le transporteur de cloud. Chaque acteur représente une entité (une personne ou un organisme) qui participe à une transaction ou processus et effectue des tâches dans le cloud computing. La Figure 2.3 montre les cinq acteurs définis par le NIST, leurs relations et leurs différentes interactions.

- **Consommateur de cloud (cloud costumer).** Une personne ou un organisme qui maintient une relation d'affaire avec le fournisseur de cloud.
- **Fournisseur de cloud (cloud provider).** Une personne, un organisme ou une entité responsable de mettre des services cloud à la disposition d'un ensemble de consommateurs.

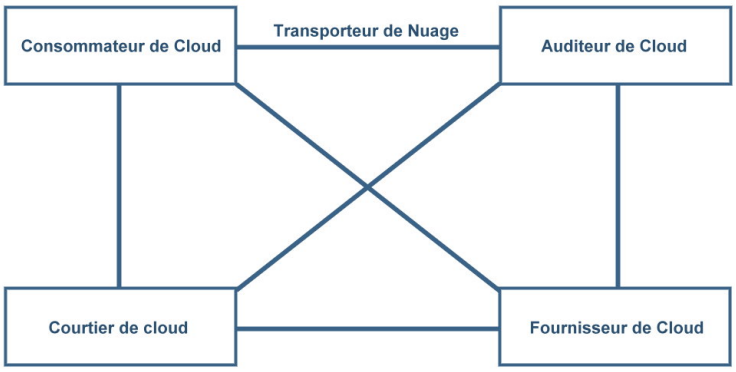


FIGURE 2.3 – Acteurs du cloud computing

- **Auditeur de cloud (cloud auditor).** Un parti qui peut procéder à une évaluation des services cloud computing en termes de performances et sécurité des services tout en restant indépendant.
- **Courtier de cloud (cloud broker).** Une entité qui gère l'utilisation, la performance et la prestation de services cloud tout en négociant les relations entre les fournisseurs et les consommateurs.
- **Transporteur de nuage (cloud carrier).** Un intermédiaire qui fournit la connectivité et transporte les services cloud du fournisseur vers le consommateur.

2.7 Technologies connexes

Le paradigme cloud computing combine un nombre important de concepts et de technologies informatiques telles que la virtualisation, l'orchestration, les architectures orientées services (SOA), les processus métiers basés-service, le web 2.0, et autres technologies basées sur Internet. Le but de cette section est d'introduire certaines de ces différentes technologies et concepts connexes.

2.7.1 Virtualisation

La virtualisation [Smith 2005] était une technologie fondamentale pour l'évolution du paradigme cloud computing dans sa forme actuelle. La virtualisation matérielle a permis aux fournisseurs cloud d'employer efficacement les ressources matérielles disponibles afin de fournir des services informatiques et de stockage à leurs utilisateurs. En particulier, la virtualisation est une méthode de déploiement de ressources informatiques qui permet de séparer les couches logicielles supérieures des couches logicielles inférieures et/ou du matériel. Le concept de virtualisation est apparu initialement dans les années 1960 [Creasy 1981] et formalisé ensuite au milieu des années 1970 [Popek 1974]. Ces dernières années, la virtualisation est de plus en plus utilisée dans le cloud computing en raison des avantages qu'elle apporte

en matière de gestion optimale des ressources informatiques et d'économie sur le matériel par mutualisation. Une infrastructure virtuelle permet de réduire les coûts informatiques tout en augmentant l'efficacité, le taux d'utilisation et la flexibilité des ressources existantes.

Dans le cloud computing, les ressources informatiques virtualisées sont appelées des machines virtuelles (VM). Une VM est un conteneur de logiciels complètement isolé, capable d'exécuter ses propres systèmes d'exploitation et applications en tant qu'une machine physique. Ainsi, une machine virtuelle se comporte exactement comme une machine physique. Elle comporte un processeur, une mémoire, un disque dur et une carte d'interface réseau virtuelle. Les VM sont créées et gérées par une couche logicielle appelée hyperviseur. Comme le montre la Figure 2.4 l'hyperviseur joue le rôle de médiateur entre le matériel (physique) et la partie logicielle (logique) permettant de faire fonctionner simultanément plusieurs systèmes d'exploitation sur une même machine physique. Ces hyperviseurs peuvent utiliser différentes techniques de virtualisation [Pooja 2014], ce qui influe sur la nature des VM qu'ils génèrent :

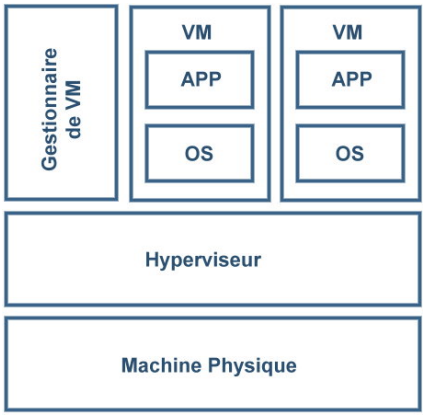


FIGURE 2.4 – Machine virtuelle

La virtualisation complète

Dans cette technique de virtualisation, le système d'exploitation hôte ou l'hyperviseur émule complètement le matériel physique qui devient visible pour les systèmes d'exploitation invités. Le logiciel d'émulation (logiciel de virtualisation) crée une couche qui atténue les différences dans les architectures matérielles et permet d'exécuter la même machine virtuelle sur différents hôtes avec différentes architectures. Cela accorde une grande flexibilité et souplesse pour faire migrer facilement les VM d'un hôte à un autre. Toutefois, cette émulation étant particulièrement coûteuse, il en résulte une forte dégradation des performances. VMWare Workstation et Virtualbox sont des exemples d'hyperviseurs utilisant cette technique.

Virtualisation basée sur le système d'exploitation hôte

Il s'agit d'installer une couche de virtualisation au-dessus du système d'exploitation hôte. Ce système d'exploitation hôte est responsable de la gestion du matériel. Les systèmes d'exploitation invités sont installés et exécutés au-dessus de la couche de virtualisation. Les applications dédiées doivent être exécutées sur des machines virtuelles. Cette approche de virtualisation basée sur l'hôte présente quelques avantages. Par exemple, l'utilisateur peut employer cette technique sans apporter aucune modification sur le système d'exploitation hôte. Le logiciel de virtualisation peut compter sur le système d'exploitation hôte pour fournir les pilotes de périphériques et d'autres services de bas niveau. Bien que cette approche soit flexible, elle est très peu utilisée par les hyperviseurs commerciaux car elle génère des performances trop faibles pour être utilisable dans la pratique.

La para-virtualisation

C'est une technique de virtualisation qui consiste à installer une couche de virtualisation directement sur le matériel. Une machine virtuelle para-virtualisée fournit des API spécifiques qui nécessitent des modifications importantes menées par le système d'exploitation sur les applications utilisateurs. La dégradation des performances est un problème essentiel dans les systèmes virtualisés. Contrairement aux autres techniques de virtualisation, la para-virtualisation tente de réduire la surcharge de virtualisation et donc, améliore les performances en modifiant seulement le noyau du système d'exploitation, ce qui la rend relativement facile et plus pratique. Il y a un nombre important d'outils et de produits de virtualisation qui utilisent cette technique de para-virtualisation, par exemple, Xen, KVM et VMware ESX.

2.7.2 Orchestration du workflow

Les systèmes cloud computing offrent un ensemble complet de modèles de service à la demande, qui pourrait être composé à l'intérieur du système cloud. Par conséquent, ces systèmes cloud devraient être en mesure d'orchestrer automatiquement les services de différentes sources et de différents types pour former un flux de service ou un workflow de manière transparente et dynamique pour les utilisateurs. L'orchestration dans le cloud est donc un type de programmation qui gère les interconnexions et les interactions entre les entités d'un système cloud donné.

L'orchestration permet d'organiser les divers composants afin qu'ils aboutissent à un résultat souhaité. Dans le contexte de l'informatique, l'orchestration consiste à combiner des tâches dans des workflows afin d'automatiser l'approvisionnement et la gestion des différents composants et ressources informatiques [Peltz 2003]. L'orchestration est plus complexe dans les environnements cloud computing, car elle implique la gestion des interactions entre les processus en cours d'exécution sur divers systèmes hétérogènes et distribués dans différents endroits.

Les produits d'orchestration des systèmes basés cloud computing simplifient la

communication entre les composants, et les connexions à d'autres applications ou utilisateurs. Ces produits comprennent généralement un portail web qui permet de gérer l'orchestration d'un seul endroit. Lors de l'évaluation des produits d'orchestration d'un système cloud, les administrateurs analysent d'abord les workflows des applications concernées. Cette étape permettra à l'administrateur de visualiser et de déterminer la complexité du workflow interne d'une application et la façon dont l'information circule souvent en dehors de l'ensemble des composants de cette application. Ceci permet d'aider l'administrateur à décider quel est le type de produit d'orchestration le plus rentable, le plus efficace, et qui répond mieux aux exigences opérationnelles du système cloud.

2.7.3 Architecture orientée services

Une architecture orientée services ou SOA (Services Oriented Architecture) est un modèle architectural pour la conception de logiciels à travers l'utilisation d'un ensemble de services atomiques [Booth 2004, Burbeck 2000]. L'objectif principal de ce modèle architectural est de répondre aux exigences de l'interopérabilité et extensibilité de l'informatique distribuée, faiblement couplée et basée sur différentes normes. Il y a beaucoup de définitions pour les architectures orientées services ; IBM Global Services [DiMare 2006] définit SOA comme une approche logique de conception de logiciels, permettant de fournir des applications en termes de services autonomes et distincts. Ces services peuvent être utilisés indépendamment des composants logiciels dont ils font partie et des plate-formes informatiques sur lesquelles ils s'exécutent.

Les services sont les principaux blocs de construction dans les architectures orientées services. Ils sont des composants logiciels autonomes qui peuvent être considérés comme des fonctions exécutées localement ou à distance, et indépendamment du langage de programmation ou de la plate-forme en cours d'exécution. Ainsi, les applications dans une architecture orientée services sont définies comme une composition de services web réutilisables, mettant en œuvre la logique métier du domaine d'application. Les services web reposent sur différentes normes basées sur XML (Extensible Markup Language) qui leur permettent de fonctionner correctement :

- SOAP (Simple Object Access Protocol), est un protocole léger de communication permettant l'échange d'informations sous forme de message XML.
- WSDL (Web Service Description Language), est un langage de description basé sur XML et normalisé par le W3C (World Wide Web Consortium). WSDL permet de décrire de façon précise les détails concernant le service web tels que les protocoles, les ports utilisés, les opérations qui peuvent être effectuées, les formats des messages d'entrée et de sortie et les exceptions pouvant être envoyées.
- UDDI (Universal Description Discovery Integration), est un annuaire distribué pour le référencement des services web, permettant à la fois la publication et l'exploration. UDDI se comporte lui-même comme un service web dont les méthodes sont appelées via le protocole SOAP. UDDI est un registre ouvert à

tout le monde et normalisé par l'OASIS (Organization for the Advancement of Structured Information Standards).

- BPEL (Business Process Execution Language), est un langage basé sur XML permettant aux services web dans une architecture orientée services d'interconnecter et de partager des données. BPEL est utilisé par les programmeurs pour définir comment un processus métier qui implique des services web sera exécuté. Les messages BPEL sont utilisés généralement pour invoquer des services à distance, orchestrer l'exécution des processus métiers et pour la gestion des événements et des exceptions.

Le style architectural SOA est structuré autour des trois acteurs de base décrits dans la Figure 2.5 : fournisseur, client et annuaire de service, ainsi que les interactions entre ces acteurs, à savoir, les opérations de publication, de recherche et d'invocation/liaison de service.

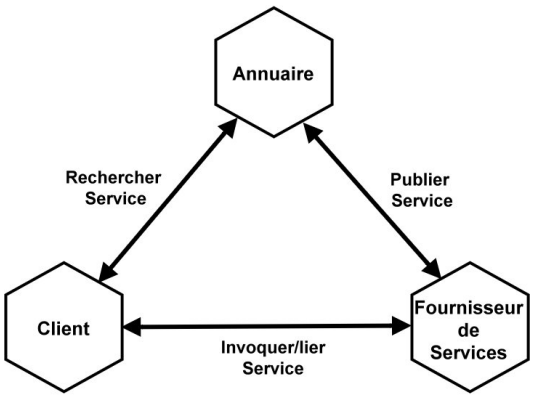


FIGURE 2.5 – Style architectural SOA

Le fournisseur de services correspond au propriétaire des services fournissant une plate-forme d'accueil pour eux. Il se charge de créer, d'héberger et de publier la description d'un service auprès d'un annuaire (ex. UDDI) afin de le mettre à la disposition des clients. Le client ou le demandeur du service représente l'entité logicielle qui recherche et invoque un service web. L'application cliente peut être elle-même un service web. L'annuaire est un registre de description de services. Il représente l'entité qui regroupe des informations sur les services web disponibles et la façon d'y accéder.

Les architectures orientées service (SOA) et le cloud computing sont considérés de manière indépendante. Néanmoins, ils sont des technologies complémentaires et peuvent être très utiles les uns pour les autres. De plus en plus, ils assument des rôles importants dans les grandes organisations et entreprises qui les utilisent pour augmenter l'efficacité opérationnelle de leurs applications. Généralement, les différentes organisations rencontreront des difficultés dans l'utilisation du cloud computing du fait que les applications dans le cloud ne disposent pas d'une base architecturale solide. SOA est considérée dans ce cas comme un prérequis pour les applications

déployées dans le cloud.

2.7.4 Processus métier basé-service

Un processus métier est un ensemble d'une ou plusieurs procédures ou activités liées qui réalisent collectivement un objectif ou une politique métier, généralement dans le cadre d'une structure organisationnelle définissant les rôles fonctionnels et leurs relations. Un processus peut être entièrement contenu dans une seule unité organisationnelle ou peut couvrir plusieurs organisations, comme dans les relations client-fournisseur [Liu 2009].

Un processus métier basé-service (SBP) est un processus métier qui consiste à regrouper un ensemble de services web élémentaires qui sont liés en termes de leur contribution à la réalisation globale du processus métier. Un service dans un processus métier est généralement la plus petite unité de travail, il représente un module offrant des capacités de calcul ou de données. Les services réalisent les différentes activités du SBP considéré. L'assemblage des services dans un SBP peut être assuré en utilisant diverses spécifications et modèles de composition de service (ex. BPEL).

Les processus métiers basés-service peuvent être conçus en utilisant une grande variété de langages tels que les langages formels, langages conceptuels et langages d'exécution. Les langages formels tels les chaînes de Markov et les réseaux de Petri sont basés sur des modèles théoriques et ils sont généralement fournis avec une sémantique précise et sans ambiguïté. Ils permettant de vérifier l'exactitude et la performance à travers plusieurs techniques d'analyse formelle (model-checking, simulation, etc.). En revanche, les langages conceptuels tels que UML (Unified Modeling Language) et BPMN (Business Process Model and Notation) sont généralement informels. Ces langages ne disposent pas d'une sémantique bien définie et ne permettent pas d'effectuer des analyses. Les langages d'exécution comme BPEL sont utilisés généralement pour spécifier le comportement des processus métiers.

Les environnements cloud sont de plus en plus adoptés par les différentes entreprises qui utilisent les infrastructures et plate-formes cloud pour déployer, héberger et exécuter leurs services, en particulier en termes de processus métiers basés-service (SBP).

2.7.5 Web 2.0

L'expression web 2.0 est souvent utilisée pour désigner la seconde phase du world wide web, passant d'une collection de sites web à une plate-forme informatique à part entière, fournissant des applications web aux utilisateurs [Andersen 2007] [O'Reilly 2009]. Les utilisateurs finaux pourront avoir un accès facile à leurs données de n'importe quel poste et à tout moment. Le terme web 2.0 a été introduit par Tim O'Reilly et Dale Dougherty en octobre 2004 lors d'une conférence brainstorming entre les deux sociétés O'Reilly Média et MediaLive International pour nommer l'évolution du world wide web.

En effet, le web 2.0 permet d'avoir une grande collaboration entre les utilisateurs d'Internet, les fournisseurs de contenu et les entreprises. Cela représente l'une des

différences les plus importantes entre le web 2.0 et le web traditionnel (Web 1.0). Dans les approches web traditionnelles, les données sont publiées sur les sites web, et les utilisateurs peuvent seulement consulter ou télécharger le contenu. De plus en plus, les utilisateurs ont plus d'impact sur la nature et la portée du contenu des sites web. Par conséquent, le web 2.0 est une technologie émergente décrivant les tendances novatrices d'utiliser la technologie de world wide web et la conception de web afin d'améliorer la créativité, le partage de l'information, la collaboration et la fonctionnalité du web.

Les applications web 2.0 comprennent généralement les caractéristiques et les techniques suivantes :

- "Cascading Style Sheets" (CSS) pour séparer la présentation et le contenu.
- Les folksonomies pour l'étiquetage collaboratif, la classification sociale, l'indexation et l'étiquetage social.
- Technologies du web sémantique.
- REST, XML et les API basés JSON.
- Les techniques innovantes du développement web telles que Ajax.
- XHTML et les balises HTML.
- La syndication, l'agrégation et les notifications des données web avec RSS.
- Les mashups et les applications hybrides, la fusion du contenu de différentes sources, et les architectures client-serveur.
- Les outils de publication "weblog".
- Les wikis pour soutenir le contenu généré par l'utilisateur.
- Les outils pour gérer la vie privée des utilisateurs sur Internet.

L'idée essentielle derrière le web 2.0 est d'améliorer l'inter-connectivité et l'interactivité des applications web. Ce nouveau paradigme pour le développement des applications web permet aux utilisateurs d'accéder au web plus facilement et efficacement. Les services cloud computing dans leur nature sont des applications web qui rendent des services informatiques à la demande. Donc il était impératif que le cloud computing adopte la technologie web 2.0, parce qu'elle représente une évolution technique naturelle.

2.8 Conclusion

Le cloud computing n'est pas un effet de mode. C'est une révolution dans la manière d'organiser, de gérer et de distribuer des ressources informatiques. C'est une nouvelle technologie qui permet de faire beaucoup mieux pour beaucoup moins cher.

Dans ce chapitre, nous avons introduit le paradigme cloud computing et ses différents aspects sous-jacents. Nous avons commencé par présenter les diverses définitions pertinentes disponibles dans la littérature dans un ordre chronologique. De

plus, nous avons présenté les concepts clés du paradigme cloud, à savoir : ses caractéristiques essentielles, modèles de service, modèles de déploiement, et acteurs du cloud. Enfin, nous avons décrit les technologies connexes au cloud computing : la technologie de virtualisation, l'orchestration du workflows, les architectures orientées services (SOA), les processus métiers basés-service et le web 2.0. Dans le chapitre suivant, nous présenterons un aperçu sur l'élasticité et l'informatique autonomique dans le cloud. La propriété d'élasticité définit la capacité d'un système cloud à supporter une montée en charge de travail tout en conservant une performance adéquate à condition que les ressources matérielles soient ajoutées.

Chapitre 3

Gestion autonomique de l'élasticité dans le cloud

Sommaire

3.1 Introduction 25
3.2 Principe de l'informatique autonomique 26
3.2.1 Propriétés autonomiques 26
3.2.2 Boucle de contrôle autonomique 28
3.3 Élasticité dans le cloud 30
3.4 Solutions de gestion autonomique de l'élasticité 32
3.5 Modèles formels pour la spécification et l'analyse de l'élasticité 39
3.6 Conclusion 45

3.1 Introduction

Avec l'évolution de l'informatique et l'émergence de nouveaux systèmes à grande échelle, complexes et distribués, l'intervention humaine devient de plus en plus difficile et trop lente à réagir pour prendre en compte certaines situations, liées au changement du contexte d'exécution, telles que la défaillance du réseau, matérielle et logicielle ou la fluctuation soudaine de la charge de travail.

Dans un environnement hautement dynamique comme le cloud, il est impératif de s'appuyer sur une gestion automatique et dynamique des ressources informatiques afin d'assurer une bonne qualité de service (QoS), garantir une bonne performance, et répondre aux besoins des clients en fonction des accords de niveaux de services (SLA). Pour satisfaire ces exigences, l'informatique autonomique est utilisée dans le cloud computing pour gérer et planifier la consommation de ressources en termes d'élasticité. La gestion autonomique dans le cloud est la capacité de contrôler automatiquement et dynamiquement l'approvisionnement des ressources informatiques selon les fluctuations de la charge de travail. L'informatique autonomique est généralement représentée dans les systèmes élastiques basés cloud par un contrôleur d'élasticité doté d'une boucle de contrôle autonomique MAPE-K (Monitor, Analyze, Plan and Execute).

Ce chapitre vise à offrir au lecteur une vision à la fois claire et concise du contexte de la thèse qui porte sur la modélisation et l'analyse des systèmes basés cloud et

leur comportement en termes d'élasticité. Pour cela, ce chapitre commence par décrire dans une première section l'informatique autonomique et sa boucle de contrôle autonomique MAPE-K comme un modèle pour la gestion d'élasticité dans le cloud. Dans une deuxième section, ce chapitre aborde le principe d'élasticité, sa définition et le quadruplet de l'élasticité : niveau, stratégie, objectif et méthode. La troisième section présente un aperçu des solutions académiques qui proposent des approches pour la gestion autonomique de l'élasticité dans le cloud. Enfin, dans une quatrième section, ce chapitre présente et évalue un ensemble d'approches formelles récentes basées sur des formalismes différents pour la modélisation et l'analyse des systèmes cloud et leur comportement élastique.

3.2 Principe de l'informatique autonomique

L'informatique autonomique a été introduite initialement en 2001 par Paul Horn, au cours d'un discours à l'université de Harvard. Cette idée a été concrétisée plus tard dans un manifeste publié pour marquer le début de l'approche [Horn 2001, Jacob 2004]. Un système autonomique est défini comme un système qui s'administre lui-même, sans avoir besoin d'aucune intervention humaine, afin d'assurer au mieux les objectifs des utilisateurs [Bantz 2003]. Ces systèmes doivent anticiper les besoins pour permettre aux utilisateurs de se concentrer sur ce qu'ils veulent accomplir plutôt de passer leur temps à superviser le comportement de leurs systèmes. L'idée principale des systèmes autonomiques est inspirée du fonctionnement des systèmes nerveux humains, qui peuvent observer et adapter le corps humain à l'environnement en lui permettant de rester dans un bon état [Mainsah 2002, Kephart 2003]. Par exemple, le système nerveux contrôle la température du corps, la fréquence cardiaque, le niveau de sucre dans le sang, et autres fonctions vitales. Dans cette section, nous introduisons l'informatique autonomique en détaillant les propriétés autonomiques et la boucle de contrôle autonomique.

3.2.1 Propriétés autonomiques

L'informatique autonomique est une tentative initiée en 2001 par IBM afin de surpasser la complexité des systèmes informatiques de nos jours et ceux du futur en leur accordant des capacités d'auto-gestion [Kephart 2003].

Selon Horn [Horn 2001], les systèmes autonomiques sont des systèmes auto-gérés avec huit caractéristiques ou éléments clés : auto-configuration, auto-guérison, auto-optimisation, auto-protection, auto-connaissance, sensibilité au contexte, ouverture, et auto-adaptation. Les quatre premiers éléments sont considérés comme caractéristiques principales de ces systèmes, alors que les quatre autres éléments sont des caractéristiques complémentaires [Berns 2009, Huebscher 2008, Kephart 2003] :

Auto-configuration (self-configuration)

L'auto-configuration se réfère à la capacité d'un élément autonome d'être parfaitement intégré dans un système selon la nature et l'état de ce système ou les

objectifs des utilisateurs. Cette propriété permet à un système informatique donné de se reconfigurer et de se mettre à jour tout seul selon des objectifs de haut niveau prédéfinis par l'être humain. L'administrateur du système devrait seulement spécifier la sortie désirée et le système serait adapté automatiquement en lui permettant de fonctionner correctement pour répondre aux objectifs de départ.

Auto-guérison (self-healing)

C'est une propriété qui représente la capacité d'un système informatique à détecter, diagnostiquer puis compenser ou réparer des pannes survenant dans le système. Les types de problèmes qui sont détectés peuvent être interprétés au sens large : ils peuvent être des erreurs de bas niveau (défaillances matérielles) ou des erreurs de haut niveau (problèmes logiciels). Ainsi, les systèmes autonomiques peuvent diagnostiquer ces problèmes qui se produisent au cours de leur fonctionnement et faire des réparations sans aucune perturbation possible. Un système est dit alors "self-healing" ou réactif aux défaillances quand il est capable de garder son activité dans un état stable en dépassant les événements gênants dus à la défaillance de ses éléments.

Auto-optimisation (self-optimisation)

C'est la capacité à chercher continuellement des moyens pour optimiser l'efficacité soit à l'égard de la performance ou du coût. À titre d'exemple, l'optimisation de taux d'utilisation de ressources informatiques. Les opérations d'optimisation peuvent être réactives à l'état du système ou à l'environnement, mais également peuvent être initiées d'une manière proactive. Ainsi, cette propriété permet aux systèmes autonomiques d'adapter une configuration optimale pour éviter des états indésirables comme la surcharge ou la sous-charge. L'optimisation se fait en respectant des critères définis généralement par les utilisateurs du système.

Auto-protection (self-protection)

Un système autonomique peut se protéger contre les attaques malveillantes et les actions qui peuvent le déstabiliser et le rendre inactif. Cette propriété d'auto-protection permet à un système de garantir la sécurité, la confidentialité et la protection de ses données. En outre, l'auto-protection doit permettre aux systèmes informatiques autonomiques d'anticiper les diverses attaques et tous les types de menaces de sécurité en vue de prendre des précautions et des réactions adéquates.

Auto-connaissance (self-knowledge)

C'est la capacité d'un système autonomique de connaître son environnement et ses capacités ultimes ou d'être en mesure de les identifier ou de les calculer. Dans ce cas, pour être en mesure de se gérer correctement, le système doit connaître ses composants, ainsi que leurs états, leurs capacités actuelles, et leurs liens avec d'autres systèmes. De plus, il doit savoir quelles sont les ressources qui lui appartiennent, ainsi

que les propriétaires des autres ressources qu'il peut emprunter ou prêter, et enfin quelles sont les ressources qui peuvent être partagées.

Sensibilité au contexte (context-awareness)

Elle caractérise la capacité d'un système autonomique de percevoir et d'utiliser les différentes informations relatives au contexte, telles que la localisation, le temps, la température ou l'identité de l'utilisateur pour adapter dynamiquement sa fonctionnalité selon des besoins pré-définis. Cette propriété permet à un système autonomique de décrire son état et l'état de ses ressources pour interagir avec d'autres systèmes. Elle lui permet aussi de découvrir automatiquement l'état des ressources disponibles dans l'environnement ou le contexte actuel.

Ouverture (openess) et auto-adaptation (self-adaptation)

Outre les propriétés précédemment mentionnées, les systèmes autonomiques doivent mettre en œuvre des normes ouvertes pour être correctement intégrés dans des environnements hétérogènes (ouverture). De plus, ils doivent anticiper leurs besoins tout en gardant leur complexité cachée des utilisateurs (auto adaptation). Par exemple, un serveur d'applications autonomique peut comparer les données d'accès en temps réel avec les données sauvegardées précédemment dans un historique pour anticiper les dégradations de performances et faire les adaptations nécessaires.

3.2.2 Boucle de contrôle autonomique

Les systèmes autonomiques sont basés généralement sur un gestionnaire autonomique qui contrôle les différentes transitions dans le système, et les comportements internes de ces éléments. Ce gestionnaire autonomique consiste en une boucle de contrôle intelligente et fermée. Dans cette sous-section, nous abordons la boucle de contrôle autonomique MAPE-K introduite par IBM [IBM Group 2005]. Nous détaillons les quatre fonctions principales de la boucle, partageant une base de connaissances, qui sont : l'observation, l'analyse, la planification et l'exécution (Figure 3.1).

La tâche d'observation ou de supervision (monitoring)

L'observation dans la boucle de contrôle MAPE-K propose des mécanismes de capture de données et d'informations de l'environnement (physique ou virtuel) relatives aux propriétés de gestion autonomique du système. Ces données recueillies de l'élément administré via ses capteurs (sondes) sont agrégées, corrélées et filtrées afin de déterminer le symptôme qui doit être analysé. Les informations traitées peuvent être transmises périodiquement au composant chargé d'analyse ou lorsque le composant de surveillance le juge nécessaire. Dans le second cas, les informations collectées doivent être donc analysées plus profondément.

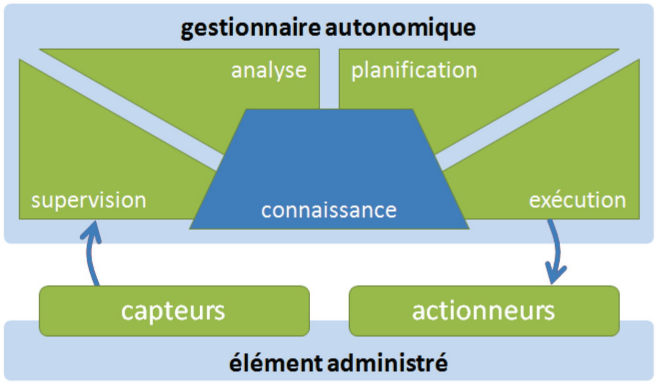


FIGURE 3.1 – Boucle de contrôle autonomique MAPE-K [IBM Group 2005]

La tâche d'analyse

Raisonne sur les données complexes d'observations fournies par les capteurs en fonction des politiques et stratégies de gestion de haut niveau. Le composant d'analyse compare ces informations avec les politiques globales de gestion du système décrites par l'administrateur. Cette comparaison fournit des indicateurs quantitatifs et qualitatifs sur l'état du système, permettant de générer des requêtes de modifications transmises au composant de planification.

La tâche de planification

Implique la prise en compte des données de surveillance en provenance des capteurs afin de produire des plans d'action définissant la manière dont les modifications requises vont être mises en œuvre. Ces modifications dépendent des décisions prises par le composant chargé de l'analyse. En outre, les modifications à apporter sur l'élément administré peuvent être planifiées d'une façon temporelle. Cela est réalisé par la définition d'un ensemble de contraintes temporelles retardant les modifications ou l'établissement d'un ordre ou des priorités d'exécutions à prendre en compte.

La tâche d'exécution

Assure la mise en œuvre des plans de modifications recommandés par le composant dédié. Cela est réalisé en exécutant les actions nécessaires sur l'élément administré à travers ses propres actionneurs. Si une modification donnée concerne plusieurs éléments administrés, les actions peuvent être distribuées sur plusieurs machines.

La base de connaissances (knowledge)

Représente les informations et les données provenant de diverses sources, partagées entre les différentes tâches de la boucle MAPE-K. Ces connaissances partagées peuvent être exploitées et mises à jour constamment par n'importe de ces quatre tâches.

3.3 Élasticité dans le cloud

La caractéristique clé des systèmes élastiques basés cloud est leur capacité de s'adapter d'une manière autonomique et en temps réel selon les changements des circonstances de fonctionnement (par exemple, augmentation dans le nombre de requêtes reçues dans le système). Cette qualité qui caractérise ce type de système est la propriété d'élasticité. Elle représente une caractéristique principale qui distingue le paradigme cloud computing des autres paradigmes. Grâce à cette propriété, les systèmes élastiques basés cloud sont capables d'ajuster dynamiquement leur allocation de ressources pour maintenir une qualité de service (QoS) adéquate et prédéfinie dans les accords de niveaux de services (SLA) en fonction des fluctuations de la charge de travail, tout en minimisant les coûts de fonctionnement. En d'autres termes, l'adaptation autonomique de la consommation de ressources informatiques dans les systèmes élastiques basés cloud en termes d'élasticité, vise à éviter à la fois le sur-approvisionnement (l'allocation de plus de ressources que nécessaire) et le sous-approvisionnement (l'allocation de moins de ressources que nécessaire) dans le but d'assurer une bonne qualité de service et minimiser les coûts.

L'élasticité est souvent associée à l'évolutivité ou "la scalabilité" d'un système cloud. Mais en réalité ces deux concepts sont différents et ne devraient jamais être utilisés pour exprimer la même chose. La scalabilité (évolutivité) d'un système cloud est sa capacité à ajouter des ressources informatiques pour répondre à une croissance de charge de travail [Agrawal 2011, Coutinho 2013]. Dans la littérature, il existe plusieurs définitions de l'élasticité dans le cloud computing. La définition la plus adoptée actuellement dans les cercles des recherches est donnée par [Herbst 2013] comme suit :

Définition 3.1. [Herbst 2013] L'élasticité est le degré auquel un système est capable de s'adapter aux changements de la charge de travail en approvisionnant et désapprovisionnant des ressources de manière autonomique, de telle façon à ce que les ressources fournies soient conformes à la demande du système.

Selon la classification proposée par Galante et de Bona [Galante 2012], l'élasticité est caractérisée par le quadruplet : < niveau, stratégie, objectif, méthode > (voir Figure 3.2). Le niveau spécifie la cible de l'élasticité, à savoir le niveau : infrastructure, plate-forme, application et répartition de charge. La stratégie est la politique de mise en œuvre de l'élasticité, elle peut être proactive, réactive ou hybride. L'objectif représente le but derrière le redimensionnement. À titre d'exemple, améliorer les performances d'un système cloud. Enfin, la méthode utilisée dans la mise en œuvre de l'élasticité à savoir l'élasticité horizontale (mise à l'échelle horizontale), l'élasticité verticale (mise à l'échelle verticale) et la migration. La suite de cette section est consacrée à la présentation détaillée du quadruplet < niveau, stratégie, objectif, méthode>.

Niveaux d'élasticité

Ce niveau représente la portée d'une action d'élasticité. Une telle portée est définie selon la nature des ressources ou par une des couches d'un système cloud :

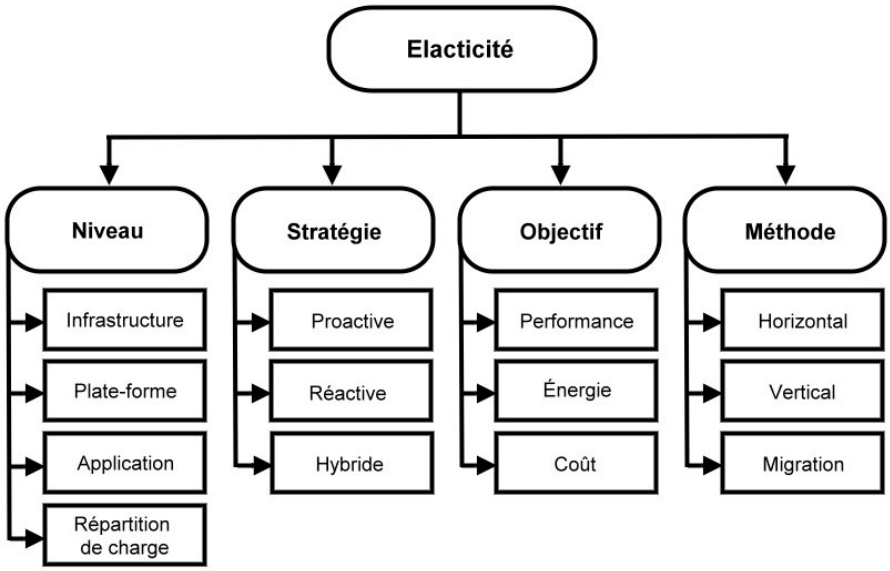


FIGURE 3.2 – Quadruplet de l'élasticité [Galante 2012]

infrastructure, plate-forme, application et même le niveau de répartition de charge. Principalement, l'élasticité est assurée au niveau IaaS (infrastructure as-a-service) où les ressources approvisionnées sont le plus souvent des instances de machines virtuelles. Néanmoins, d'autres services d'infrastructure peuvent également être redimensionnés ou répliqués telles que la bande passante et la capacité de stockage. Au niveau PaaS (platform as-a-service), l'élasticité consiste au redimensionnement ou la réplication des conteneurs ou encore des bases de données. En outre, l'élasticité est mise en œuvre au niveau SaaS (software as-a-service) où des ressources supplémentaires peuvent être accordées à des applications ou des services web. Enfin, l'élasticité peut être réalisée au niveau de la répartition de charge par le redimensionnement d'équilibreurs de charge ou par l'augmentation de leur nombre.

Stratégies d'élasticité

Les stratégies sont des politiques d'élasticité qui représentent les interactions nécessaires pour l'exécution des actions d'élasticité. Dans le cloud computing, l'élasticité est mise en œuvre selon trois types de stratégie : proactive, réactive et hybride (proactive et réactive) [Galante 2012]. Les stratégies proactives utilisent des techniques heuristiques, mathématiques et analytiques pour anticiper les fluctuations de la charge de travail et les changements de comportement dans les systèmes élastiques basés cloud et aussi pour décider quand et comment mettre le système à l'échelle. Les stratégies réactives sont basées sur le mécanisme Règle-Condition-Action. Une règle est composée d'un ensemble de conditions qui déclenchent des actions d'élasticité lorsqu'elles sont remplies afin de prendre des mesures nécessaires pour augmenter ou réduire la quantité de ressources allouées. Chaque condition prend en considération

un événement ou une métrique du système et le compare aux informations obtenues d'un service de monitoring. Enfin, une stratégie hybride est une combinaison d'une stratégie proactive et d'une stratégie réactive.

Objectif et méthode

Dans les systèmes basés cloud, l'élasticité est utilisée pour plusieurs raisons et pour réaliser divers objectifs [Vaquero 2011]. Du point de vue du fournisseur, l'élasticité assure une meilleure utilisation des ressources informatiques ce qui fournit des économies d'échelle et permet de servir plusieurs utilisateurs simultanément. Du point de vue de l'utilisateur, l'élasticité est utilisée principalement pour éviter l'insuffisance des ressources, et par conséquent, la dégradation des performances du système. En outre, l'élasticité peut être utilisée pour assurer l'efficacité énergétique, où l'utilisation de la quantité minimum de ressources permettant d'atteindre cet objectif.

L'élasticité dans le cloud [Galante 2012, Coutinho 2015] peut être fournie à l'aide de trois méthodes. L'élasticité horizontale consiste à la réplication (scale up) ou la suppression (scale down) des instances de machines virtuelles (VM), conteneurs, applications (services web) et équilibreurs de charge selon les changements de la charge de travail. La mise à l'échelle horizontale est la méthode la plus utilisée actuellement pour assurer l'élasticité dans les environnements coud computing. L'élasticité verticale appelée aussi "redimensionnement ou mise à l'échelle verticale" modifie la capacité globale des ressources allouées à une instance (VM, conteneur, application, équilibreur de charge) en ajoutant (scale up) ou supprimant (scale down) des ressources telles que CPU, mémoire, disque, bande passante. L'élasticité verticale est la méthode la plus simple. Néanmoins, elle est moins populaire que l'élasticité horizontale du fait qu'elle elle moins efficace et plus restrictive. Enfin, la méthode de migration consiste à transférer une machine virtuelle, un conteneur ou une application en cours d'exécution d'un hôte surchargé à un autre moins chargé. La migration est généralement utilisée dans le cloud pour d'autres objectifs tels que la tolérance aux pannes et l'isolation. Néanmoins, certaines solutions cloud non élastiques emploient cette méthode pour simuler le comportement obtenu par la mise à l'échelle verticale. Par exemple, en déplaçant une machine virtuelle d'un serveur chargé à un autre moins chargé afin de gérer une augmentation de la charge de travail [Knauth 2011], ou en redéployant un service web ou une application d'une machine virtuelle chargée ou défaillante à une autre.

3.4 Solutions de gestion autonomique de l'élasticité

Au cours de ces dernières années, un effort accru a été orienté vers la gestion autonomique des ressources informatiques dans le cloud en termes d'élasticité. Plusieurs contributions concernant la gestion et la planification de l'élasticité horizontale et verticale à différents niveaux, fournissant des contrôleurs d'élasticité autonomiques basés sur différentes stratégies, proactives et/ou réactives. Cette section présente

quelques solutions académiques pour garantir l'élasticité dans les environnements cloud computing.

Dans [Al-Shishtawy 2013], les auteurs ont proposé un gestionnaire d'élasticité appelé ElastMan qui permet d'automatiser l'élasticité des modèles de stockage de données "key-value stores" qui s'exécutent dans le cloud. Le gestionnaire ElastMan redimensionne automatiquement les services cloud selon les changements de la charge de travail afin d'assurer une bonne qualité de service (QoS) avec un coût minimum. ElastMan consiste en deux composants principaux : le composant Feedforward Controller et le composant Feedback Controller. En combinant ces deux composants, ElastMan peut gérer la performance des machines virtuelles dans le cloud, les fluctuations de la charge de travail et les exigences rigoureuses de qualité de service (QoS). Le composant Feedforward Controller de ElastMan surveille la charge de travail et utilise un modèle de régression logistique pour prédire si la charge de travail violera les accords de niveaux de services (SLA), et réagir selon cette prédiction. Le composant Feedforward Controller est utilisé principalement pour répondre rapidement aux fluctuations brusques de la charge de travail. Le deuxième composant Feedback Controller surveille la performance de service et réagit selon l'écart de la déviation de la performance souhaitée spécifiée dans l'accord de niveaux de services (SLA). Le composant Feedback Controller est employé dans ElastMan pour corriger les erreurs dans le modèle de régression logistique utilisé par le composant Feedforward Controller et pour gérer les changements graduels dans la charge de travail.

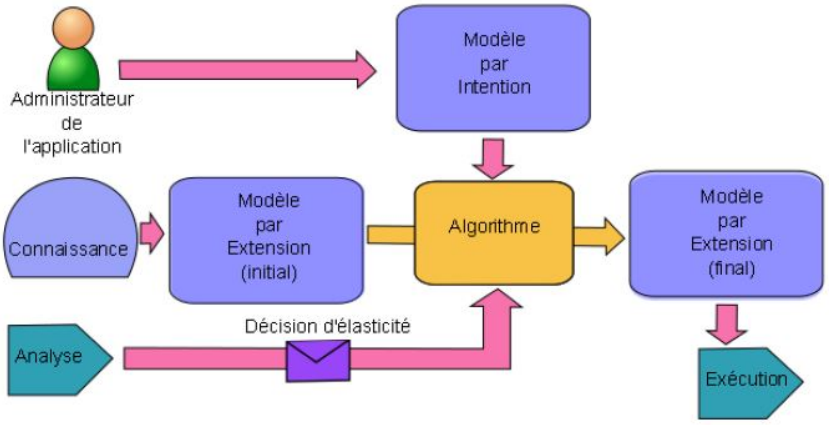


FIGURE 3.3 – Fonctionnement global de Vulcan, un gestionnaire de planification de l'élasticité [Letondeur 2014]

Les auteurs de [Letondeur 2014] ont proposé un Framework pour la gestion autonomique de l'élasticité des applications dans le cloud nommé Vulcan. En particulier, ils ont proposé une boucle de contrôle autonomique fermée basé sur le modèle MAPE-K (Monitor, Analyze, Plan and Execute) de IBM permettant d'assurer l'élasticité tout en considérant des scénarios complexes dans les systèmes basés cloud. Cette boucle de contrôle consiste en un ensemble de composants. Premièrement le

composant Planification qui décrit comment une application doit se reconfigurer en fonction d'une décision d'élasticité. Comme le montre la Figure 3.3, lorsque le composant de planification reçoit une décision d'élasticité d'un composant Analyzer, il calcule le nouvel état de l'application à contrôler (nouvelle architecture de l'application). Pour accomplir cette tâche, il utilise une description initiale qui représente l'architecture actuelle de l'application et une autre description qui consiste en toutes les architectures possibles de l'application. Un algorithme associé utilise ces deux descriptions pour calculer et déterminer les modifications à effectuer dans l'architecture de l'application selon la décision d'élasticité. Ces modifications consistent principalement à l'ajout/suppression d'une machine virtuelle ou conteneur ainsi que d'autres opérations pour la reconfiguration de l'architecture actuelle.

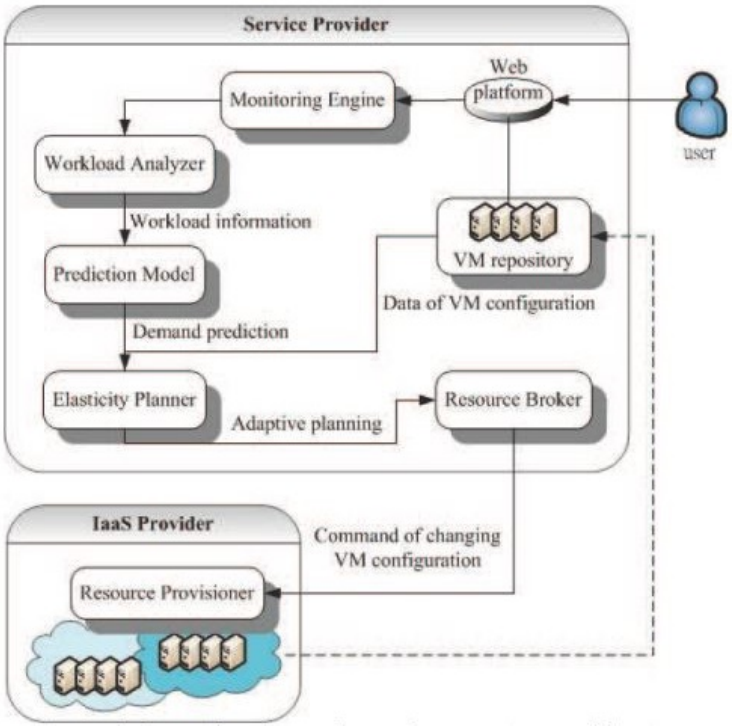


FIGURE 3.4 – Architecture pour l'approvisionnement de l'élasticité [Hwang 2014]

Une solution d'approvisionnement des ressources en termes d'élasticité verticale et horizontale a été présentée dans [Hwang 2014]. Les auteurs ont proposé dans un premier temps un algorithme proactif qui opère en deux phases : la première phase détermine le nombre et le type de machines virtuelles optimal à approvisionner pour répondre à une charge de travail anticipée dans le futur (réservation de ressources à long terme). La deuxième phase fournit des ressources supplémentaires si la capacité des ressources réservées actuellement est insuffisante (allocation à la demande des ressources). Par ailleurs, une architecture est proposée pour les fournisseurs de services cloud. Cette architecture utilise l'algorithme en deux phases pour assurer l'approvisionnement élastique des ressources, tout en optimisant les coûts de fonc-

tionnement. Comme le montre la Figure 3.4, l'architecture proposée est constituée d'un composant Monitoring Engine qui surveille les fluctuations de la charge de travail et le taux d'utilisation des ressources. Un composant Workload Analyser qui utilise les modèles analytiques pour analyser la charge de travail. Un composant Prediction Model qui prédit la charge de travail entrante. Un composant Elasticity Planner qui exécute l'algorithme à deux phases proposé initialement, afin d'assurer une allocation optimale des ressources. Enfin, un composant Resource Broker qui effectue la planification adaptative et la délivrance des ressources au fournisseur IaaS.

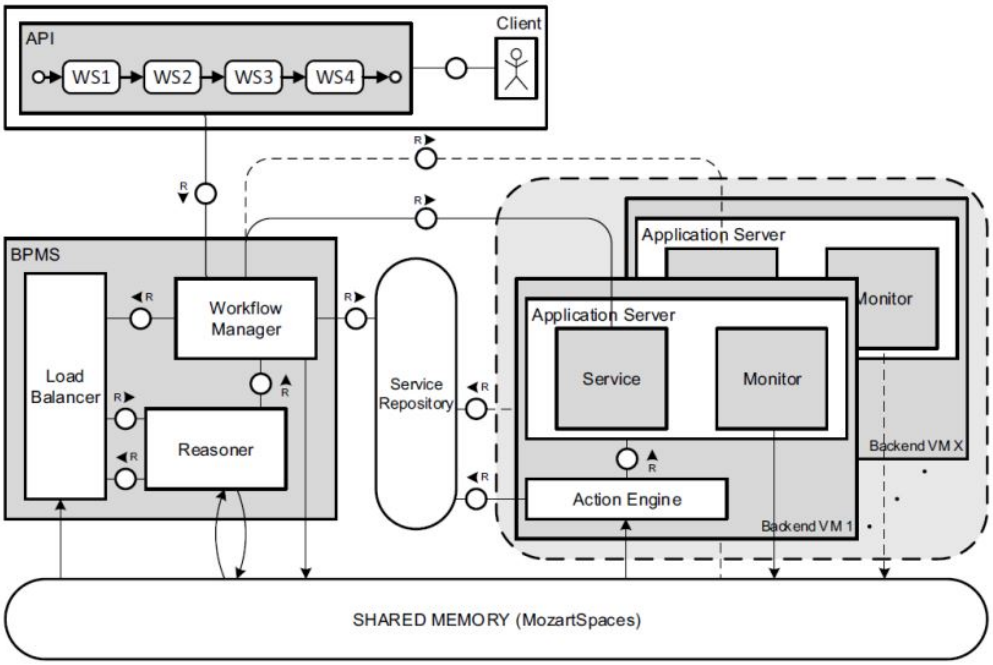


FIGURE 3.5 – Architecture ViePEP [Schulte 2012]

Dans [Ali-Eldin 2012], les auteurs se sont concentrés sur la gestion de l'élasticité horizontale dans les environnements cloud au niveau infrastructure. Pour atteindre cet objectif, ils ont proposé un contrôleur hybride de l'élasticité qui couple entre un contrôleur réactif pour la création de nouvelles instances VM (scale up) et un autre contrôleur proactif pour supprimer les instances des machines virtuelles (scale down). Le contrôleur réactif prend les décisions de créations des machines virtuelles selon la charge de travail actuelle dans le système cloud, alors que le contrôleur proactif se base sur l'histoire de la charge de travail pour prendre les décisions de suppressions. Lorsqu'il y a une contradiction entre les deux contrôleurs, la priorité est donnée au contrôleur réactif. Les états des services cloud sont représentés en utilisant la théorie des files d'attente comme un système à base d'une boucle de contrôle fermée. Dans cette approche, le contrôleur crée/détruit les instances de machines virtuelles afin d'assurer une bonne qualité de service (QoS). Pour valider leur approche, les auteurs

ont comparé et évalué différents scénarios de couplage entre les deux contrôleurs réactifs et proactifs. Les résultats de l'évaluation ont montré que ce couplage est une solution plus efficace que l'utilisation d'une approche purement proactive. Il permet d'assurer l'élasticité et maintenir une qualité de service (QoS) adéquate dans les systèmes basés cloud.

Les auteurs dans [Schulte 2012] ont introduit la plate-forme de Vienne (ViePEP) pour la mise en œuvre des processus élastiques dans les environnements basés cloud. La Figure 3.5 présente l'architecture de ViePEP, qui est constituée d'un ensemble de composants : le composant Workflow Manager pour contrôler l'exécution des workflows, le composant Load Balancer pour équilibrer la charge entre les appels de service et le composant Reasoner pour optimiser l'environnement du processus. ViePEP permet de surveiller l'exécution du processus et de raisonner sur l'optimisation du taux d'utilisation de ressources à travers l'environnement actuel et futur du système. Pour ce faire, la plate-forme peut déclencher une série d'actions d'élasticité, par exemple, la création/suppression d'une instance de machine virtuelle qui accueille un service chargé/sous-utilisé. La plate-forme ViePEP a été étendue dans [Hoenisch 2013a, Hoenisch 2013b]. La première extension consiste en un algorithme de prédiction et de raisonnement qui se base sur les informations de l'environnement actuel et futur du système pour l'exécution du processus élastique. La deuxième extension est un autre algorithme pour la planification d'allocation de ressources basé sur des exigences non fonctionnelles définies par l'utilisateur. Cet algorithme permet d'assurer l'optimisation du taux d'utilisation de ressources.

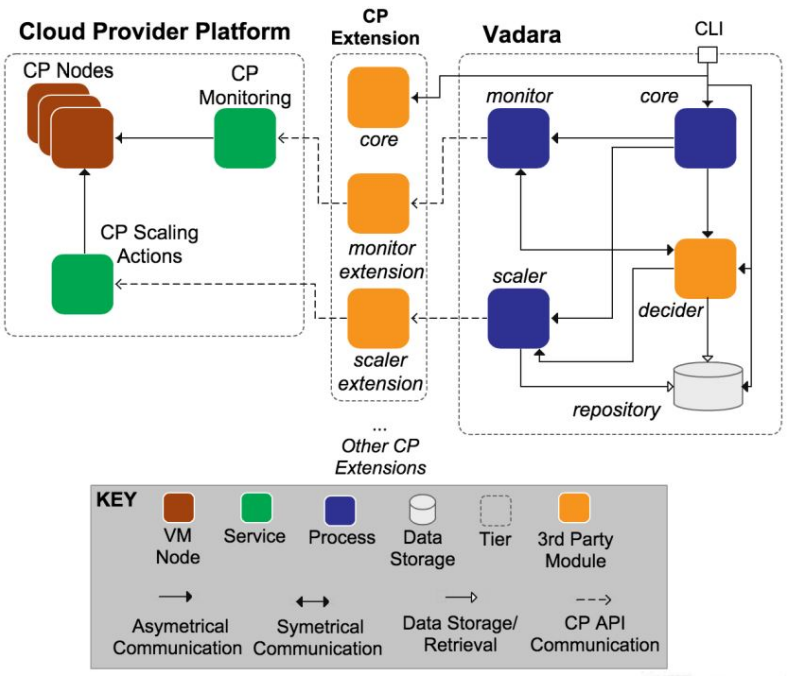


FIGURE 3.6 – Architecture du Framework Vadara [Loff 2014]

Dans [Loff 2014], les auteurs ont proposé Vadara, un Framework générique qui fournit une couche d'abstraction pour les fournisseurs de services cloud existants, permettant l'utilisation et le développement de stratégies d'élasticité de manière unifiée. Le Framework proposé permet de découpler les stratégies d'élasticité des plate-formes cloud sous-jacentes, ce qui rend l'utilisation des stratégies génériques. Pour fournir l'élasticité, les auteurs ont proposé d'ajouter une extension aux plate-formes cloud afin de permettre l'interaction entre les composants du fournisseur (par exemple, suivi et mise à l'échelle des composants) et les composants du Framework. Comme le montre la Figure 3.6, le Framework Vadara met en œuvre un ensemble de composants : un composant Core qui configure et initialise les éléments du Framework. Un composant Monitor qui collecte, regroupe et soumet les informations de suivi à un composant Decider. Le composant Decider analyse les informations de suivi et envoie les actions d'élasticité appropriées pour un composant Scaler. Le composant Scaler fait appel au service de mise à l'échelle du PaaS afin d'exécuter les actions d'élasticité choisies par le composant Decider.

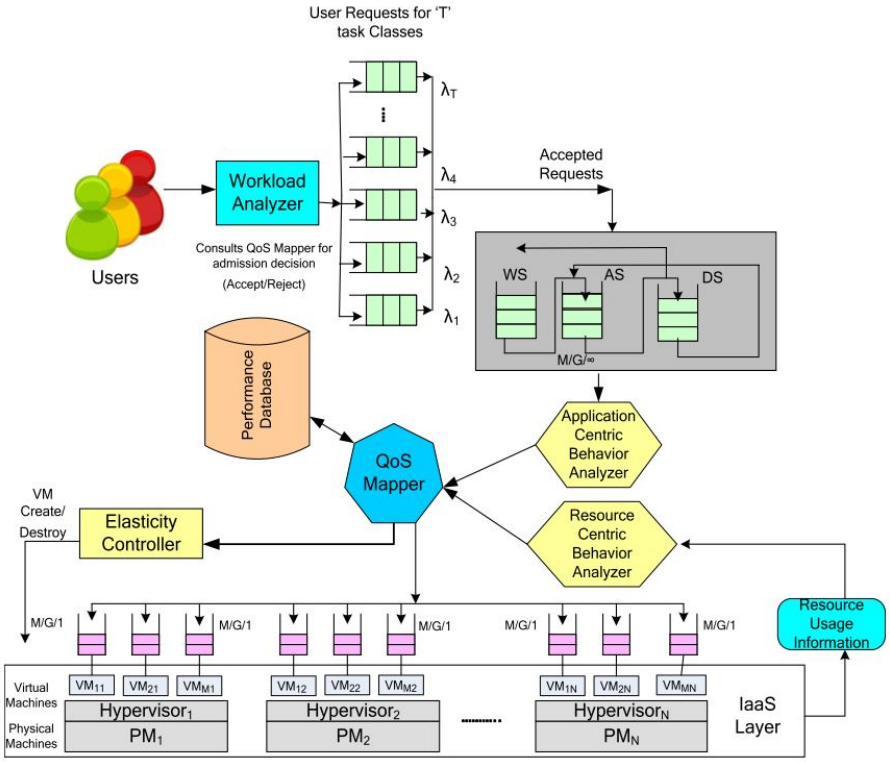


FIGURE 3.7 – Le Framework QoS-Aware Resource Elasticity (QRE) [Kaur 2014]

Les auteurs de [Kaur 2014], ont proposé le Framework QoS-Aware Resource Elasticity (QRE) pour assurer l'élasticité des ressources au niveau application tout en tenant compte des exigences de qualité de service (QoS). L'approche proposée transforme les attributs de qualité de service d'une application (en termes de temps de réponse et taux d'utilisation de ressources) en des attributs de ressources cloud

afin de garantir leur élasticité. Le Framework proposé capture le comportement des applications en utilisant un modèle analytique permettant d'estimer les ressources requises par la charge de travail d'une application donnée afin de garantir la satisfaction des exigences de qualité de service. Comme le montre la Figure 3.7, le Framework QRE comprend un composant Workload Analyzer qui interagit avec les utilisateurs d'une application, et avec un autre composant QoS Mapper pour décider si une requête utilisateur est acceptée ou non. Le composant Application Centric Behavior Analyzer analyse le comportement de l'application afin de prédire la fréquence d'arrivée des différentes tâches par rapport à la variation du taux d'arrivée et la charge de travail. Le composant Resource Centric Behavior Analyzer récupère les informations sur le taux d'utilisation de ressources actuellement dans le système en utilisant le composant Moniteur propre à chaque fournisseur IaaS. Le composant QoS Mapper transforme les exigences de qualité de service de l'application à des allocations de ressources. Pour réaliser cette tâche, le composant QoS Mapper consulte la base de données Performance Database qui contient des informations sur la charge de travail actuelle et les attributs de qualité de service (QoS). Enfin, le composant Elasticity Controller qui reçoit des requêtes du composant QoS Mapper pour l'ajout et la suppression des instances de machines virtuelles sur l'infrastructure cloud.

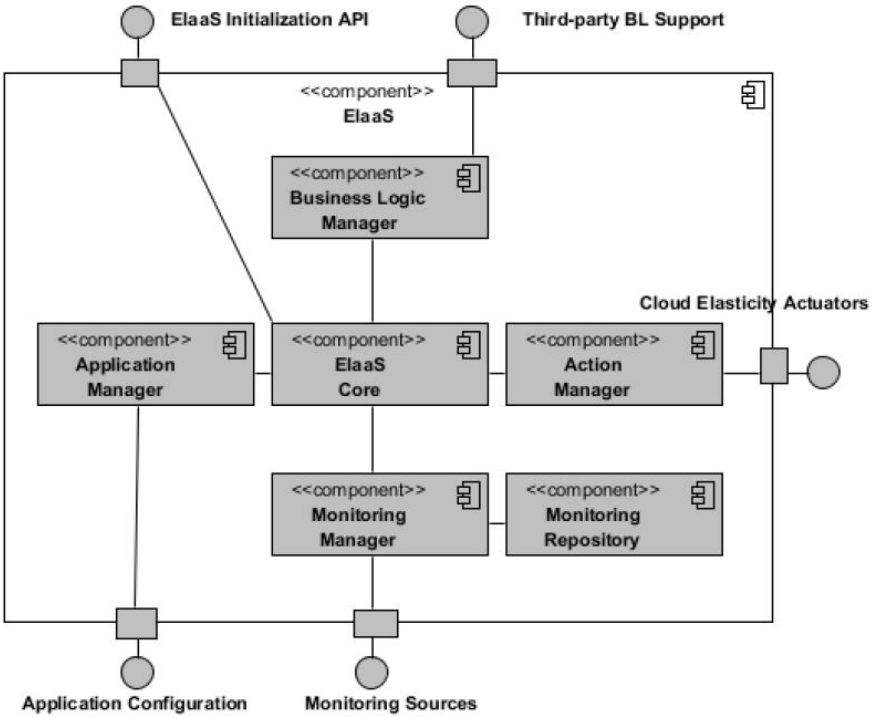


FIGURE 3.8 – Les composants de ElaaS [Kranas 2012]

Un service pour la gestion de l'élasticité dans les systèmes basés cloud appelé ElaaS (elasticity as-a-service) a été présenté dans [Kranas 2012]. ElaaS est implémenté comme une application SaaS qui peut être utilisée dans n'importe quel en-

vironnement basé cloud. ElaaS est constituée d'un nombre de composants comme le montre la Figure 3.8. Le composant Core de ElaaS coordonne les activités et les processus. Le composant Application Manager recueille et analyse les informations relatives à l'utilisateur et l'application. Le composant Monitoring Manager communique avec les sources de surveillance (sondes) qui peuvent être installées dans les différents niveaux cloud (infrastructures, plate-forme ou application). Le composant décideur dans le Framework est nommée Business Logic Manager. Enfin, le composant Action Manager envoie des messages d'action pour les composants appropriés d'une application et/ou plate-forme et produit un nouveau graphe de déploiement selon l'action exécutée. Par conséquent, l'élasticité est assurée selon le graphe de déploiement résultat.

Ces différentes solutions introduites, présentent des mécanismes pour garantir l'élasticité dans les environnements cloud computing. Elles proposent des contrôleurs d'élasticité autonomiques utilisés pour gérer et planifier la consommation des ressources informatiques dans le cloud. Cependant, ces approches ne sont pas génériques et aucune d'entre elles ne permet de gérer l'élasticité complètement. Chacune des solutions décrites, se focalise sur un aspect d'élasticité particulier sans tenir compte des autres aspects. À titre d'exemple, certaines d'entre elles traitent l'élasticité horizontale ou verticale, d'autres gèrent l'élasticité seulement aux niveaux service ou infrastructure, et d'autres se concentrent uniquement sur le concept de stratégies d'élasticité. En outre, l'exactitude d'aucune de ces approches proposées n'a été prouvée puisque elles ne sont pas basées sur des modèles formels.

3.5 Modèles formels pour la spécification et l'analyse de l'élasticité

Il y a eu un nombre important de travaux de recherche dans la littérature qui portent sur la définition de modèles formels pour la spécification et l'analyse des systèmes cloud en général, tels que [Freitas 2012, Rady 2013, Kikuchi 2014, Benzadri 2014]. Toutefois, aucune attention n'a été attribuée dans ces travaux au caractère élastique de ces systèmes. En effet, une fondation conceptuelle solide pour la modélisation et l'analyse du comportement complexe des systèmes élastiques cloud est encore absente. Dans ce contexte, quelques tentatives et approches basées sur des modèles et des cadres formels se penchent actuellement sur la modélisation et l'analyse formelle des aspects comportementaux des systèmes élastiques basés cloud. Dans la suite de cette section nous présentons et nous évaluons un ensemble d'approches formelles récentes basées sur des formalismes différents.

Les auteurs de [Amziani 2013b] ont proposé un Framework formel pour la modélisation de l'élasticité des processus métiers basés service (SBP) dans les environnements cloud computing. Le Framework permet aussi d'évaluer le comportement élastique des processus métiers basés service et les différentes stratégies d'élasticité. Le formalisme adopté par les auteurs était les réseaux de Petri (PN). Dans ce travail, les auteurs mettent en évidence l'importance de la modélisation de l'élasticité au niveau SaaS. Par conséquent, ils se sont concentrés uniquement sur l'élasticité

horizontale au niveau service tout en tenant compte des notions de la charge de travail des services et du temps de réponse. En utilisant ce modèle basé réseaux de Petri, ils expriment deux opérations pour la mise à l'échelle horizontale : la réplication et la consolidation des instances de services. Par ailleurs, ils définissent un contrôleur d'élasticité à base de réseaux de Petri pour assurer l'élasticité des processus métiers basés service. Le contrôleur surveille un processus métier basé service (SBP) et exécute l'action appropriée (réplication/consolidation d'instance de service) pour garantir l'élasticité du SBP, déployé dans une infrastructure ou une plate-forme cloud. Ce contrôleur permet la mise en œuvre et l'exécution des différents types de stratégies d'élasticité. L'approche proposée a été étendue en utilisant les réseaux de Petri colorés (CPN) pour introduire la notion de services à états dans [Amziani 2013c] et la notion de temps dans [Amziani 2013a]. En outre, à base de cette approche et le standard Open Cloud Computing Interface (OCCI), un modèle pour l'approvisionnement autonomique des ressources dans le cloud a été proposé dans [Mohamed 2015]. Le modèle gère l'élasticité en couvrant les fonctions de la boucle de contrôle autonomique MAPE-K.

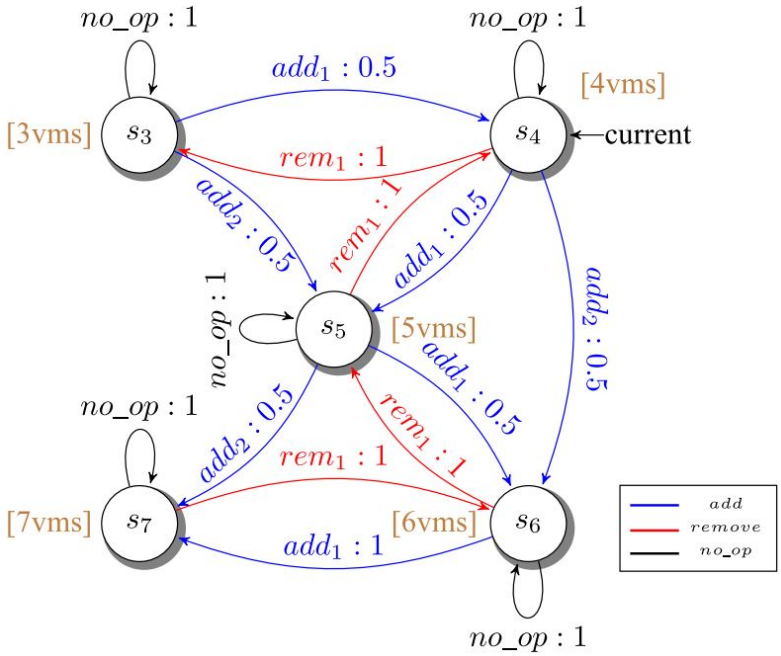


FIGURE 3.9 – Vue générale sur le modèle MDP [Naskos 2014]

Dans [Naskos 2014], les auteurs ont proposé un modèle formel pour l'analyse quantitative d'élasticité dans le niveau infrastructure. En particulier, ils ont proposé d'utiliser les processus de décision markovien (MDP) pour modéliser les actions d'élasticité horizontale et les stratégies d'élasticité proactives au niveau infrastructure. D'une part, les différentes actions possibles d'élasticité sont représentées comme un aspect non déterministe. D'autre part, les effets d'actions d'élasticité sont représentés comme un aspect probabiliste. Comme le montre le modèle MDP dans

la Figure 3.9, chaque état représente une taille de cluster différente (nombre de machines virtuelles). Les transitions entre les états du modèle représentent des actions d'élasticité (mise à l'échelle du nombre de machines virtuelles). Une transition démarre à partir d'un état initial et aboutit à un état qui correspond à l'exécution d'une action d'élasticité (par exemple, l'état après l'ajout d'une nouvelle machine virtuelle). Ces transitions sont transformées en probabilités, chacune représente la possibilité d'exécution d'une action d'élasticité spécifique. À base des modelés MDP, l'approche propose d'utiliser une vérification continue en ligne avant d'exécuter des actions d'élasticité. L'approche commence par l'instanciation d'un modèle en fonction des conditions de la charge de travail et de l'environnement cloud. Ensuite, ce modèle est vérifié en ligne afin d'exécuter des actions d'élasticité. L'approche proposée permet d'effectuer des évaluations des modèles et des stratégies (politiques) d'élasticité différentes en termes de taux d'utilisation de ressources afin d'éviter le sous-approvisionnement et sur-approvisionnement.

Ces deux approches proposent des modèles formels pour la spécification de l'élasticité horizontale dans les environnements cloud computing aux niveaux service [Amziani 2013b] et infrastructure [Naskos 2014]. Ils couvrent aussi la modélisation des stratégies d'élasticité proactives dans [Naskos 2014] et les deux types de stratégies (proactives et réactives) dans [Amziani 2013b]. Toutefois, les modèles proposés sont très simples et ne sont pas génériques pour représenter tous les aspects relatifs aux systèmes élastiques basés cloud. Les deux approches se sont limitées à la méthode de mise à l'échelle horizontale à un niveau cloud spécifique. En outre, les détails architecturaux des systèmes basés cloud sont négligés dans [Amziani 2013b] et partiellement supportés dans [Naskos 2014].

La contribution dans [Dustdar 2015] vise à proposer une méthodologie systématique pour développer des systèmes élastiques basés cloud. L'objectif principal dans ce travail est de présenter une approche formelle qui permet aux fournisseurs d'applications cloud de caractériser précisément les comportements souhaités de leurs systèmes élastiques basés cloud. Pour réaliser cet objectif, les auteurs ont proposé une formalisation des aspects comportementaux relatifs à ces systèmes. La formalisation présentée est basée sur des expressions régulières temporisées (TRE). Les auteurs estiment que les expressions régulières temporisées représentent un formalisme très approprié pour exprimer le comportement élastique des systèmes basés cloud. Cette formalisation permet au fournisseur de services cloud d'exprimer de façon naturelle certains aspects pertinents propres au comportement de ce type de système tels que les fluctuations de la charge de travail, les performances du système et l'allocation des ressources sous la forme de patterns temporisés. Elle leur permet aussi d'exprimer les comportements désirables et indésirables des systèmes élastiques basés cloud en termes d'occurrence de tels patterns pendant une période d'observation. En outre, les auteurs proposent une simple formalisation des contrôleurs d'élasticité réactives à base de seuils.

Une autre contribution similaire à celle proposée ci-dessus est présentée dans [Bersani 2014]. Les auteurs de ce travail proposent une approche de modélisation des concepts et des propriétés relatives au comportement des systèmes élastiques ba-

sés cloud en utilisant un formalisme différent appelé CLTLt (d) (Timed Constraint LTL). Plus particulièrement, ils proposent d'utiliser la logique temporelle CLTLt (d) pour caractériser des propriétés inhérentes à l'élasticité, la gestion de ressources et la qualité de service (QoS) dans les systèmes basés cloud. L'utilisation de la logique temporelle ouvre la voie à l'utilisation des outils de vérification pour analyser si les propriétés proposées, qui sont des aspects spécifiques au comportement élastique, sont vérifiées ou non lors de l'exécution des systèmes élastiques basés cloud. En outre, les auteurs ont effectué une évaluation préliminaire pour déterminer la faisabilité de vérification des propriétés proposées, exprimées en CLTLt (d) sur les traces d'exécution d'une application en utilisant ZOT, un outil de vérification basé sur les solveurs SAT et SMT.

Dans ces deux dernières approches, les auteurs ont essayé de présenter des formalisations qui permettent de spécifier le comportement élastique des systèmes basés cloud d'une manière précise. Cependant, les modèles formels proposés sont très simples pour être utilisés dans le monde réel. Les deux approches considèrent uniquement les opérations d'élasticité horizontale par la création et la suppression des instances de machines virtuelles (niveau infrastructure) selon des stratégies réactives. Aucune autre méthode d'élasticité ou niveau cloud, prenant en compte des aspects architecturaux détaillés n'a été prise en compte par les modèles proposés.

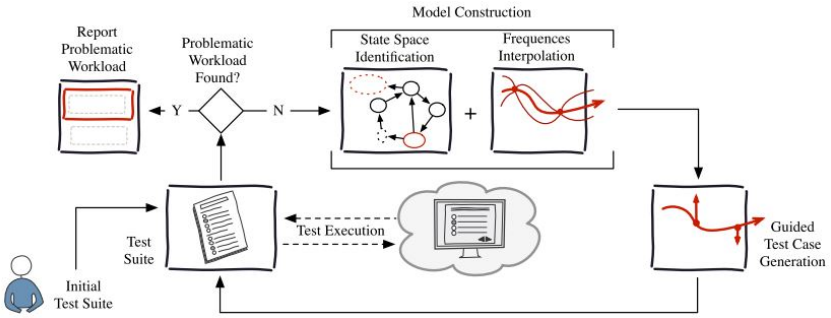


FIGURE 3.10 – Aperçu le Framework basé modèle de génération de tests [Gambi 2013a]

Les auteurs dans [Gambi 2013a] ont défini un Framework formel basé sur un modèle de génération de tests pour l'analyse des propriétés d'élasticité. Pour fournir l'élasticité dans les systèmes élastiques basés cloud, les auteurs dans ce travail visent à faire face à la complexité des interactions qui se produisent dans ces systèmes telles que les interactions entre l'infrastructure cloud, les applications déployées, la logique de contrôle et les conditions environnementales externes. Ils estiment que c'est difficile de garantir l'élasticité dans les systèmes basés cloud à cause de l'absence de directives génériques et applicables pour modéliser et tester ce genre de systèmes. La Figure 3.10 présente un aperçu sur le Framework proposé. L'approche proposée consiste en un processus qui commence par une suite de tests initiale (ensemble de cas de tests) fournie par un concepteur ou générée aléatoirement. Cette suite de

tests contient un ensemble de profils de charge de travail, dont chacun peut être défini comme une instanciation d'une classe paramétrique des charges de travail. Une classe paramétrique est définie comme une fonction paramétrique temporisée. La suite de tests initiale est testée sur une instance d'une application déployée dans un environnement cloud. Si la suite conduit à la violation d'une propriété d'élasticité, le profil de la charge de travail correspondante est marqué comme problématique et signalé au concepteur afin de conduire le diagnostic, sinon, la suite de tests subit une tentative de raffinement. La première étape pour raffiner la suite de tests est la construction d'un modèle qui capture le comportement élastique du système. Ce modèle est un système de transitions étiquetées (LTS) où les états capturent les ressources allouées et les étiquettes décrivent la fréquence d'occurrence de transitions durant l'exécution de la suite de tests. L'état initial LTS représente la configuration initiale du système basé cloud. À base du modèle LTS construit, la suite de tests peut être raffinée afin d'identifier d'autres profils de charge de travail problématiques. Une autre approche de formalisation connexe à la génération automatiquement des cas de tests pour les systèmes élastiques basés cloud a été présentée dans [Gambi 2013b]. Cette approche repose sur un simple modèle pour la description du comportement élastique des systèmes cloud, construit en utilisant des modèles de substitution (SM). Le modèle permet de générer des suites de tests qui sont utilisées plus tard pour exposer les erreurs fonctionnelles et les problèmes de performance liés à l'élasticité dans le cloud. Dans ces deux approches, l'élasticité est traitée d'une manière générale sans adresser spécifiquement une des méthodes d'élasticité. En outre, les auteurs dans ces travaux se sont concentrés sur le niveau infrastructure seulement et ont négligé complètement les aspects structurels des systèmes basés cloud ainsi que la représentation des stratégies d'élasticité afin de simplifier leur formalisation. Par conséquent, le comportement interne de ce genre de systèmes en termes d'interactions entre les différents composants cloud ainsi que les changements architecturaux (shape-shifting) ne sont pas pris en charge par les deux approches.

Synthèse

Nous avons jugé important d'énumérer un ensemble de caractéristiques qu'un modèle ou une approche formelle pour la spécification des systèmes élastiques basés cloud et leur comportement doit supporter afin de cerner la problématique de notre travail, à savoir le niveau de prise en charge de détails architecturaux, la technique d'analyse adoptée, les stratégies d'élasticité, les méthodes d'élasticité et les niveaux du modèle cloud couverts. Nous veillons donc à identifier ces critères récapitulés dans le Tableau 3.1 pour permettre d'évaluer les différentes approches mentionnées ci-avant et déterminer leurs carences et lacunes.

À notre connaissance, actuellement dans la littérature, aucune des approches formelles existantes y compris celles présentées dans cette section, ne fournit une méthodologie générique et exhaustive pour la modélisation et l'analyse du comportement élastique des systèmes basés cloud. Généralement, les chercheurs dans ce domaine se concentrent principalement sur le niveau infrastructure du modèle

cloud et la mise à l'échelle horizontale sachant que c'est la méthode la plus couramment adoptée par les fournisseurs de services (voir Tableau 3.1). Par exemple, les approches dans [Naskos 2014, Dustdar 2015, Bersani 2014] proposent différents modèles formels pour l'élasticité horizontale dans le cloud mais seulement au niveau infrastructure. Les modèles formels dans [Gambi 2013a] et [Gambi 2013b] ont traité également l'élasticité au niveau infrastructure mais sans tenir compte d'une méthode d'élasticité particulière. L'exception a été faite par les contributions de [Amziani 2013b, Amziani 2013c, Amziani 2013a] qui capturent la méthode de la mise à l'échelle horizontale au niveau application uniquement. La spécification formelle de l'élasticité aux niveaux plate-forme et répartition de charge, ainsi que la méthode d'élasticité verticale et la méthode de migration n'ont été prises en charge par aucune de ces contributions.

Approche		[Amziani 2013b] [Amziani 2013c] [Amziani 2013a]	[Naskos 2014]	[Dustdar 2015]	[Bersani 2014]	[Gambi 2013a] [Gambi 2013b]
Stratégie	Proactive	✓	✓	–	–	–
	Réactive	✓	–	✓	✓	–
Méthode	Migration	–	–	–	–	–
	Horizontale	✓	✓	✓	✓	–
	Verticale	–	–	–	–	–
Niveau cloud	Infrastructure	–	✓	✓	✓	✓
	Plate-forme	–	–	–	–	–
	Application	✓	–	–	–	–
	Répartition de charge	–	–	–	–	–
Niveau de détails	Éléments architecturaux	–	Infrastructure	–	–	–
	Interactions internes	–	–	–	–	–
Formalisme		PN/CPN	MDP	TRE	CLTLt (d)	LTS/SM
Technique d'analyse		Vérification de preuve	Vérification formelle	–	Solveurs SAT/SMT	–

TABLE 3.1 – Modèles formels pour la spécification et l'analyse de l'élasticité

Concernant les stratégies d'élasticité, les auteurs dans [Dustdar 2015] et [Bersani 2014] ont pris en compte la modélisation des stratégies réactives, par contre [Naskos 2014] s'est concentré uniquement sur les stratégies proactives (voir Tableau 3.1). En outre, les travaux dans [Amziani 2013b, Amziani 2013c, Amziani 2013a] ont traité les deux types de stratégie proactive et réactive. Enfin, les auteurs dans [Gambi 2013a, Gambi 2013b] n'ont pas tenu compte de l'aspect stratégique de l'élasticité dans leurs modèles. Par ailleurs, le comportement élastique dans les modèles présentés est considéré d'un très haut niveau d'abstraction, tout en ignorant les détails conceptuels et architecturaux des systèmes élastiques basés cloud. À l'exception de [Naskos 2014] qui considère quelques détails mineurs, quasiment tous les modèles et les approches formelles présentées pour la description de l'élasticité dans les environnements cloud computing ne prennent pas en charge la modélisation des éléments architecturaux et les interactions internes dans les systèmes basés cloud. Nous constatons que ces approches ont négligé beaucoup d'aspects importants dans le processus de spécification en essayant de simplifier la tâche de l'analyse de l'élasticité dans les systèmes basés cloud tels que : les aspects structuraux des architectures cloud, les interactions client/application dans le cloud et les différentes relations entre les composants cloud. Cela rend ces approches difficiles à utiliser dans le monde réel.

Malgré ces quelques tentatives pour la formalisation des aspects structurels et comportementaux de ces systèmes, aucune des approches proposées ne supporte l'ensemble des caractéristiques que nous avons définies. Le Tableau 3.1 récapitule les diverses caractéristiques remplies par les approches que nous avons étudiées.

3.6 Conclusion

Dans la première partie de ce chapitre, nous avons introduit les concepts de base relatifs à notre travail. Nous avons présenté tout d'abord l'informatique autonomique et la boucle de contrôle autonomique MAPE-K comme un modèle pour la gestion de l'élasticité dans les environnements cloud computing. Ensuite, nous avons présenté le principe d'élasticité dans le cloud à travers sa définition et particulièrement le quadruplet qui la spécifie : niveau, stratégie, objectif et méthode. Dans la deuxième partie de ce chapitre, nous avons donné un aperçu des travaux existants liés aux différents aspects de la gestion autonomique de l'élasticité dans le cloud. À cette fin, nous avons présenté un nombre de solutions de recherche proposées pour garantir l'élasticité dans les environnements cloud computing. Ensuite, nous avons présenté les tentatives et les approches qui proposent des modèles basés sur des formalismes différents pour la modélisation et l'analyse formelle des systèmes élastiques basés cloud et leur comportement. Enfin, nous avons proposé une évaluation de ces travaux existants dans la littérature selon un ensemble de critères que nous avons jugés utiles pour situer notre contribution.

Chapitre 4

Fondements formels

Sommaire

4.1 Introduction 46
4.2 Les systèmes réactifs bigraphiques 47
4.2.1 Anatomie des bigraphes et forme graphique 47
4.2.2 Notation 49
4.2.3 Définitions formelles 49
4.2.4 Opérations sur les bigraphes 51
4.2.5 Forme algébrique 54
4.2.6 Logique de typage 56
4.2.7 Aspect dynamique des bigraphes 57
4.2.8 Logique spatiale BiLog 59
4.2.9 Outils autour des BRS 59
4.3 Langage Maude 60
4.3.1 Syntaxe et notations 61
4.3.2 Langage de stratégies 63
4.3.3 Analyse formelle dans Maude 65
4.4 Conclusion 67

4.1 Introduction

Avec le développement des exigences du marché et l'apparition de nouvelles technologies, la conception des systèmes informatiques est devenue de plus en plus difficile à maîtriser. Une des solutions proposées pour spécifier et analyser ces systèmes complexes consiste à recourir aux méthodes formelles qui offrent des mécanismes d'abstraction non ambigus, une rigueur et une précision dans la spécification des aspects structurels et comportementaux de ces systèmes.

Les méthodes formelles se basent généralement sur des concepts mathématiques rigoureux permettent de spécifier et développer les systèmes informatiques. Par ailleurs, contrairement aux méthodes traditionnelles de tests, qui ne peuvent pas traiter de manière exhaustive tous les cas possibles d'utilisation d'un système, les méthodes formelles permettent aux développeurs de décrire et vérifier les propriétés d'un système informatique d'une manière systématique.

Dans notre travail, nous nous sommes intéressés à deux formalismes pour la modélisation et la vérification des systèmes élastiques basé cloud, à savoir, les systèmes

réactifs bigraphiques et le langage formel Maude. Les systèmes réactifs bigraphiques (BRS) sont un nouveau formalisme qui diffère des formalismes traditionnels tels que la notation Z, les réseaux de Petri, la méthode B et l'algèbre de processus par son aspect graphique et sa capacité de représenter à la fois la localité et la connectivité des systèmes informatiques ubiquitaires et distribués. Néanmoins, les outils développés autour des BRS sont limités en termes d'expressivité et de performance. Pour cela, nous avons aussi opté pour l'utilisation du langage de spécification formelle Maude comme étant l'alternative la plus adéquate.

Ce chapitre est constitué de deux parties. La première est consacrée à la présentation des différentes définitions, concepts et outils des systèmes réactifs bigraphiques. Alors que dans la deuxième partie, nous présenterons le langage formel Maude. Précisément, nous décrivons sa syntaxe et notations, son langage de stratégies et ses techniques d'analyses formelles. Les définitions données dans ce chapitre sont tirées de [Conforti 2006, Milner 2006, Milner 2008, Milner 2009] pour les systèmes réactifs bigraphiques et de [Clavel 2007, Eker 2007, Martí-Oliet 2009, Clavel 2011] pour le langage Maude.

4.2 Les systèmes réactifs bigraphiques

Les systèmes réactifs bigraphiques (BRS) représentent un formalisme émergeant pour la modélisation de l'évolution temporelle et spatiale du calcul. La théorie de bigraphes a été initialement introduite par Robin Milner [Milner 2008] afin de fournir un modèle intuitif entièrement graphique capable de représenter à la fois la connectivité et la localité des systèmes ubiquitaires. Un système réactif bigraphique est constitué d'un ensemble de bigraphes représentant l'état du système et un ensemble de règles de réaction définissant la façon dont le système peut évoluer. Dans ce qui suit, nous présentons tout d'abord les bigraphes d'une manière informelle avant d'introduire la définition formelle de ce modèle.

4.2.1 Anatomie des bigraphes et forme graphique

Considérons un exemple d'un bigraphe B montré dans la Figure 4.1a ci-dessous. Dans la forme graphique des bigraphes, les entités et les composants (réels ou virtuels) d'un système sont exprimés par des nœuds, représentés comme des ovales, cercles, triangles et autres formes graphiques. L'emplacement spatial des nœuds est décrit en termes d'imbrications arbitraires entre les différents nœuds d'un système donné. Tous les nœuds dans un bigraphe possèdent un identifiant (type), appelé contrôle et désigné par des lettres (ex. M, L, etc.). L'ensemble de contrôles d'un bigraphe est appelé la signature du bigraphe. Un nœud est dit atomique s'il ne peut pas contenir d'autres nœuds. Les nœuds non atomiques peuvent être soit actifs soit passifs. Les interactions entre les différents nœuds sont représentées par des liens, par exemple, l'hyper-arc dans la Figure 4.1a reliant le nœud L et le nœud M. Chaque nœud peut avoir zéro, un ou plusieurs ports, représentés par des points ronds pour exprimer des connexions possibles. Ces dernières peuvent être considérées comme

des sockets de communication dans lesquelles les liens peuvent être branchés. Nous notons que les nœuds qui ont le même contrôle ont aussi le même nombre de ports. Les rectangles en pointillés indiquent des régions (parfois appelées racines), leur rôle est de décrire les parties du système qui ne sont pas nécessairement adjacentes. Les carrés en gris représentent des sites, ils sont des parties abstraites du modèle. Les régions et les sites sont indexés par des nombres naturels (à partir de 0) de gauche à droite. Les nœuds, les sites et les régions sont appelés les places d'un bigraphe. En plus des hyper-arcs, un bigraphe peut avoir d'autres types de liens de communications qui sont les noms internes et les noms externes. Dans notre exemple (voir Figure 4.1a), y_0, y_1 et y_2, sont des noms externes, alors que x_0 et x_1 représentent des noms internes. Ils expriment des liens (potentiels) à d'autres bigraphes, représentant des environnements externes.

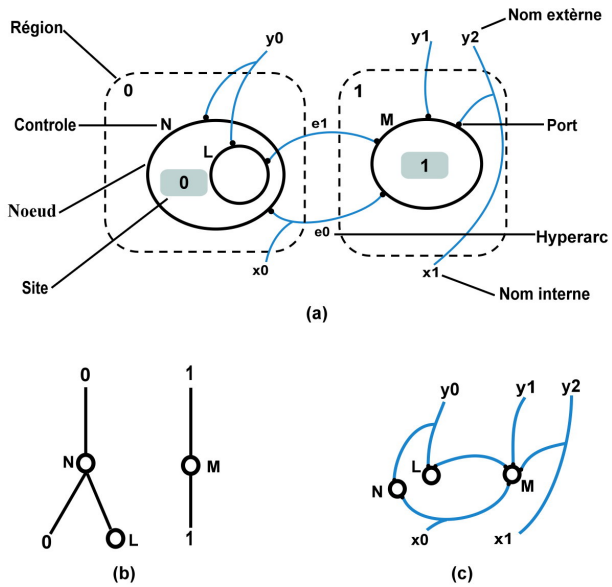


FIGURE 4.1 – Anatomie des bigraphes

Les capacités d'un bigraphe d'interagir avec l'environnement extérieur sont enregistrées dans son interface. Par exemple, nous écrivons $B : \langle 2, \{x_0, x_1\}\rangle \rightarrow \langle 2, \{y_0, y_1, y_2\}\rangle$ pour indiquer que le bigraphe B a deux sites, deux régions et deux ensembles de noms externes ($\{y0, y1, y2\}$) et noms internes ($\{x0, x1\}$). La paire $\langle 2, \{x_0, x_1\}\rangle$ est appelée l'interface interne de B, alors que $\langle 2, \{y_0, y_1, y_2\}\rangle$ représente son interface externe.

En résumé, la localité dans un bigraphe est représentée par la distribution spatiale des nœuds, alors que la connectivité est exprimée par des liens. Cette caractéristique peut être rendue explicite en définissant les bigraphes en termes de deux graphes distincts. En effet, un bigraphe peut être vu comme une combinaison de deux structures indépendantes (voir Figure 4.1) : le graphe de places et le graphe de liens [Milner 2008]. L'intersection entre ces deux graphes est un ensemble commun de nœuds, correspondant aux entités réelles ou virtuelles du bigraphe. Le graphe

de places (voir Figure 4.1b) a la structure d'une forêt, représentant la distribution spatiale des différentes entités. Le graphe de liens (voir Figure 4.1c) est un hypergraphe montrant le schéma de connectivité des différents nœuds. Alors qu'un arc dans le graphe de places montre la relation d'imbrication entre les éléments de l'application, un hyper-arc dans le graphe de liens établit une connexion entre les ports de ces éléments. Chaque arbre dans le graphe de places représente une région qui peut contenir des nœuds et des sites, correspondant aux feuilles de l'arbre.

4.2.2 Notation

Nous interprétons souvent un nombre naturel comme un ordinal fini $m = \{0, 1, ..., m-1\}$. Nous écrivons $S \uplus T$ pour indiquer l'union des ensembles reconnus ou supposés être disjoints. Si une fonction f comporte un domaine S et $S' \subseteq S$, alors $f \restriction S$ représente la restriction de f vers S'. Pour deux fonctions f et g ayant des domaines disjoints S et T, nous écrivons $f \uplus g$ pour la fonction ayant le domaine $S \uplus T$ de telle manière que $(f \uplus g) \restriction S = f$ et $(f \uplus g) \restriction T = g$. Nous écrivons Id_s pour exprimer la fonction de l'identité sur l'ensemble S.

Dans la définition des bigraphes, nous supposons que les noms internes et externes, les identifiants des nœuds et des hyper-arcs sont tirés de trois ensembles finis, respectivement X, V et E, disjoints les uns des autres. On note les identifiants par des lettres minuscules : x, y, z pour les noms internes et externes, v, u pour les nœuds, et $e_0, e_1, ...$ pour les hyper-arcs. Les lettres majuscules $A, B, ...$ sont utilisées pour désigner les bigraphes et leurs constituants. Nous appelons l'interface triviale $\varepsilon \stackrel{\text{def}}{=} \langle 0, \emptyset \rangle$ l'origine.

4.2.3 Définitions formelles

Comme indiqué précédemment, un bigraphe est composé de deux structures orthogonales : le graphe de places qui définit les notions de localité ou de confinement entre les différentes entités, et le graphe de liens qui capture la connectivité des entités du bigraphe, ignorant leur imbrication. Nous commençons par la définition formelle de la signature d'un bigraphe, ensuite nous fournissons les définitions respectives d'un bigraphe, d'un graphe de places et d'un graphe de liens [Milner 2009].

Définition 4.1 (signature). La signature d'un bigraphe B prend la forme $(\mathcal{K}, ar)$. Elle comporte un ensemble $\mathcal{K}$ dont les éléments sont des contrôles et une fonction de transformation $ar : \mathcal{K} \to \mathbb{N}$, qui associe une arité à chaque contrôle. La signature $\mathcal{K}$ assigne à chaque nœud d'un bigraphe B un contrôle, dont les arités indiquent le nombre de ports des nœuds. Une signature pour le bigraphe B de l'exemple présenté dans la Figure 4.1a est donnée par $\mathcal{K} = \{N : 2, L : 2, M : 4\}$.

Définition 4.2 (bigraphe). Formellement, un bigraphe prend la forme :

$$G = (V_G, E_G, ctrl_G, G^P, G^L) : I \to J$$

- V_G est un ensemble fini de nœuds qui peuvent être imbriqués.
- E_G est un ensemble fini d'hyper-arcs.
- $ctrl_G : V_G \to \mathcal{K}$ est une fonction de transformation qui associe à chaque nœud $v_i \in V_G$ un contrôle $k \in \mathcal{K}$ indiquant le nombre de ports. La signature $\mathcal{K}$ est un ensemble fini de contrôles.
- $G^P = (V_G, ctrl_G, prnt_G) : m \to n$ est le graphe de places associé à G. m et n représentent respectivement le nombre de sites et de régions. $prnt_G : m \uplus V_G \to V_G \uplus n$ est une fonction de parenté qui associe à chaque nœud ou site son parent hiérarchique.
- $G^L = (V_G, E_G, ctrl_G, link_G) : X \to Y$ est le graphe de liens de G, où $link : X \uplus P \to E_G \uplus Y$ est une fonction de transformation. X, Y et P représente respectivement l'ensemble de noms internes, l'ensemble de noms externes et l'ensemble de ports de G.
- $I = \langle m, X \rangle$ et $J = \langle n, Y \rangle$ représentent respectivement les interfaces internes et externes du bigraphe G. Ils représentent la capacité du bigraphe G d'interagir avec l'environnement extérieur.

Définition 4.3 (graphe de places). Un graphe de places est défini formellement par :

$$G^P = (V_G, ctrl_G, prnt_G) : m \to n$$

- V_G est un ensemble fini de nœuds.
- $ctrl_G : V_G \to \mathcal{K}$ est une fonction de transformation associant un contrôle à chaque nœud du bigraphe G.
- $prnt_G : m \uplus V_G \to V_G \uplus n$ est une fonction de parenté associant à chaque nœud ou site son parent hiérarchique. La notation $m \uplus V_G$ est utilisée pour indiquer que les deux ensembles sont conservés disjoints.
- m et n représentent respectivement le nombre de sites et de régions du bigraphe G. La notation $m \to n$ permet d'enregistrer les interfaces du graphe de places, où m et n représentent respectivement les interfaces internes et externes du graphe de places.

Définition 4.4 (graphe de liens). Un graphe de liens est défini formellement par :

$$G^L = (V_G, E_G, ctrl_G, link_G) : X \to Y$$

- V_G est un ensemble fini de nœuds.
- E_G est un ensemble fini d'hyper-arcs.
- $ctrl_G : V_G \to \mathcal{K}$ est une fonction de contrôle.
- $link : X \uplus P \to E_G \uplus Y$ est une fonction de transformation montrant le flux de données des noms internes X ou des ports P vers les noms externes Y ou les hyper-arcs E.

4.2.4 Opérations sur les bigraphes

Des bigraphes complexes peuvent être construits à partir des bigraphes élémentaires en appliquant les opérations de composition et de produit tensoriel [Milner 2008]. Dans cette section, nous fournissons les définitions formelles de ces deux opérations bigraphiques et nous expliquons également leur principe à travers des exemples.

Algorithmiquement, la composition peut être considérée comme le placement d'un bigraphe B dans un contexte représenté par un autre bigraphe G. La composition de deux bigraphes B et G peut être réalisée si et seulement si l'interface externe de B correspond à l'interface interne de G. Nous écrivons $G \circ B$ pour indiquer la composition de ces bigraphes via l'hébergement des régions du bigraphe B dans les sites du bigraphe G, et la fusion des noms externes de B avec les liens de G qui ont des noms internes correspondants. Un exemple de composition est montré dans la Figure 4.2. Le bigraphe A (Figure 4.2c) représente le résultat de la composition de deux bigraphes G (Figure 4.2a) et B (Figure 4.2.b). Précisément, la région 0 dans le bigraphe B contenant le nœud C est fusionnée avec le site à l'intérieur du nœud D dans le bigraphe G. En outre, le nœud D est lié sur le nom commun y aux nœuds de contrôle A et B. Nous appelons le bigraphe G le contexte ou l'environnement du bigraphe B.

Dans la suite de cette sous-section, nous présentons la définition formelle de l'opération de composition de deux bigraphes. Pour cela, nous définissons tout d'abord la composition au niveau du graphe de places et graphe de liens, avant de présenter la composition au niveau du bigraphe.

Définition 4.5 (composition des graphes de places). Si $F : k \to m$ et $G : m \to n$ sont deux graphes de places avec des supports disjoints, leur composite $G \circ F = (V, ctrl, prnt) : k \to n$ à l'ensemble de nœuds $V = V_F \uplus V_G$ et la fonction contrôle $ctrl = ctrl_F \uplus ctrl_G$. Sa fonction de parenté $prnt$ est définie par : Si $w \in k \uplus V$ est un site ou un nœud dans $G \circ F$ alors

$$prnt(w) \stackrel{\text{def}}{=} \begin{cases} prnt_F(w) & \text{si } w \in k \uplus V_F \text{ et } prnt_F(w) \in V_F, \\ prnt_G(j) & \text{si } w \in k \uplus V_F \text{ et } prnt_F(w) = j \in m, \\ prnt_G(w) & \text{si } w \in V_G. \end{cases}$$

Définition 4.6 (composition des graphes de liens). Si $F : X \to Y$ et $G : Y \to Z$ sont deux graphes de liens avec des supports disjoints, leur composite $G \circ F = (V, E, ctrl, link) : X \to Z$ à un ensemble de nœuds $V = V_F \uplus V_G$, un ensemble d'hyper-arcs $E = E_F \uplus E_G$ et une fonction de contrôle $ctrl = ctrl_F \uplus ctrl_G$. Sa fonction $link$ est définie par : Si $q \in X \uplus P_F \uplus P_G$ est un point de $G \circ F$ alors

$$link(q) \stackrel{\text{def}}{=} \begin{cases} link_F(q) & \text{si } q \in X \uplus P_F \text{ et } link_F(q) \in E_F, \\ link_G(y) & \text{si } q \in X \uplus P_F \text{ et } link_F(w) = y \in Y, \\ link_G(q) & \text{si } q \in P_G. \end{cases}$$

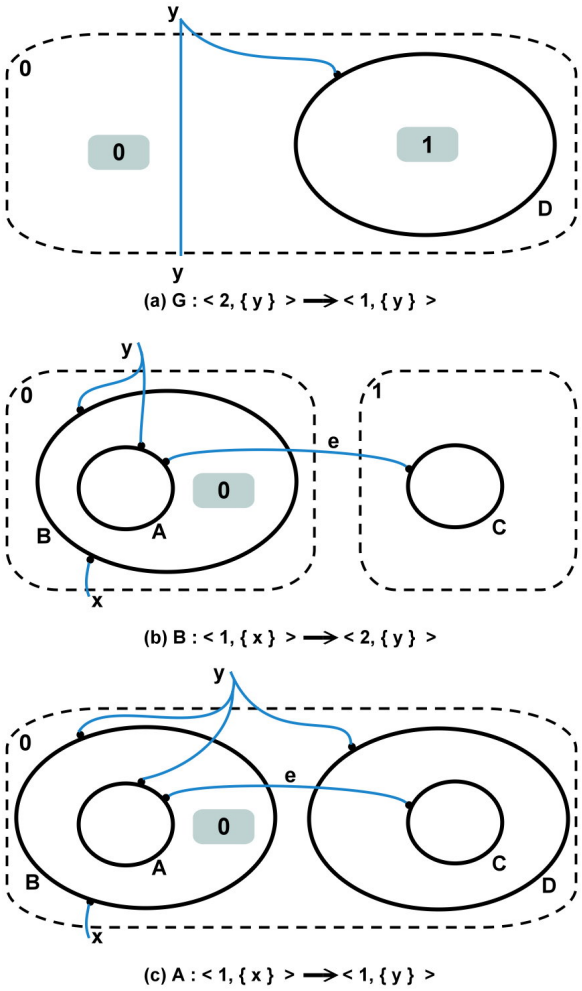


FIGURE 4.2 – Composition des bigraphes $A = G \circ B$

Définition 4.7 (composition des bigraphes). Si $F : \langle k, X\rangle \to \langle m, Y\rangle$ et $G : \langle m, Y\rangle \to \langle n, Z\rangle$ sont deux bigraphes avec des supports disjoints, leur composite est donné par : $G \circ F \stackrel{\text{def}}{=} (G^P \circ F^P, G^L \circ F^L) : \langle k, X\rangle \to \langle n, Z\rangle$

L'autre opération fondamentale pour la composition des bigraphes est le produit tensoriel ou la composition horizontale, notée par $\otimes$. Le produit tensoriel consiste simplement à juxtaposer les régions des bigraphes, soumis à la composition en joignant les liens ouverts communs. Plus précisément, on dit que deux bigraphes $F_i : \langle m_i, X_i\rangle \to \langle n_i, Y_i\rangle$, avec $(i = 0, 1)$ ayant des interfaces disjointes si $X_0 \cap X_1 = \emptyset$ et $Y_0 \cap Y_1 = \emptyset$. Le produit tensoriel sur les interfaces $I = \langle m, X\rangle$ et $J = \langle n, Y\rangle$ est défini par $I \otimes J = \langle m + n, X \uplus Y\rangle$. La Figure 4.3 représente un

exemple du produit tensoriel. Le résultat de cette opération effectuée sur les deux bigraphes G (Figure 4.3a) et B (Figure 4.3b) est donné par le bigraphe A (Figure 4.3c). La région 0 du bigraphe G est placée à côté des deux régions 0 et 1 du bigraphe B et le nœud D est lié sur le nom commun y aux nœuds de contrôle A et B.

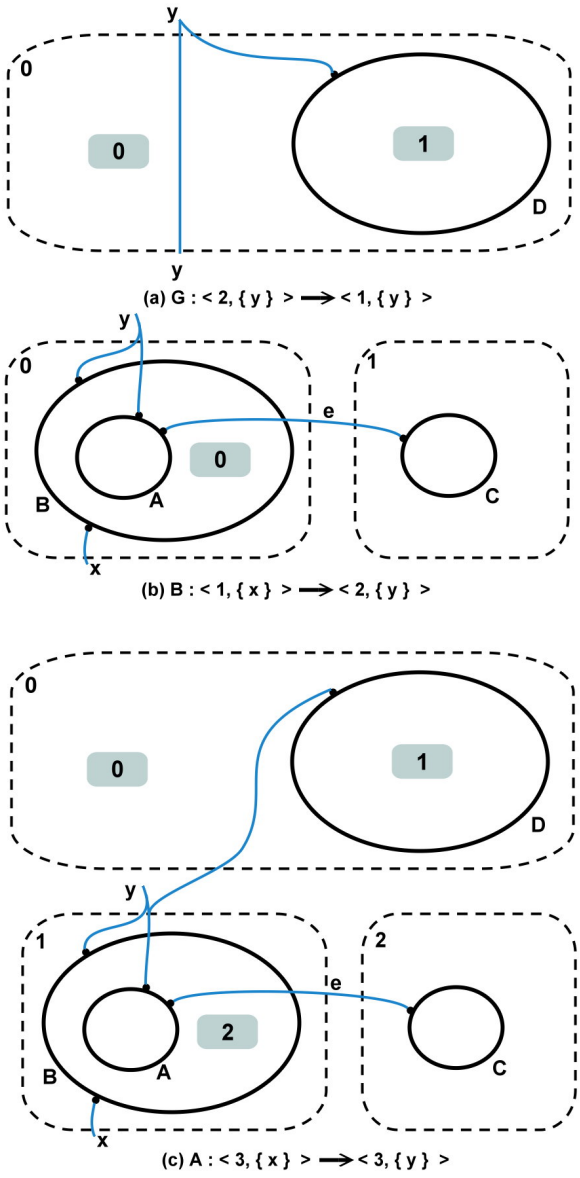


FIGURE 4.3 – Produit tensoriel des bigraphes : $A = G \otimes B$

Définition 4.8 (produit tensoriel des graphes de places). Si $F_i = (V_i, ctrl_i, prnt_i) : m_i \to n_i$, $(i = 0, 1)$ sont deux graphes de places disjoints, leur produit tensoriel $F_0 \otimes F_1 : m_0 + m_1 \to n_0 + n_1$ est donné par :

$$F_0 \otimes F_1 \stackrel{\text{def}}{=} (V_0 \uplus V_1, ctrl_0 \uplus ctrl_1, prnt_O \uplus prnt'_1)$$

où $prnt'_1(m_0 + i) = n_0 + j$ lorsque $prnt_1(i) = j$.

Définition 4.9 (produit tensoriel des graphes de liens). Si $F_i = (V_i, E_i, ctrl_i, link_i) : X_i \to Y_i$, $(i = 0, 1)$ sont deux graphes de liens disjoints, leur produit tensoriel $F_0 \otimes F_1 : X_0 \uplus X_1 \to Y_1 \uplus Y_2$ est donné par :

$$F_0 \otimes F_1 \stackrel{\text{def}}{=} (V_0 \uplus V_1, E_0 \uplus E_1, ctrl_0 \uplus ctrl_1, link_O \uplus link_1).$$

Définition 4.10 (produit tensoriel des bigraphes). Si $F_i : \langle m_i, X_i \rangle \to \langle n_i, Y_i \rangle$, $(i = 0, 1)$ sont deux bigraphes avec des supports disjoints, leur produit tensoriel $F_0 \otimes F_1 : \langle m_0 + m_1, X_0 \uplus X_1 \rangle \to \langle n_0 + n_1, Y_0 \uplus Y_1 \rangle$ est donné par :

$$F_0 \otimes F_1 \stackrel{\text{def}}{=} \langle F_0^P \otimes F_1^P, F_0^L \otimes F_1^L \rangle.$$

4.2.5 Forme algébrique

En plus de leur forme graphique riche et intuitive, un langage de termes algébrique est défini pour représenter les bigraphes. En effet, les bigraphes peuvent être construits en utilisant des bigraphes élémentaires à l'aide des opérations algébriques de composition et du produit tensoriel. À partir de ces opérations, il est possible d'obtenir d'autres opérations supplémentaires ressemblant étroitement aux opérateurs traditionnels trouvés dans l'algèbre de processus. Cela permet de représenter les termes bigraphiques structurellement en termes de notation d'algèbres de processus, telles que la composition parallèle. Dans la suite, nous présentons la définition formelle de certaines opérations principales de ce langage de termes : le produit parallèle (noté par ||), la fusion (notée par |) et l'imbrication (notée par .). Nous récapitulons les opérations de base du langage de termes à travers des exemples dans le Tableau 4.1 [Milner 2009]. Nous relevons que les bigraphes élémentaires sont notés par i_{di} avec $(i = 0, 1, ...)$.

Définition 4.11 (produit parallèle).

Produit parallèle des graphes de places : Si $F_i = (V_i, ctrl_i, prnt_i) : m_i \to n_i$, $(i = 0, 1)$ sont deux graphes de places avec des supports disjoints, leur produit parallèle $F_0||F_1 : m_0 + m_1 \to n_0 + n_1$ est donné par :

$$F_0||F_1 \stackrel{\text{def}}{=} F_0 \otimes F_1.$$

Produit parallèle des graphes de liens : Si $F_i = (V_i, E_i, ctrl_i, link_i) : X_i \to Y_i$, $(i = 0, 1)$ sont deux graphes de liens avec des supports disjoints et $link_0 \cup link_1$ est une fonction de transformation, leur produit parallèle $F_0||F_1 : X_0 \cup X_1 \to Y_0 \cup Y_1$ est donné par :

$$F_0||F_1 \stackrel{\text{def}}{=} (V_0 \uplus V_1, E_0 \uplus E_1, ctrl_0 \uplus ctrl_1, link_O \cup link_1).$$

Produit parallèle des bigraphes : Si $F_i : I_i \to J_i$, $(i = 0, 1)$ sont deux bigraphes avec des supports disjoints, leur produit parallèle $F_0||F_1 : \langle m_0 + m_1, X_0 \cup X_1 \rangle \to \langle n_0 + n_1, Y_0 \cup Y_1 \rangle$ est donné par :

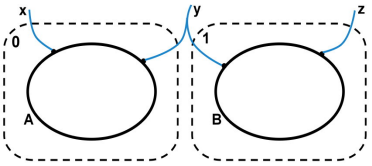
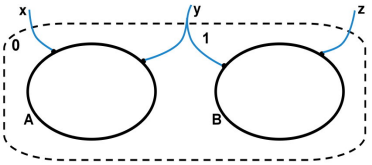
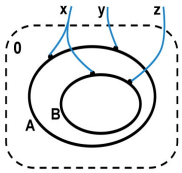
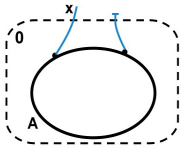
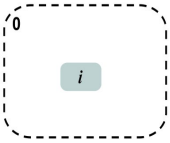
Description	Forme algébrique	Forme graphique
Produit parallèle	$A_x y \| \| B_y z$	
Fusion	$A_x y \| B_y z$	
Imbrication	$A_x y.B_y z$	
Clôture de nom	$/z A_x z$	
Identité (bigraphe élémentaire)	id_i	
Région vide	1	
Site numéroté i	d_i	

TABLE 4.1 – Langage de termes bigraphiques

$$F_0||F_1 \stackrel{\text{def}}{=} \langle F_0^P||F_1^P, F_0^L||F_1^L\rangle.$$

Définition 4.12 (fusion des bigraphes). Étant donné deux bigraphes G_i : $\langle m_i, X_i\rangle \to \langle n_i, Y_i\rangle$ avec $(i = 0, 1)$, supposons que $G_0||G_1$ est défini, leur fusion est donnée par :

$$G_0|G_1 \stackrel{\text{def}}{=} (merge_{n_0+n_1} \otimes id_{Y_0\cup Y_1}) \circ (G_0||G_1)$$

Avec $G_0|G_1 : \langle m_0 + m_1, X_0 \cup X_1\rangle \to \langle 1, Y_0 \cup Y_1\rangle$.

Définition 4.13 (imbrication des bigraphes). Étant donné deux bigraphes $F : I \to \langle m, X\rangle$ et $G : m \to \langle n, Y\rangle$, leur imbrication $G.F : I \to \langle n, X \cup Y\rangle$ est définie par :

$$G.F \stackrel{\text{def}}{=} (G||id_x) \circ F.$$

4.2.6 Logique de typage

Dans la plupart des cas d'application des bigraphes, il est nécessaire d'avoir la possibilité de limiter leurs possibilités de construction et l'ensemble de bigraphes admissibles. Cela peut être réalisé en imposant des contraintes en termes d'une classification de contrôles des bigraphes modélisant un système donné [Milner 2008]. Dans la suite, nous allons définir formellement comment spécifier ce genre de restriction en utilisant le principe de typage et les sortes. Tout d'abord nous donnons quelques notations de base nécessaires. On note les sortes en utilisant des lettres minuscules : a, b, c. Nous écrivons $\widehat{ab}$ pour les sortes disjonctives, signifiant qu'un nœud peut être soit de sorte a ou b. Un bigraphe qui satisfait un typage Σ est dit Σ-typé.

Définition 4.14 (typage de places). Un typage de places $\Sigma_P = (\Theta_P, \mathcal{K}, \Phi_P)$ comporte un ensemble non-vide de sortes Θ_P, une signature Σ_P-typée $\mathcal{K}$ qui associe une sorte à chaque contrôle et un ensemble non-vide de règles de formation pour Φ_P. Φ_P est une propriété d'un ensemble de bigraphes Σ_P-typés qui est satisfaite par les identités et les symétries, et préservée par la composition, le produit tensoriel et le produit parallèle. Lors de l'application d'un typage de places Σ_P sur une interface n, nous écrivons $\vec{\theta}$, où $\vec{\theta} = \theta_n...\theta_0$ énumère les sortes θ_i affectées à chaque $i \in n$.

Définition 4.15 (typage de liens). Un typage de liens est un triplé $\Sigma_L = (\Theta_L, \mathcal{K}, \Phi_L)$, où Θ_L est un ensemble non-vide de sortes, $\mathcal{K}$ est une signature Σ_L-typée associant une sorte à chaque contrôle et Φ_L est un ensemble non-vide de règles de formation pour Σ_L. Φ_L est une propriété d'un ensemble de bigraphes Σ_L-typés qui est satisfaite par les identités et les symétries, et préservée par la composition, le produit tensoriel et le produit parallèle. Lors de l'application d'un typage de liens Σ_L sur une interface $\left\{\vec{x}\right\}$, nous écrivons $\{x_1 : \theta_1, ..., x_n : \theta_n\}$, où chaque $\theta_i \in \Theta_L$.

Exemple. Prenons un exemple simple dans lequel les bigraphes sont utilisés pour modéliser un hôpital. Nous définissons des nœuds de contrôle B pour représenter les bâtiments, des F-nœuds pour modéliser les étages, des R-nœuds pour modéliser les salles de réanimation, des O-nœuds pour représenter les salles d'opération, etc. Un exemple d'un bigraphe modélisant un hôpital est donné par (voir Figure 4.4) :

$$H = B.(F.(R|O)|F.(id|R)).$$

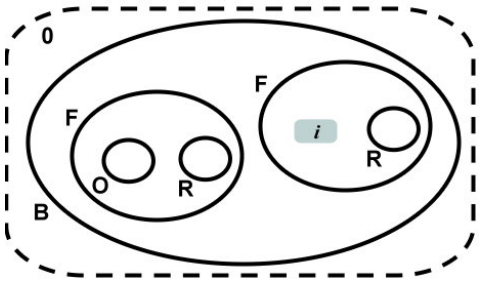


FIGURE 4.4 – Bigraphe H modélisant un hôpital

Nous tenons à noter que la sémantique prévue interdit les R-nœuds de contenir des B-nœuds (les chambres ne peuvent pas contenir des bâtiments). Par conséquent, un bigraphe $H = R.(F|B)$ est considéré incorrect (pas admissible). Dans ce cas, il faut imposer des restrictions sur la construction de bigraphes modélisant un hôpital quelconque. Cela est effectué en utilisant le typage de place comme suit :
$\Theta_P = \{b, f, r\}$, $\mathcal{K} = \{B : b, F : f, R : r, O : r\}$ et $\Phi_P = \{$un nœud de r-sorte est atomique, tous les fils d'un nœud de b-sorte ont une sorte f, tous les fils d'un nœud de f-sorte ont une sorte r $\}$.

Les liens peuvent être classés de la même façon en ajoutant des règles pour imposer des restrictions sur les liens. Par exemple : un nœud de sorte-f est toujours lié à un nœud de sorte-b. Nous notons que le typage de places et de liens sont souvent combinés dans plusieurs applications des bigraphes.

4.2.7 Aspect dynamique des bigraphes

Jusqu'à présent nous avons largement discuté de l'aspect statique et structurel des systèmes réactifs bigraphiques. Néanmoins, ces systèmes sont aussi équipés d'une sémantique dynamique qui définit comment les bigraphes peuvent reconfigurer leurs places et leurs liens. Ces reconfigurations sont définies en termes de règles de réaction semblables aux règles de réécriture de graphes [Milner 2008]. Les règles de réaction bigraphiques diffèrent des règles de réécriture habituelles puisqu'elles sont paramétrées, c'est-à-dire une partie du bigraphe devient un paramètre qui peut être changé ou supprimé par la règle de réaction. En outre, les réactions peuvent être exprimées algébriquement, contrairement aux approches traditionnelles de réécriture de graphes.

Une règle de réaction $R \to R'$ comporte une paire de bigraphe <redex, reactum>. Le redex est une pré-condition pour la réaction qui représente un pattern qui va être changé. Le reactum représente la post-condition qui spécifie le pattern après le changement effectué par la réaction. On distingue deux types de reconfigurations : (1) reconfiguration sur les places par l'ajout, la suppression ou le déplacement d'un nœud (2) reconfiguration sur les liens par la création ou la destruction d'un lien entre deux nœuds ou entre un nœud et un nom interne/externe. Un exemple d'une règle réaction est donné dans la Figure 4.5. Cet exemple montre comment un bigraphe dans un état initial $S0$ sur le côté gauche (redex), évolue pour devenir un autre bigraphe $S1$ sur le côté droit (reactum). La règle modélise une personne avec un nœud de contrôle P dans la même région qu'une salle représentée par un nœud de contrôle R. Elle lui permet d'entrer dans la salle et se connecter à l'ordinateur modélisé par un nœud de contrôle PC. Le nœud P est déplacé à l'intérieur du nœud R et un lien est créé entre les deux nœuds P et PC. Cette règle de réaction peut être représentée algébriquement par : $P|R.(PC) \to R.(P_e|PC_e)$.

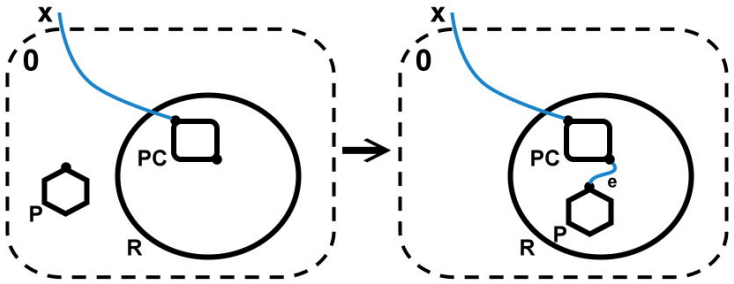


FIGURE 4.5 – Exemple d'une règle de réaction bigraphique

Définition 4.16 (règle de réaction). Une règle de réaction prend la forme : $R = (R : m \to J, R' : m \to J, \eta)$ et généralement écrite $R \to R'$, où $R : m \to J$ est un bigraphe appelé redex (pattern à changer), $R' : m \to J$ est également un bigraphe appelé reactum (nouveau bigraphe changé) et l'instanciation $\eta : m' \to m$ est une transformation d'ordinaux.

Après avoir défini formellement un bigraphe et une règle de réaction, nous pouvons maintenant donner la définition d'un système réactif bigraphique (BRS).

Définition 4.17 (système réactif bigraphique). Un système réactif bigraphique (BRS) consiste en une paire (B, R), où B est un ensemble de bigraphes et R est un ensemble de règles de réaction définies sur B. La relation de réaction d'un BRS est représentée par $\to$. Nous écrivons $(B(\Sigma), R)$ lorsque les éléments de B sont Σ-typés.

4.2.8 Logique spatiale BiLog

La logique spatiale BiLog a été introduite par Conforti [Conforti 2006]. En BiLog, il est possible d'exprimer des formules comme $A \circ \varphi$ pour décrire un nœud de contrôle A agissant comme un contexte d'un bigraphe satisfaisant φ et $(A_a \circ T) \otimes (B_b \circ T)$ pour représenter deux places adjacentes avec des noms différents et contenus quelconques. Les formules de la logique BiLog sont présentées ci-dessous. Nous indiquons qu'un ensemble de bigraphes élémentaires dans BiLog est décrit par $\Omega(\mathcal{K})$. La signature $\mathcal{K}$ est souvent omise lorsqu'une signature arbitraire est présumée.

$$\Omega ::= id_I | K_{\vec{x}} | join | 1 | \gamma_{m,n} | y/X | /x$$
$$\varphi, \psi ::= \bot | \varphi \Rightarrow \psi | id | \Omega | \varphi \circ \psi | \varphi \otimes \psi$$

4.2.9 Outils autour des BRS

Les systèmes réactifs bigraphiques représentent un formalisme relativement récent. Néanmoins, un nombre d'outils et d'environnements de manipulation des systèmes réactifs bigraphiques ont vu le jour. Cependant, ces outils ne sont pas encore matures et la majorité d'entre eux représentent que des prototypes ayant plusieurs limites. Dans cette section, nous présentons quelques outils pertinents pour les BRS.

BPL Tool. Est un prototype qui représente une des premières implémentations de systèmes réactifs bigraphiques. BPL Tool permet la manipulation, la simulation et la visualisation des systèmes réactifs bigraphiques. Cet outil consiste en un parseur, un moteur de matching et un moteur de normalisation. Il comporte ainsi, une interface web et une interface en ligne de commandes. Le langage utilisé dans l'outil BPL Tool est appelé BPL ; il consiste en un ensemble de blocs de langage Standard ML (SML) [Milner 1997] qui permettent d'écrire BPL directement dans SML. L'outil BPL Tool a été utilisé pour la spécification de plusieurs types de systèmes tels que : les systèmes de téléphonie mobile [Højsgaard 2011], WS-BPEL, HomeBPEL [Bundgaard 2008] et les modèles platographiques [Elsborg 2009].

BigMC. Bigraphical Model Checker est le seul model-checker conçu pour opérer sur les systèmes réactifs bigraphiques. Il permet d'effectuer des vérifications formelles sur différents systèmes modélisés à l'aide de BRS. L'outil BigMC est basé sur le moteur de matching de BPL, ce qui lui permet d'explorer tous les états possibles d'un système bigraphique et de vérifier si une propriété est vraie ou fausse dans chaque état du système. BigMC a la possibilité de fournir des contre-exemples ou des traces d'exécution dans le cas où la propriété n'est pas vérifiée. Les contre-exemples peuvent être utilisés pour découvrir les causalités ayant mené à des violations de la propriété. BigMC implémente une vérification explicite d'états avec un support limité pour la vérification symbolique.

BigraphER. Consiste en une bibliothèque OCaml et une ligne de commandes qui permettent la manipulation, la visualisation et la simulation des systèmes réactifs bigraphiques, BRS stochastiques et les BRS avec partage [Sevegnani 2015, Sevegnani 2016]. L'outil est composé de trois modules distincts : un compilateur (compiler), un moteur de matching (matching engine) et un moteur de réécriture

(rewriting engine). L'outil prend en entrée un fichier source contenant la spécification du modèle. Le langage utilisé dans l'outil est appelé "BigraphER Specification Language", qui ressemble à la forme algébrique de bigraphes. Le langage de spécification BigraphER permet de définir la signature du modèle, un ensemble de bigraphes et un ensemble de règles de réaction. Le compilateur est le composant qui traduit le fichier source d'entrée en une représentation du modèle au moment d'exécution. En outre, le compilateur peut également générer une représentation graphique de chaque bigraphe spécifié dans le fichier d'entrée à l'aide du générateur automatique de graphes Graphviz [Ellson 2001]. Le moteur de matching implémente un algorithme de matching basé sur un codage SAT. Il est utilisé par le moteur de réécriture pour appliquer des règles de réaction à un état et pour vérifier l'égalité des états.

Les différentes limites de ces outils nous ont conduit à l'utilisation de langage Maude comme l'alternative la plus adéquate qui permet d'exécuter et de vérifier formellement les modèles bigraphiques obtenus. La section suivante est consacrée à la présentation de ce langage formel.

4.3 Langage Maude

Maude [Clavel 2007, Clavel 2011] est un langage de haut niveau, basé sur les théories mathématiques de la logique de réécriture et la logique équationnelle. Maude est destiné principalement à la spécification formelle des systèmes informatiques, et plus particulièrement les systèmes concurrents et répartis. Ce langage a été développé sous la direction de José Meseguer au centre de recherche américain "SRI Laboratory" en collaboration avec l'université de l'Illinois.

Dans le système Maude, les aspects structurels et statiques d'un système donné sont décrits par des équations et des types de donnés, alors que les aspects comportementaux et dynamiques sont décrits par des règles de réécriture. Les spécifications Maude sont exécutables, et elles peuvent être soumises à la simulation et l'analyse formelle en utilisant l'interpréteur Maude et plusieurs techniques et outils de vérification et d'analyse formelle tels que LTL Maude et la technique de vérification par invariants. Dans cette section, nous introduisons l'environnement Maude, ses spécificités et son fonctionnement.

Le langage Maude est caractérisé par [Clavel 2007] :

La simplicité. Les expressions de programmation de base dans le langage Maude sont simples et faciles à comprendre. En effet, ces expressions qui ont une simple interprétation de réécriture sont soient des équations soient des règles de réécriture dans lesquelles, le côté gauche représente un pattern qui peut être remplacé par un autre pattern dans le côté droit.

L'expressivité. Le langage Maude permet de représenter naturellement les systèmes déterministes et non déterministes grâce aux modules fonctionnels et les modules systèmes. Le calcul déterministe est implémenté à l'aide des équations dans des modules fonctionnels. Par contre, le calcul non déterministe est représenté avec des règles de réécriture dans des modules systèmes.

La performance. Cet aspect a été pris en charge par les implémentations successives de Maude. Maude est compétitive en termes de temps d'exécution, où plusieurs règles de réécriture peuvent être exécutées par seconde.

4.3.1 Syntaxe et notations

Dans le langage Maude, les unités basiques de spécification ou de programmation sont appelées des modules. On distingue trois types de modules : les modules fonctionnels pour implémenter les théories équationnelles, les modules systèmes qui implémentent les théories de réécriture et définissent le comportement dynamique d'un système, et les modules orientés-objet qui implémentent les théories de réécriture orientées-objet. Dans cette section, nous fournissons les notations et les aspects syntaxiques de base des différents modules Maude.

Modules fonctionnels

Ces modules définissent les types de données et les opérations qui sont utilisées par les équations (conditionnelles et non conditionnelles), à savoir : les sortes, les sous-sortes et les opérations. Les attributs des opérations sont définis sous Maude à l'aide des mots-clés : `ctor` (constructeur), `assoc` (associative), `comm` (commutative), `id` (élément neutre) et `prec` (degré de précédence). Un module fonctionnel dans Maude est déclaré avec le mot-clé `fmod`, suivi de nom du module :

`fmod <nom-module> is <déclarations et expressions> endfm .`

Les déclarations et les expressions dans un module fonctionnel peuvent être :

Des importations d'autres modules. Des modules fonctionnels prédéfinis peuvent être importés en utilisant les primitives `protecting, including` ou `extending`.

Des déclarations des sortes ou des sous-sortes. Les sortes (les types de données) sont déclarées avec le mot-clé `sort`, et les sous-sortes sont définis en utilisant `subsort`. Dans le cas où nous avons plusieurs sortes ou sous-sortes à déclarer, nous utilisons le mot-clé `sorts` ou `subsorts`.

Des déclarations des opérations. Les opérations qui agissent sur les sortes et les sous-sortes sont déclarées à l'aide du mot-clé `op`.

Des déclarations des équations. Les équations peuvent être conditionnelles ou non. Une équation non conditionnelle suit la syntaxe suivante : `eq <Terme-1>=<Terme-2>[<Attributs>] .`

Les deux termes `Terme-1` et `Terme-2` dans l'équation `Terme-1 = Terme-2` doivent avoir la même sorte. Pour qu'une équation soit exécutable, toutes les variables dans `Terme-1` (la partie droite de l'équation) doivent apparaître dans `Terme-2` (la partie gauche). Les équations conditionnelles prennent la forme suivante :

`ceq <Terme-1>=<Terme-2> if <EqCondition-1>...<EqCondition-k> [<Attributs>] .`

La syntaxe des conditions d'une équation conditionnelle est de trois variantes :

- Équations ordinaires de la forme `t = t'`,
- Équations de correspondance (matching) `t := t'`,

- Équations booléennes de la forme `t`, où `t` est un terme algébrique de genre `[Bool]`, équivalent à `t = true`.

Modules systèmes

Un module système dans Maude implémente une théorie de réécriture. Dans une théorie de réécriture on retrouve des sortes, des types, des opérateurs, des déclarations d'équations, et des règles de réécriture, qui peuvent être conditionnelles ou non. Par conséquent, toute théorie de réécriture a une théorie équationnelle sous-jacente. Les transitions concurrentes locales dans un module système sont représentées par des règles de réécriture. Les règles peuvent être exécutées si la partie gauche d'une règle correspond à un fragment de l'état du système, et si la condition de la règle est satisfaite. La transition spécifiée par la règle pourrait être appliquée, et le fragment identifié de l'état se transforme en l'instance correspondante du côté droit. Un module système Maude est déclaré selon la syntaxe :

```
mod <nom-module> is <déclarations-et-expressions> endm .
```

Un module système dans Maude est déclaré avec le mot-clé `mod`, suivi de nom du module. Les déclarations et les expressions dans un module fonctionnel peuvent être : des importations d'autres modules fonctionnels ou systèmes, des déclarations des sortes ou des sous-sortes, déclarations des opérations, déclarations des équations et déclarations des règles de réécriture conditionnelles. Les règles de réécriture conditionnelles et non conditionnelles déclarées au sein d'un module système prennent la syntaxe suivante :

```
rl [<Étiquette>] : <Terme-1> => <Terme-2> [<Attributs>] .
crl [<Étiquette>] : <Terme-1> => <Terme-2> if <Condition-1>...
<Condition-k>[<Attributs >] .
```

Les deux termes `<Terme-1>` et `<Terme-2>` sont des termes de même sorte qui peuvent contenir des variables. `<Étiquette>` est l'étiquette de la règle de réécriture ; elle peut être omise. Les conditions d'une règle de réécriture conditionnelle `<Condition-k>` peuvent contenir des expressions de réécriture qui testent la possibilité de réécrire des termes algébriques.

Modules orientés objet

Dans le langage Maude, les systèmes basés sur le paradigme orienté objet peuvent être définis à travers des modules orientés objet. Les modules orientés objet sont déclarés avec la syntaxe suivante : `omod < nom-module > is < déclarations et expressions > endom .`

Les modules orientés objet offrent une syntaxe appropriée qui permet de déclarer les concepts de base du paradigme orienté objet :

- `Oid` pour identifier les objets,
- `Cid` pour identifier les classes,
- `Objects` pour définir les objets et
- `Msg` pour déclarer les messages.

Dans les modules orientés objet, un objet est représenté par un terme :
`< O : C | a1 : v1 ,..., an : vn >,` où `O` est l'identificateur de l'objet, `C` est l'identificateur de sa classe, les `ai` sont les noms des attributs de l'objet et enfin les `vi` correspondent aux valeurs des attributs. Les messages dans les modules orientés objet n'ont pas de forme syntaxique prédéfinie, leur structure est définie par l'utilisateur. La seule contrainte est que le premier argument doit être l'identifiant de l'objet destination.

4.3.2 Langage de stratégies

Le passage de la logique équationnelle à la logique de réécriture permet de spécifier le comportement dynamique des différents types de systèmes en utilisant les règles de réécriture. Cependant, à cause de la nature déclarative et non déterministe de ces règles, il est très difficile de contrôler leur exécution, et assurer que le processus de réécriture ne va pas dans des directions indésirables.

Le langage de stratégies associé au langage Maude [Eker 2007, Martí-Oliet 2009] fournit un cadre qui permet de définir et d'exprimer différentes stratégies et plans d'exécutions contrôlant la façon dont les termes sont réécrits dans Maude et imposant des restrictions sur l'exécution des règles. La conception du langage est basée sur une séparation stricte entre les règles de réécriture définies dans des modules systèmes et les expressions des stratégies qui sont spécifiées séparément dans des modules de stratégies. En fait, cette séparation permet de définir divers modules de stratégie pour contrôler de différentes manières les réécritures d'un module système donné. Les modules de stratégies sont déclarés en suivant la syntaxe :

```
smod <nom-module> is <déclarations-et-expressions> endsd .
```

Un module de stratégies est déclaré avec le mot-clé `smod`, suivi de nom du module. Les déclarations et les expressions dans un module de stratégies sont des variables, des importations d'autres modules et des déclarations de stratégies. Une stratégie déclarée dans un module de stratégies est définie sous la forme :
`strat s : @ type-de-terme . sd s := Expression .`
L'identificateur d'une stratégie est déclaré en utilisant le mot-clé `start`. Le type de terme sur lequel cette stratégie peut être appliquée est spécifié après le symbole . La définition d'une stratégie est introduite avec le mot-clé `sd`, suivi de l'identificateur de stratégie et l'expression de cette stratégie. Les définitions des stratégies peuvent être conditionnelles, dans ce cas, une stratégie est définie avec le mot-clé `csd`.

Une stratégie est décrite dans Maude comme étant une opération qui produit un ensemble de termes (résultat), lorsqu'elle est appliquée sur un terme donné. Les stratégies peuvent être définies à partir des stratégies de base. Ces dernières consistent en l'application d'une règle de réécriture (identifiée par l'étiquette de la règle) sur un terme donné ; elle permet aux variables de la règle d'être instanciées à travers des substitutions avant l'application de la règle. Les stratégies basiques sont combinées pour former des stratégies plus complexes à l'aide de nombreux opérateurs et expressions régulières de combinaisons telles que, l'union (`|`), la concaténation (`;`),

l'itération (`*` et `+`) et d'autres types d'opérateurs. L'opérateur de concaténation (`;`) est associatif alors que l'opérateur d'union (`|`) est associatif et commutatif. Les différents opérateurs utilisés pour construire des stratégies représentant le langage de stratégies de Maude sont récapitulés dans le Tableau 7.1 [Eker 2007].

Opérateur	Description
`idle`	Le processus d'exécution retourne toujours un résultat sans modifier un terme `t` sur lequel il est appliqué
`fail`	Le processus d'exécution ne retourne jamais un résultat (échoue)
`l (étiquette)`	Créer un processus d'application pour une règle avec une étiquette `l`
`s1 ; s2`	Imposer `s2`, suivi par `s1` sur la pile du processus en cours d'exécution
`s1 | s2`	Cloner le processus en cours d'exécution. Imposer `s1` sur la pile du processus cloné et `s2` sur la pile du processus original
`s*`	Cloner le processus en cours d'exécution et imposer `s*` suivi par `s` sur la pile du processus en cours d'exécution
`s+`	Imposer `s*` suivi par `s` sur la pile du processus en cours d'exécution
`!`	Répéter jusqu'à la fin du processus d'exécution

TABLE 4.2 – Langage de stratégies de Maude

Exemple. À travers l'exemple de "traverser la rivière" (River Crossing), nous illustrons comment ce problème peut être résolu par l'utilisation des stratégies sous Maude [Martí-Oliet 2009] . Dans ce problème un berger a besoin de transporter à l'autre côté d'une rivière un loup, une chèvre et un chou. Le berger n'a qu'un seul bateau avec deux places pour lui-même et pour un autre voyageur. Le problème dans cet exemple est que, en l'absence du berger, le loup mangerait la chèvre et la chèvre mangerait le chou. Les côtés gauches et droits de la rivière sont représentés par les opérations `left` et `right` dans un module système (voir Listing 4.1). Le berger, le loup, la chèvre et le chou sont aussi représentés par des opérations `s, w, g` et `c` avec des attributs de différentes sortes indiquant le groupe de chaque élément et le côté de la rivière où chacun se trouve. L'opération `Initial` désigne la situation initiale, où il est supposé que tous les éléments sont situés sur le côté gauche de la rivière à travers une équation. Les règles de réécriture représentent comment le loup et la chèvre mangent et les différentes manières de traverser la rivière.

Pour résoudre ce problème, les deux premières équations sont utilisées pour forcer le système Maude à les appliquer avant toute règle de réécriture. Mais outre le fait que cette solution introduit un problème de cohérence qui devait être résolu, elle change aussi la sémantique du problème. Une autre solution consiste à utiliser trois stratégies définies dans un module de stratégies (voir Listing 4.2). La stratégie `eating` qui contrôle la façon de manger pour la chèvre et le loup (contrôle l'exécution des règles `wolf-eats` et `goat-eats`). La stratégie `onecross` qui applique une des autres règles une seule fois. Enfin, la stratégie `allce` renvoie tous les états accessibles

où les règles de manger ont eu la priorité.

Listing 4.1– Module système RIVER-CROSSING

```
mod RIVER-CROSSING is
sorts Side Group .
ops left right : -> Side .
op change : Side -> Side .
ops s w g c : Side -> Group .
op __ : Group Group -> Group [assoc comm] .
op init : -> Group .
vars S S' : Side .
eq change(left) = right .
eq change(right) = left .
eq initial = s(left) w(left) g(left) c(left) .
crl [wolf-eats] : w(S) g(S) s(S') => w(S) s(S') if S =/= S' .
crl [goat-eats] : c(S) g(S) s(S') => g(S) s(S') if S =/= S' .
rl [shepherd-alone] : s(S) => s(change(S)) .
rl [wolf] : s(S) w(S) => s(change(S)) w(change(S)) .
rl [goat] : s(S) g(S) => s(change(S)) g(change(S)) .
rl [cabbage] : s(S) c(S) => s(change(S)) c(change(S)) .
endm
```

Listing 4.2– Module de strategies RIVER-CROSSING-STRAT

```
smod RIVER-CROSSING-STRAT is
protecting RIVER-CROSSING .
strat eating : @ Group .
sd eating := (wolf-eats | goat-eats) ! .
strat oneCross : @ Group .
sd oneCross := shepherd-alone | wolf | goat | cabbage .
strat allCE : @ Group .
sd allCE := (eating ; oneCross) * .
endsm
```

4.3.3 Analyse formelle dans Maude

Une théorie de réécriture est spécifiée dans Maude comme un module système fournissant un modèle mathématique exécutable. Ainsi, les spécifications Maude peuvent être utilisées pour simuler les différents comportements des systèmes à modéliser. Néanmoins, les spécifications Maude peuvent être utilisées pour effectuer de nombreux types d'analyses formelles. Dans des conditions appropriées, les modèles mathématiques peuvent être vérifiés pour assurer la satisfaction de certaines propriétés définies, ou pour fournir un ou plusieurs contre-exemples montrant la violation d'une propriété en question.

Le langage Maude est entouré de nombreuses techniques et outils d'analyses formelles des comportements de systèmes tels que, la technique de vérification par invariants, le prouveur de théorèmes, le model-checker LTL, l'analyse de terminaison et l'analyse de cohérence. Dans cette section, nous abordant la technique de vérification par invariants et model-checker LTL, en raison de leur pertinence avec

le travail présenté dans cette thèse. Pour plus de détails, le lecteur peut consulter [Clavel 2007, Clavel 2011].

La technique de vérification par invariants est une technique simple, mais très utile, qui peut être menée dans Maude en utilisant la commande `search`. Elle représente une des techniques les plus courantes pour vérifier les propriétés de sûreté dans les différents types de systèmes informatiques. Néanmoins, elle peut être utilisée aussi pour la vérification des propriétés de vivacité.

Étant donné un système de transition `t` et un état initial `s0`, un invariant `I` est un prédicat définissant un sous-ensemble d'états qui comprend l'état `s0` et tous les états accessibles à partir de `s0` par un nombre fini de transitions. Par conséquent, un invariant est un prédicat qui définit un ensemble d'états contenant tous les états accessibles depuis `s0`. Dans ce cas, la sûreté est vérifiée si l'invariant est maintenu, ce qui signifie que rien de mauvais ne peut jamais se produire. Les propriétés de vivacité peuvent donc être vérifiées si on considère le cas contraire, c'est-à-dire, que les invariants sont considérés comme des états désirables (des états recherchés ou souhaités).

La technique de vérification par invariants est implémentée dans Maude par la commande `search`. Sa syntaxe est conforme à la forme générale suivante [Clavel 2011] : `search [n, m] in <nom-module>: <Terme-1> <flèche-de-recherche> <Terme-2> such that <condition> .`

- `n` est un argument optionnel fournissant une limite sur le nombre de solutions souhaitées,
- `m` est un argument optionnel indiquant la profondeur maximale de la recherche,
- `<nom-module>` le module où la recherche aura lieu,
- `<Terme-1>` le terme initial,
- `<Terme-2>` le pattern qui doit être atteint,
- `<flèche-de-recherche>` est une flèche indiquant la forme de la preuve de réécriture de `<Terme-1>` au `<Terme-2>` :
  - `=> 1` la preuve de réécriture comprenant exactement une seule étape,
  - `=> +` la preuve de réécriture consiste en une ou plusieurs étapes,
  - `=> *` la preuve consiste en aucune, une ou plusieurs étapes et
  - `=>!` indique que seuls les chemins à états déterministe sont exploité.
- `<Condition>` est une propriété optionnelle qui doit être satisfaite par l'état atteint.

La deuxième technique de vérification, que nous présentons, est basée sur la logique temporelle linéaire (LTL). Elle représente une technique riche, intuitive et largement utilisée pour la spécification et la vérification de différents types de propriétés. Par exemple, les propriétés de sûreté (quelque chose de mauvais n'arrive jamais), de vivacité (quelque chose de bon finit par arriver) et d'équité (quelque chose de bon se répète infiniment).

Soit AP un ensemble fini de propositions élémentaires. Les formules de la logique temporelle linéaire propositionnelle LTL (AP) sont définies dans Maude inductivement comme suit :

- True : T∈LTL(AP).
- Propositions élémentaires : si p ∈ AP, alors p ∈ LTL(AP).
- Opérateur Next : si $\phi \in$ LTL(AP), alors $\phi \in$ LTL(AP).
- Opérateur Until : si $\phi, \psi \in$ LTL(AP), alors $\phi \cup \psi \in$ LTL(AP).
- Opérateurs logiques : si $\phi, \psi \in$ LTL(AP), alors les formules $\neg\phi$, et $\phi \vee \psi$ appartient à LTL(AP).

D'autres opérateurs et conjonctions LTL peuvent être définis en termes de l'ensemble de conjonctions présentées comme suit :

Opérateurs booléens :

- False : $\perp = \neg T$.
- Conjunction : $\phi \wedge \psi = \neg((\neg\phi) \vee (\neg\psi))$
- Implication : $\phi \rightarrow \psi = (\neg\phi) \vee \psi$.

Opérateurs temporels supplémentaires :

- Eventually : $<> \phi = T \cup \phi$.
- Henceforth : $[]\phi = \neg <> \neg\phi$.
- Release : $\phi R \psi = \neg((\neg\phi) \cup (\neg\psi))$.
- Unless : $\phi W \psi = (\phi \cup \psi) \vee ([]\phi)$.
- Leads-to : $\phi \rightarrow \psi = [](\phi \rightarrow (<> \psi))$.
- Strong implication : $\phi \Rightarrow \psi = [](\phi \rightarrow \psi)$.
- Strong equivalence : $\phi \Leftrightarrow \psi = [](\phi \leftrightarrow \psi)$.

La signature LTL de la syntaxe mathématique ci-dessus est définie dans Maude à travers un module fonctionnel déclaré dans le fichier "model-checker.maude".

4.4 Conclusion

Dans ce chapitre, nous avons introduit les fondements formels adoptés dans notre travail. Dans un premier temps, nous avons présenté les systèmes réactifs bigraphiques. Pour ce faire, nous avons d'abord commencé par décrire l'anatomie des bigraphes, leur forme graphique et définitions formelles. Nous avons également exposé à travers des exemples les principales opérations qui peuvent être effectuées sur les bigraphes. Ensuite, nous avons introduit leur forme algébrique, ainsi que le principe de typage des bigraphes. Enfin, nous avons présenté l'aspect dynamique des systèmes réactifs bigraphiques, la logique spatiale BiLog et les principaux outils autour de ce formalisme. Nous avons aussi présenté les différents concepts relatifs au langage Maude. Plus précisément, nous avons introduit la syntaxe et les notations du langage Maude, le langage de stratégies associé et les techniques d'analyses formelles dans le système Maude.

Dans le chapitre suivant, nous aborderons une approche de modélisation des systèmes élastiques basés cloud en adoptant les systèmes réactifs bigraphiques typés.

Chapitre 5

Une architecture basée BRS pour la modélisation des systèmes élastiques cloud

Sommaire

5.1 Introduction 68
5.2 Principe de modélisation 69
5.2.1 Éléments architecturaux d'un système élastique cloud 69
5.2.2 Méta-modèle d'un système élastique cloud 70
5.3 Sémantique basée BRS d'un système élastique cloud 72
5.3.1 Bigraphe front-end 76
5.3.2 Bigraphe back-end 78
5.3.3 Bigraphe contrôleur d'élasticité 79
5.4 Conclusion 82

5.1 Introduction

L'émergence du paradigme cloud computing a donnée naissance à un nouveau type de systèmes autonomiques dans lesquels l'élasticité est un principe de conception clé. Ces systèmes sont appelés les systèmes élastiques cloud. L'élasticité des ressources dans ce type de systèmes présente un grand avantage pour les utilisateurs cloud, ainsi que pour les fournisseurs de services qui exploitent cette caractéristique pour trouver un compromis entre la qualité de service (QoS), la charge de travail actuelle, et les coûts d'exploitation dans les environnements cloud computing. Leur objectif principal est de maintenir un niveau adéquat de qualité de service et minimiser les coûts lorsque le système est confronté à des charges de travail fluctuantes.

La conception et la mise en œuvre des systèmes élastiques basés cloud sont des tâches très stimulantes et délicates qui peuvent provoquer des situations indésirables telles que l'instabilité dans l'allocation des ressources et la dégradation sévère de la qualité de service, si elles ne sont pas correctement effectuées. Par conséquent, il est nécessaire de définir un modèle qui permet de décrire précisément les architectures des systèmes basés cloud et leurs comportements élastiques. Dans notre travail, nous prétendons que les méthodes formelles caractérisées par leur efficacité, fiabilité et

précision, sont un excellent candidat qui permet de réduire la complexité de la modélisation des systèmes élastiques basés cloud. En particulier, nous nous intéressons aux systèmes réactifs bigraphiques [Milner 2008, Milner 2009] qui diffèrent des formalismes traditionnels tels que la notation Z, les réseaux de Petri, la méthode B et l'algèbre de processus par leur aspect graphique et leur capacité de représenter à la fois la localité et la connectivité des systèmes distribués. En effet, les BRS représente une solution idéale pour capturer la complexité architecturale d'un système élastique cloud. D'une part, les notations algébriques et l'aspect graphique orientée humain des systèmes réactifs bigraphiques fournissent une représentation intuitive et claire des différents aspects conceptuels et architecturaux des systèmes élastiques basés cloud. D'autre part, la discipline de typage associée aux bigraphes permet d'exprimer la sémantique des systèmes cloud avec une grande précision. Cela permet de retenir seulement des modèles conçus avec une structure significative désirée.

Dans ce chapitre, nous présentons un modèle formel, basé sur les systèmes réactifs bigraphiques et leur logique de typage, permettant de modéliser les aspects architecturaux des systèmes élastiques cloud. Tout d'abord, nous présentons le principe de modélisation de l'approche proposée en introduisant les différents éléments architecturaux constituant tels systèmes. Ensuite, nous proposons une sémantique formelle basé bigraphes qui permet de modéliser indépendamment chaque partie d'un système élastique basé cloud, à savoir, les parties front-end, back-end et contrôleur d'élasticité.

5.2 Principe de modélisation

Avant d'aborder la spécification formelle des systèmes élastiques cloud en utilisant les BRS, nous devons tout d'abord dégager les éléments architecturaux qui constituent de tels systèmes à partir des définitions existantes dans la littérature. Ensuite, nous proposons un méta-modèle semi formel qui décrit les principaux éléments architecturaux et les différentes dépendances entre eux.

5.2.1 Éléments architecturaux d'un système élastique cloud

Un système élastique cloud peut être divisé en trois entités principales [Bersani 2014, Dustdar 2015], le front-end, le back-end et le contrôleur d'élasticité interagissant entre elles via plusieurs moyens (voir Figure 5.1) :

Le front-end. Représente l'interface utilisée par les clients pour accéder au cloud. Les clients de cloud sont classés en deux types : des utilisateurs finaux (end users) qui sont des simples consommateurs de services, et des développeurs (developers) qui représentent des clients exploitant le cloud pour héberger leurs applications.

Le back-end. Représente le système à contrôler caractérisé par différentes instances de calcul (ressources et applications). À un haut niveau d'abstraction, il est considéré comme un ensemble de ressources informatiques (ex. serveurs, machines virtuelles, conteneurs, équilibreurs de charge, applications, etc.) distribuées sur plusieurs sites informatiques. Ces différentes ressources sont fournies en tant que

services à la demande que les utilisateurs (clients) peuvent consommer selon leurs besoins et en ne payant que leurs consommations réelles.

Le contrôler d'élasticité. Est l'entité responsable de la supervision du système cloud et la prise de décision d'élasticité. Dans un système élastique basé cloud, le contrôleur d'élasticité est généralement mis en œuvre par une boucle de contrôle autonomique fermée. Une des boucles de contrôles autonomiques la plus adoptée pour la gestion de l'élasticité dans le cloud est la boucle de contrôle MAPE-K (Monitor, Analyse, Plan, Execute) de IBM [IBM Group 2005].

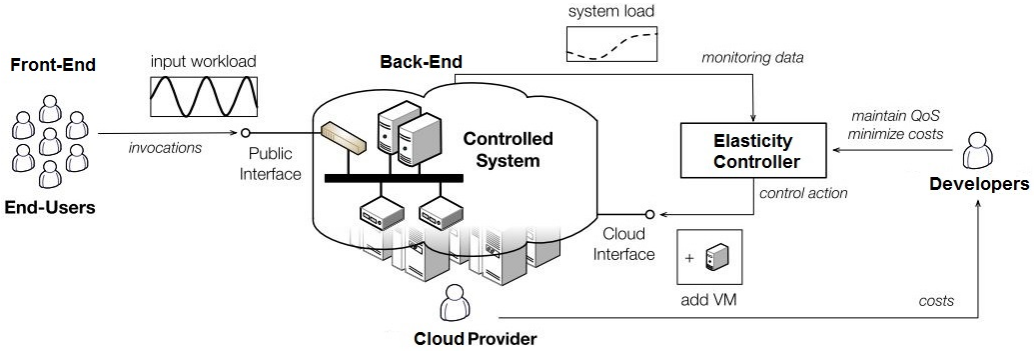


FIGURE 5.1 – L'architecture d'un système élastique cloud [Bersani 2014]

5.2.2 Méta-modèle d'un système élastique cloud

L'ensemble des notions et des concepts d'un système élastique cloud pris en considération dans notre modèle bigraphique est issu du méta-modèle illustré par la Figure 5.2. Ce méta-modèle cerne tous les éléments principaux, jugés nécessaires pour la modélisation des architectures de ce type de systèmes.

Dans le méta-modèle de la Figure 5.2, les éléments architecturaux sont représentés sous la forme d'instances cloud de différents types. Une instance cloud peut être un équilibreur de charge, un serveur, une machine virtuelle, un conteneur ou une application. Ces diverses instances possèdent des attributs similaires qui décrivent leurs identifiants, leurs états (chargés ou sous-utilisés) et la quantité de ressources associée à chaque instance (mémoire, CPU, stockage et bande passante). Dans un système élastique cloud, ces instances sont hébergées généralement dans le back-end du système où il existe une relation hiérarchique entre les différents types d'instances. Par exemple, un serveur peut contenir seulement des instances de conteneurs et des instances de machines virtuelles qui ne peuvent elles-mêmes contenir que des instances d'applications.

Les instances cloud sont offertes aux divers clients en tant que services et ressources informatiques consommables à la demande. Nous pouvons distinguer deux types de clients dans le méta-modèle, des développeurs et des utilisateurs finaux. Les clients cloud peuvent accéder à leurs instances allouées ou demander d'autres services en utilisant le front-end du système.

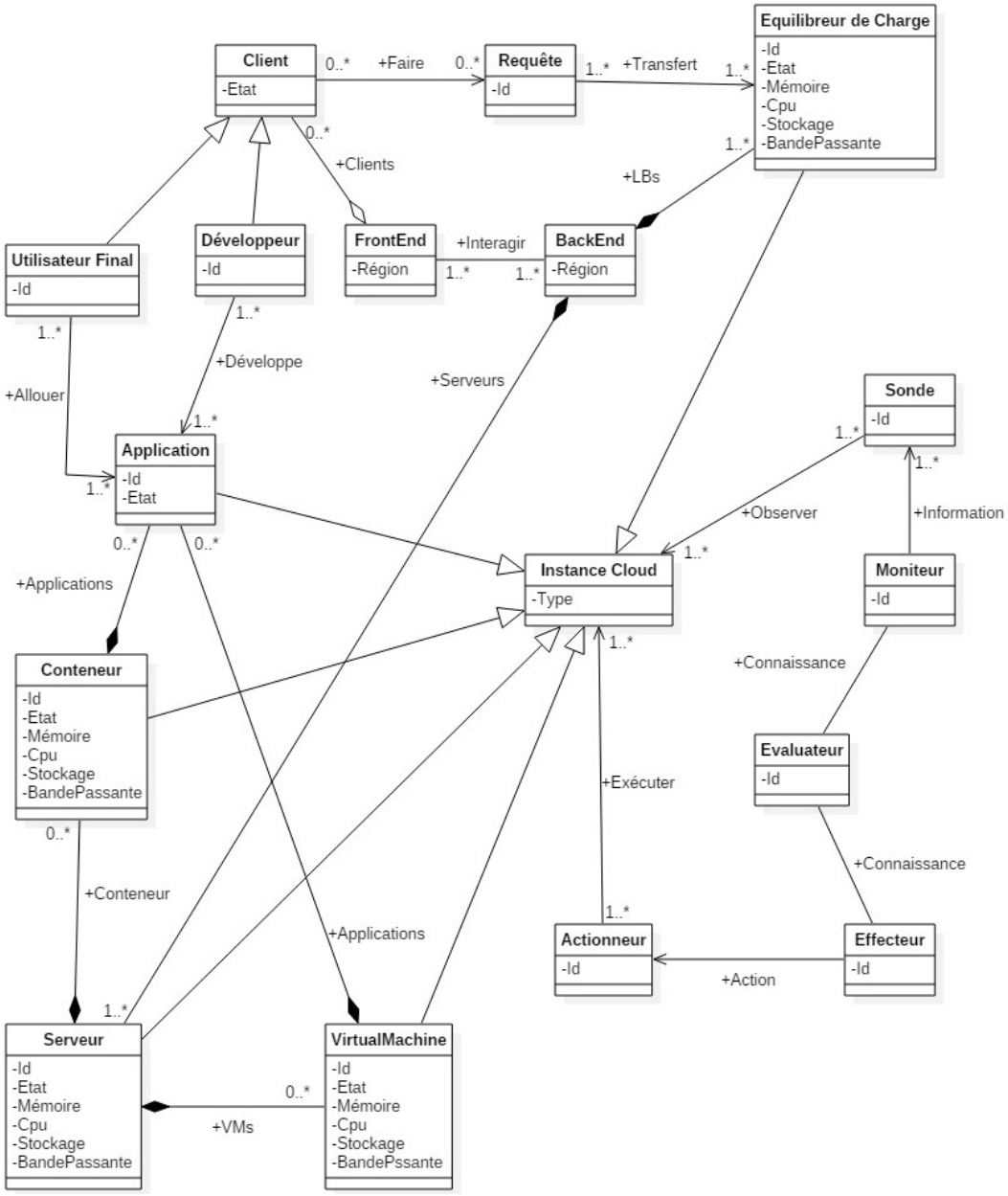


FIGURE 5.2 – Méta-modèle pour les systèmes élastiques cloud

Afin de garantir l'élasticité des instances cloud et assurer une bonne qualité de service pour les différents clients, un système élastique cloud utilise un contrôleur d'élasticité qui gère l'approvisionnement des ressources informatiques dans le système. Le contrôleur peut changer le nombre d'instances cloud allouées à un client ou un ensemble de clients, ou augmenter/diminuer la quantité de ressources acquises par une ou plusieurs instances données. Il se base sur des informations d'observation récupérées par des sondes associées aux instances cloud afin de déclencher des actions d'élasticité réalisées par des actionneurs. La structure du contrôleur d'élasticité montrée dans notre méta-modèle est constituée de trois composants partagent une base de connaissances : un moniteur, un évaluateur et un effecteur.

Le méta-modèle défini se concentre sur l'identification des éléments conceptuels relatifs aux architectures des systèmes élastiques cloud ce qui permet de simplifier considérablement le passage vers les systèmes réactifs bigraphiques. Néanmoins, contrairement aux méthodes formelles et plus spécifiquement les BRS, ce modèle semi formel ne permet pas de définir la sémantique comportementale ou réaliser des vérifications de propriétés ce qui constituent le but derrière la spécification des systèmes élastiques cloud.

5.3 Sémantique basée BRS d'un système élastique cloud

Dans notre approche de modélisation pour les systèmes élastiques basés cloud, nous associons une sémantique précise et rigoureuse à tous les éléments architecturaux de tels systèmes (voir Tableau 5.1). La sémantique définie se base sur les concepts des BRS qui prennent en charge d'une part l'aspect interactions entre les éléments d'un système élastique cloud et d'autre part la modularité de la spécification. Le Tableau 5.1 récapitule les éléments d'une architecture d'un système basé cloud et leurs concepts correspondants dans la théorie des systèmes réactifs bigraphiques.

Élément	Sémantique bigraphique
Système élastique basé cloud	Bigraphe : $CS = (V_{CS}, E_{CS}, ctrl_{CS}, CS^P, CS^L)$: $I_F \| I_B \| I_E \rightarrow J_F \| J_B \| J_E$
Front-end	Bigraphe :$F = (V_F, E_F, ctrl_F, F^P, F^L) : I_F \rightarrow J_F$
Back-end	Bigraphe : $B = (V_B, E_B, ctrl_B, B^P, B^L) : I_B \rightarrow J_B$
Contrôleur d'élasticité	Bigraphe : $E = (V_E, E_E, ctrl_E, E^P, E^L) : I_E \rightarrow J_E$
Entité	Nœud : $v \in V_{CS}/V_{CS} = V_F \uplus V_B \uplus V_E$
Identité de nœud	Contrôle : $k \in \mathcal{K}_{CS}/\mathcal{K}_{CS} = \mathcal{K}_F \uplus \mathcal{K}_B \uplus \mathcal{K}_E$
Interaction	Hyper-arc : $e \in E_{CS}/E_{CS} = E_F \uplus E_B \uplus E_E$
Entité abstraite	Site : $s \in S_{CS}/S_{CS} = S_F \uplus S_B \uplus S_E$

TABLE 5.1 – Sémantique bigraphique des éléments d'un système élastique basé cloud

Nous modélisons un système élastique basé cloud par un bigraphe CS composé de trois bigraphes élémentaires indépendants : F modélisant de le front-end du

système élastique basé cloud, B décrivant son back-end et E pour représenter le contrôleur d'élasticité. La composition des trois bigraphes est réalisée en utilisant le produit parallèle ($||$). Il représente un terme bigraphique obtenu en juxtaposant deux ou plusieurs bigraphes et en fusionnant leurs noms communs. Les autres aspects structurels qui existent dans un système élastique basé cloud sont définis par des nœuds. Leurs différents types d'interactions sont modélisées par des liens. Les sites sont utilisés pour représenter des entités du modèle qui ont été dissimulées (parties abstraites du modèle).

L'utilisation du produit parallèle pour la composition des divers bigraphes accorde une haute expressivité, et lucidité dans la modélisation des systèmes élastiques basé cloud. Cette opération permet d'exprimer chaque partie d'un système élastique cloud séparément et indépendamment des autres parties, ce qui garantit un traitement efficace de la complexité de tels systèmes. En outre, le produit parallèle favorise l'évolutivité du modèle lui-même en permettant l'intégration et la composition facile d'autres bigraphes, modélisant par exemple d'autres front-end ou back-end.

Nous illustrons dans la suite comment nous définissons les aspects structurels clés des systèmes élastiques basés cloud à travers des expressions bigraphiques modulaires et indépendantes. Notre but principal est de fournir une approche exhaustive et sans ambiguïté pour la modélisation des systèmes élastiques basés cloud tout en considérant la complexité du modèle. Pour réaliser cette tâche, nous exprimons les aspects essentiels en associant un ensemble de sortes pour chaque bigraphes modélisant une partie spécifique du système cloud : le front-end, le back-end et le contrôleur d'élasticité.

La discipline de typage associée au bigraphe CS modélisant un système élastique basé cloud est définie par : $\Sigma_{CS} = (\Theta_{CS}, \mathcal{K}_{CS}, \Phi_{CS})$, où $\Theta_{CS} = \Theta_F \uplus \Theta_B \uplus \Theta_E$ est un ensemble non-vide de sortes de CS donné par l'union disjointe des ensembles de sortes du front-end F, back-end B et contrôleur d'élasticité E. $\mathcal{K}_{CS} = \mathcal{K}_F \uplus \mathcal{K}_B \uplus \mathcal{K}_E$ est une signature Σ_{CS}-typée obtenue par l'union disjointe des signatures de F, B et E. $\Phi_{CS} = \Phi_F \uplus \Phi_B \uplus \Phi_E$ est un ensemble non-vide de règles de formation constitué par l'union disjointe des ensembles de règles de formation de F, B et E.

Dans notre modèle, nous représentons chaque entité de l'architecture cloud avec un nœud imbriqué dans une région bien précise. Les interactions entre les différentes entités sont modélisées par des liens (hyper-arcs, noms internes et noms externes). Les sortes et les contrôles utilisés pour coder les différents bigraphes qui forment le bigraphe principale CS sont indiqués dans le Tableau 5.2. Nous notons qu'un nœud avec une sorte disjonctive peut est écrit par$\widehat{\ }$. Par exemple, un nœud avec une sorte disjonctive $\widehat{be}$ signifie qu'il peut être de sorte b ou e.

Afin de garantir que seulement des bigraphes avec une structure significative peuvent être conçus, nous définissons un ensemble de règles de typage Φ_{CS} avec des conditions $\Phi_i, 0 \leq i \leq 24$ imposant des restrictions sur la construction des bigraphes qui modélisent des systèmes élastiques basés cloud (voir Tableau 5.3).

Description	Contrôle	Arité	Sorte	Bigraphe	Notation graphique
Utilisateur final	EU	2	c	F	Cercle
Utilisateur final avec des requêtes	EU_R	2	c	F	Cercle
Développeur	D	2	c	F	Cercle
Développeur avec des requêtes	D_R	2	c	F	Cercle
Requête	R^t	0	q	F	Triangle
Requête	R^t	0	q	B	Triangle
Équilibreur de charge	LB^a	3	b	B	Cercle
Équilibreur de charge sous-utilisé	LB^a_U	3	b	B	Cercle
Équilibreur de charge chargé	LB^a_L	3	b	B	Cercle
Serveur	SE	3	e	B	Rectangle arrondi
Serveur sous-utilisé	SE_U	3	e	B	Rectangle arrondi
Serveur chargé	SE_L	3	e	B	Rectangle arrondi
Machine virtuelle	VM	3	v	B	Rectangle arrondi
Machine virtuelle sous-utilisée	VM_U	3	v	B	Rectangle arrondi
Machine virtuelle chargée	VM_L	3	v	B	Rectangle arrondi
Conteneur	CT	3	t	B	Rectangle arrondi
Conteneur sous-utilisé	CT_U	3	t	B	Rectangle arrondi
Conteneur chargé	CT_L	3	t	B	Rectangle arrondi
Service (application)	$S^{p,a}$	3	s	B	Cercle
Ressources	R	1	r	B	Cercle
Mémoire	M	1	r	B	Losange
CPU	CU	1	r	B	Losange
Stockage	SR	1	r	B	Losange
Bande passante	B	1	r	B	Losange
Actionneur	A	2	i	B	Hexagone
Sonde (capteur)	SN	2	i	B	Hexagone
Évaluateur	EV	1	o	E	Cercle
Moniteur	MO	2	m	E	Cercle
Effecteur	E	2	f	E	Cercle

TABLE 5.2 – Contrôles et sortes du bigraphe CS

	Description de la règle
Φ_0	Tous les fils d'une 0-région ont une sorte 0, où $0 \in \{c, q\}$
Φ_1	Tous les fils d'un c-nœud ont une sorte q
Φ_2	Tous les nœuds de c-sorte sont actifs
Φ_3	Tous les nœuds de q-sorte sont atomiques
Φ_4	Dans un c-nœud, un port est toujours lié à un r-nom et l'autre peut être lié à un ou plusieurs nœuds de s-sorte du bigraphe back-end B
Φ_5	Tous les fils d'une 1-région ont une sorte 1, où $1 \in \{b, e, v, t, s, r, i, q\}$
Φ_6	Tous les fils d'un b-nœud ont une sorte q du bigraphe front-end F
Φ_7	Tous les fils d'un e-nœud ont une sorte $\widehat{vtri}$
Φ_8	Tous les fils d'un v-nœud ont une sorte $\widehat{sri}$
Φ_9	Tous les fils d'un s-nœud ont une sorte q
Φ_{10}	Tous les nœuds de $\widehat{bevts}$-sorte actifs
Φ_{11}	Tous les nœuds de $\widehat{ri}$-sorte sont atomique
Φ_{12}	Dans un b-nœud, un port est toujours lié à un r-nom, le deuxième port peut être lié aux e-nœuds et le dernier port peut être lié aux i-nœuds
Φ_{13}	Dans un e-nœud, un port peut être lié aux b-nœuds, le deuxième port peut être lié aux v-nœuds et le dernier port peut être lié aux $\widehat{ri}$-nœuds
Φ_{14}	Dans un v-nœud, un port peut être lié aux e-nœuds, le deuxième port peut être lié aux s-nœuds et le dernier port peut être lié aux $\widehat{ri}$-nœuds
Φ_{15}	Dans un s-nœud, un port peut être lié aux v-nœuds, le deuxième port peut être lié aux i-nœuds et le dernier port peut être lié à un ou plusieurs c-nœuds du bigraphe front-end F
Φ_{16}	Un nœud de r-sorte peut être lié aux $\widehat{evt}$-nœuds
Φ_{17}	Un nœud de i-sorte peut être lié à un ou plusieurs $\widehat{bevts}$-nœuds
Φ_{18}	Tous les fils d'une 2-région ont une sorte 2, où $2 \in \{o, m, f\}$
Φ_{19}	Tous les nœuds de $\widehat{omf}$-sorte sont atomiques
Φ_{20}	Un nœud de m-sorte est toujours lié à un i-nom
Φ_{21}	Un nœud de f-sorte est toujours lié à un a-nom
Φ_{22}	Un nœud de o-sorte peut être lié aux $\widehat{mf}$-nœuds
Φ_{23}	Un nœud de m-sorte peut être lié aux $\widehat{of}$-nœuds
Φ_{24}	Un nœud de f-sorte peut être lié aux $\widehat{om}$-nœuds

TABLE 5.3 – Règles de typage Φ_{CS} pour le bigraphe CS

5.3.1 Bigraphe front-end

Les contrôles utilisés pour définir des bigraphe de type front-end F sont organisés en deux sortes $c = \{EU, EU_R, D, D_R\}$ et $q = \{R^t\}$ représentant respectivement les clients du cloud et de leurs requêtes (voir partie 1 du Tableau 5.2). Les requêtes constituent la charge de travail d'entrée dans le système élastique basé cloud. Les clients sont classés en deux catégories, des utilisateurs finaux représentés par les contrôles EU et EU_R, et des développeurs qui peuvent être codés par les contrôles D et D_R. Un client peut être soit inactif (EU, D) ou demandeur (EU_R, D_R). Tous les nœuds d'une c-sorte sont imbriqués dans la même 0-région. Ils ont deux ports. Un port est toujours connecté à un r-nom, utilisé pour interagir avec un back-end d'un système cloud. Le deuxième port peut être lié à un ou plusieurs services (applications) représentés par des nœuds de s-sorte. Les requêtes des clients sont modélisées par des nœuds de contrôle R^t imbriqués au sein de leur client demandeur. Le paramètre t est utilisé pour enregistrer le temps d'acheminement de n'importe quelle requête donnée. Les nœuds de c-sorte sont actifs (objet aux réactions) alors que les nœuds de q-sorte sont atomiques (nœuds vides qui ne peuvent pas héberger d'autres nœuds). Ces différentes contraintes sur le bigraphe F sont exprimées par les règles Φ_0-Φ_4 (voir partie 1 du Tableau 5.3)

Définition 5.1 (bigraphe F). La définition formelle d'un bigraphe F modélisant la partie front-end d'un système élastique basé cloud est donnée par : $F = (V_F, E_F, ctrl_F, F^P, F^L) : I_F \to J_F$.

- V_F et E_F sont des ensembles de nœuds et d'hyper-arcs du bigraphe F représentant respectivement les clients du cloud et leurs interactions potentielles avec le système cloud.
- $ctrl_F : V_F \to \mathcal{K}_F$ une fonction de transformation qui associe à chaque nœud $v \in V_F$ un contrôle $k \in \mathcal{K}_F$. La signature $\mathcal{K}_F$ est un ensemble fini de contrôles associés au front-end F.
- $F^P = (V_F, ctrl_F, prnt_F) : m_F \to n_F$ est le graphe de places de F. $prnt_F : m_F \uplus V_F \to V_F \uplus n_F$ est une fonction de parenté qui indique les localités spatiales des clients. m_F and n_F sont le nombre de site et de régions du bigraphe F.
- $F^L = (V_F, E_F, ctrl_F, link_F) : X_F \to Y_F$ représente le graphe de liens du bigraphe F, où $link_F : X_F \uplus P_F \to E_F \uplus Y_F$ est une fonction de transformation qui spécifie les interactions actuelles des clients, X_F and Y_F sont respectivement les noms internes et les noms externes et P_F est l'ensemble de ports de F.
- I_F and J_F sont les interfaces internes et externes du bigraphe front-end F.

La discipline de typage $\Sigma_F = (\Theta_F, \mathcal{K}_\mathcal{F}, \Phi_F)$ est associée au bigraphe F sachant que Θ_F est un ensemble non-vide de sortes de F, $\mathcal{K}_\mathcal{F}$ est la signature de F et Φ_F est un ensemble non-vide de règles de formation de F.

Exemple. Un exemple d'un bigraphe F représentant le front-end d'un système élastique basé cloud est donné dans la Figure 5.3. Il décrit deux clients, un développeur inactif D et un utilisateur final EU_R avec une nouvelle requête R^t. Les carrés

en gris dans la Figure 5.3, sont des sites. Ils modélisent des parties du modèle qui ont été dissimulées. L'interface du bigraphe F est donné par : $\langle m_F, \{\emptyset\}\rangle \rightarrow \langle 1, \{r\}\rangle$.

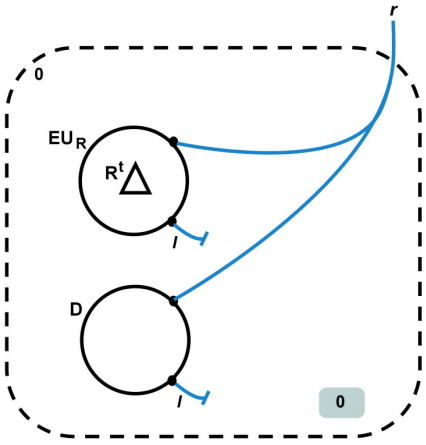


FIGURE 5.3 – Un exemple d'un bigraphe front-end F

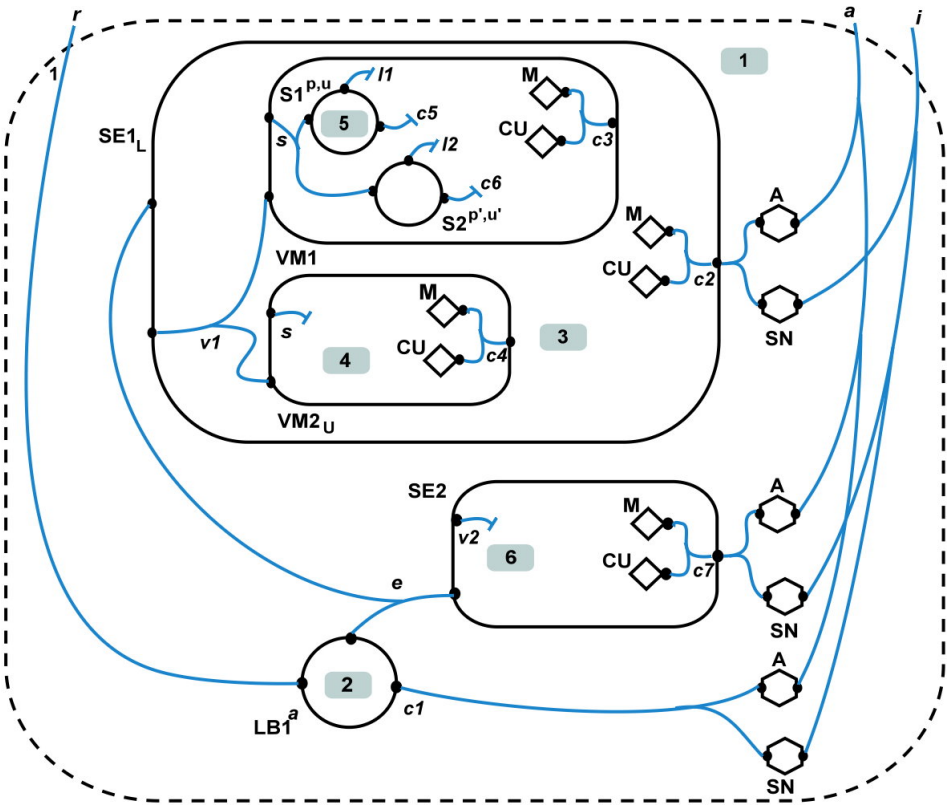


FIGURE 5.4 – Un exemple d'un bigraphe back-end B

5.3.2 Bigraphe back-end

Le deuxième bigraphe B modélisant la partie back-end d'un système élastique basé cloud peut être représenté en utilisant les sortes suivantes : $q = \{R^t\}$, $b = \{LB^a, LB^a_U, LB^a_L\}$, $e = \{SE, SE_U, SE_L\}$, $v = \{VM, VM_U, VM_L\}$, $t = \{CT, CT_U, CT_L\}$, $s = \{S^{p,a}\}, r = \{R, M, CU, SR, B\}, i = \{A, SN\}$. Les contrôles de sortes b, e, v et t sont utilisés pour modéliser respectivement des équilibreurs de charge, des serveurs, des machines virtuelles et des conteneurs (voir partie 2 du Tableau 5.2).

Nous utilisons les contrôles LB^a_U, SE_U, VM_U, CT_U pour représenter un composant cloud sous-utilisé, tandis que les contrôles LB^a_L, SE_L, VM_L, CT_L sont utilisés pour exprimer un composant cloud chargé ou défaillant. Le contrôle $S^{p,a}$, sert à représenter un services web ou une application donné hébergé au système cloud. Le paramètre a est une abstraction pour la puissance de calcul acquis actuellement par un équilibreur de charge LB_a ou un service $S^{p,a}$,a donné. Le paramètre p représente le temps de traitement actuel d'un service ou d'une application cloud.

La r-sorte regroupe les contrôles utilisés pour la modélisation des ressources informatiques, où les contrôles M, CU, SR et B codent respectivement la mémoire, la CPU, la capacité de stockage et la bande passante associé a une ressource cloud. Le contrôle R peut être utilisé comme une abstraction qui représente des différents contrôles d'une r-sorte. Enfin, les nœuds de contrôle A et SN indique respectivement des actionneurs et des sondes (capteurs). Les contraintes sur la structure du bigraphe B sont exprimés par les règles Φ_5-Φ_{17} (voir partie 2 du Tableau 5.3). En particulier, les règles Φ_5-Φ_9 définit les contraintes sur l'imbrication des nœuds, alors que les règles Φ_{10}-Φ_{11} décrit l'activité des contrôles. Enfin, les restrictions sur les liens sont exprimées par les règles Φ_{12}-Φ_{17}.

Définition 5.2 (bigraph B). Un bigraphe B modélisant un back-end d'un système élastique basé cloud est défini par $B = (V_B, E_B, ctrl_B, B^P, B^L) : I_B \to J_B$.

- V_B and E_B sont des ensembles de nœuds et d'hyper-arcs du back-end B. Ils représentent respectivement les éléments d'une architecture cloud and leurs différentes interactions. $ctrl_B$ est la fonction de transformation associé au bigraphe B.
- B^P et B^L sont respectivement le graphe de places et le graphe de liens de B.

La discipline de typage associée à un bigraphe B qui modélise le back-end d'un système élastique cloud est définie par : $\Sigma_B = (\Theta_B, \mathcal{K}_\mathcal{B}, \Phi_B)$.

Exemple. La Figure 5.4 représente une instance simplifiée d'un bigraphe B modélisant un back-end d'un système élastique basé cloud. Elle montre une architecture cloud composée d'un équilibreur de charge $LB1^a$, deux serveurs $SE1_L$ et $SE2$, deux machines virtuelles $VM1$ et $VM2_L$ hébergées dans le même serveur $S1_L$ et deux services $S1^{p,a}$ et $S2^{p',a'}$ déployés ensemble sur la machine virtuelle $VM1$. Les nœuds de contrôles M et CU représentent des ressources liées à leur propre composant cloud. En outre, une sonde SN et un actionneur A sont liés à chaque composant cloud. Leur rôle est de surveiller les performances de chaque ressource et appliquer les politiques d'élasticité. $\langle m_B, \{\emptyset\}\rangle \to \langle 1, \{r, a, i\}\rangle$ est l'interface du

bigraphe B. Les sites 1-5 sont des abstractions exprimant des nœuds qui ont été dissimulés. Par exemple, 1-site peut représenter des nœuds de $\widehat{bei}$-sorte.

5.3.3 Bigraphe contrôleur d'élasticité

Dans les systèmes élastiques basés cloud, l'approvisionnement des ressources peut être ajusté par un contrôleur d'élasticité. En particulier, le contrôleur décide les actions d'élasticité à déclencher afin de maintenir une bonne qualité de service dans un système cloud. Les décisions sont basées sur plusieurs facteurs tels que les ressources disponibles, la charge de travail actuelle, l'état du système, etc.

Dans notre modèle, nous adoptons la boucle de contrôle autonomique MAPE-K (Monitor, Analyse, Plan, Execute) de IBM pour définir la structure d'un contrôleur d'élasticité représenté par un bigraphe E. La structure proposée de notre contrôleur d'élasticité est constituée de différentes entités interagissant les uns avec les autres : une entité *Moniteur* pour la tâche d'observation. Elle représentant l'entité qui collecte et filtre les informations reçues des sondes (capteurs) embarquées dans le back-end. La tâche d'exécution est représentée par une entité nommée *Effecteur*. Il génère des signaux de contrôle pour les actionneurs afin qu'ils exécutent des actions d'élasticité selon un plan prédéfini. Pour les tâches d'analyse et de planification, nous proposons l'entité *Évaluateur*. Le rôle de cette entité est d'analyser les informations récupérées de moniteur, et sur cette base, elle génère un plan d'actions d'élasticité.

Dans notre travail, les différentes tâches des entités constituant le contrôleur d'élasticité sont considérées à un haut niveau d'abstraction dont les interactions se produisent en arrière-plan (niveau d'abstraction inférieur). Un bigraphe E peut être défini en utilisant les sortes : $o = \{EV\}, m = \{MO\}$ et $f = \{E\}$ (voir partie 3 du Tableau 5.2), où le contrôle EV (évaluateur) est une abstraction pour les tâches d'analyse et de planification, le contrôle MO (moniteur) représente la tâche d'observation et le contrôle E (effecteur) exprime la tâche d'exécution. Les différents nœuds d'un contrôleur d'élasticité sont liés ensemble par un k-hyperarc représentant les connaissances partagées entre les composants de la boucle de contrôle. Le contrôleur d'élasticité communique avec le back-end du système cloud à travers les nœuds de contrôle SN (sondes) et A (actionneurs) qui lui sont liés par les $\widehat{in}$-noms. Les contraintes sur la structure du bigraphe E sont codée par les règles Φ_{18}-Φ_{24} (voir partie 3 du Tableau 5.3).

Définition 5.3 (bigraphe contrôleur d'élasticité E). Nous définissons un bigraphe E modélisant le contrôleur d'élasticité d'un système cloud par $E = (V_E, E_E, ctrl_E, E^P, E^L) : I_E \to J_E$. Sa discipline de typage associée est donnée par : $\Sigma_E = (\Theta_E, \mathcal{K}_\mathcal{E}, \Phi_E)$.

Exemple. Un exemple d'un bigraphe E est représenté dans la Figure 5.5. L'interface pour un bigraphe E est donnée par $\langle m_E, \{\emptyset\}\rangle \to \langle 1, \{a, i\}\rangle$.

Après avoir défini les trois bigraphes de base modélisant les entités front-end, back-end et contrôleur d'élasticité d'un système cloud, nous donnons à présent la définition formelle d'un bigraphe CS complet représentant un système élastique cloud.

Définition 5.4 (bigraphe CS). Un bigraphe CS modélisant un système élas-

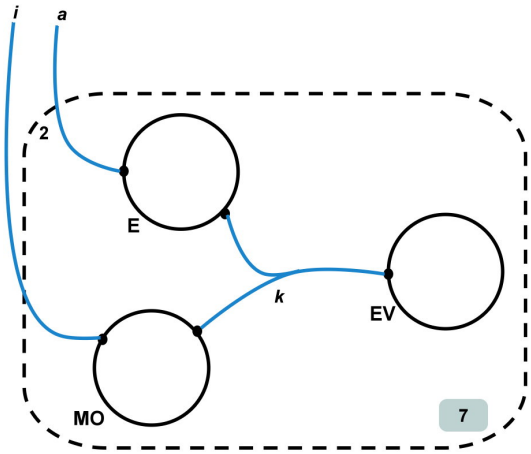


FIGURE 5.5 – Un exemple d'un bigraphe contrôleur d'élasticité E

tique basé cloud est défini formellement par $CS \stackrel{\text{def}}{=} F_r||B_{rai}||E_{ai}$, où F, B et E représentent respectivement des bigraphes spécifiant les parties front-end, back-end et contrôleur d'élasticité d'un système cloud.

$$CS = (V_{CS}, E_{CS}, ctrl_{CS}, CS^P, CS^L) : I_{CS} \to J_{CS}, \text{ où}$$

- $V_{CS} = V_F \uplus V_B \uplus V_E$ est un ensemble fini de nœuds dans le système élastique basé cloud CS donné par l'union disjointe des ensembles de nœuds du front-end F, back-end B et contrôleur d'élasticité E.
- $E_{CS} = E_F \uplus E_B \uplus E_E$ est un ensemble fini d'hyper-arcs constitué par l'union disjointe des ensembles d'hyperarcs du front-end F, back-end B et contrôleur d'élasticité E.
- $ctrl_{CS} = ctrl_F \uplus ctrl_B \uplus ctrl_E : V_{CS} \to \mathcal{K}_{CS}$, est une fonction de transformation qui associe à chaque nœud $v \in V_{CS}$ du bigraphe CS représentant le système élastique basé cloud, un contrôle $k \in \mathcal{K}_{CS}$ qui définit son identité. La signature $\mathcal{K}_{CS} = \mathcal{K}_F \uplus \mathcal{K}_B \uplus \mathcal{K}_E$ est un ensemble fini de contrôles associés au bigraphe CS.
- $CS^P = F^P||B^P||E^P$ est le graphe de places de CS donné par le produit parallèle des graphes de places de F, B et E.
- $CS^L = F^L||B^L||E^L$ est le graphe de liens de CS. Il représente le produit parallèle des graphes de liens du front-end F, back-end B, contrôleur d'élasticité E.
- $I_{CS} = I_F||I_B||I_E = \langle m_{CS}, X_{CS} \rangle, m_{CS} = m_F + m_B + m_E$ and $X_{CS} = X_F \cup X_B \cup X_E$, représente l'interface interne du bigraphe CS. m_{CS} est le nombre de sites et X_{CS} est l'ensemble de noms internes de CS.
- $J_{CS} = J_F||J_B||J_E = \langle n_{CS}, Y_{CS} \rangle, n_{CS} = n_F + n_B + n_E$ and $Y_{CS} = Y_F \cup Y_B \cup Y_E$, est l'interface externe de CS, où n_{CS} et Y_{CS} sont respectivement le nombre de régions et l'ensemble de noms externes de CS.

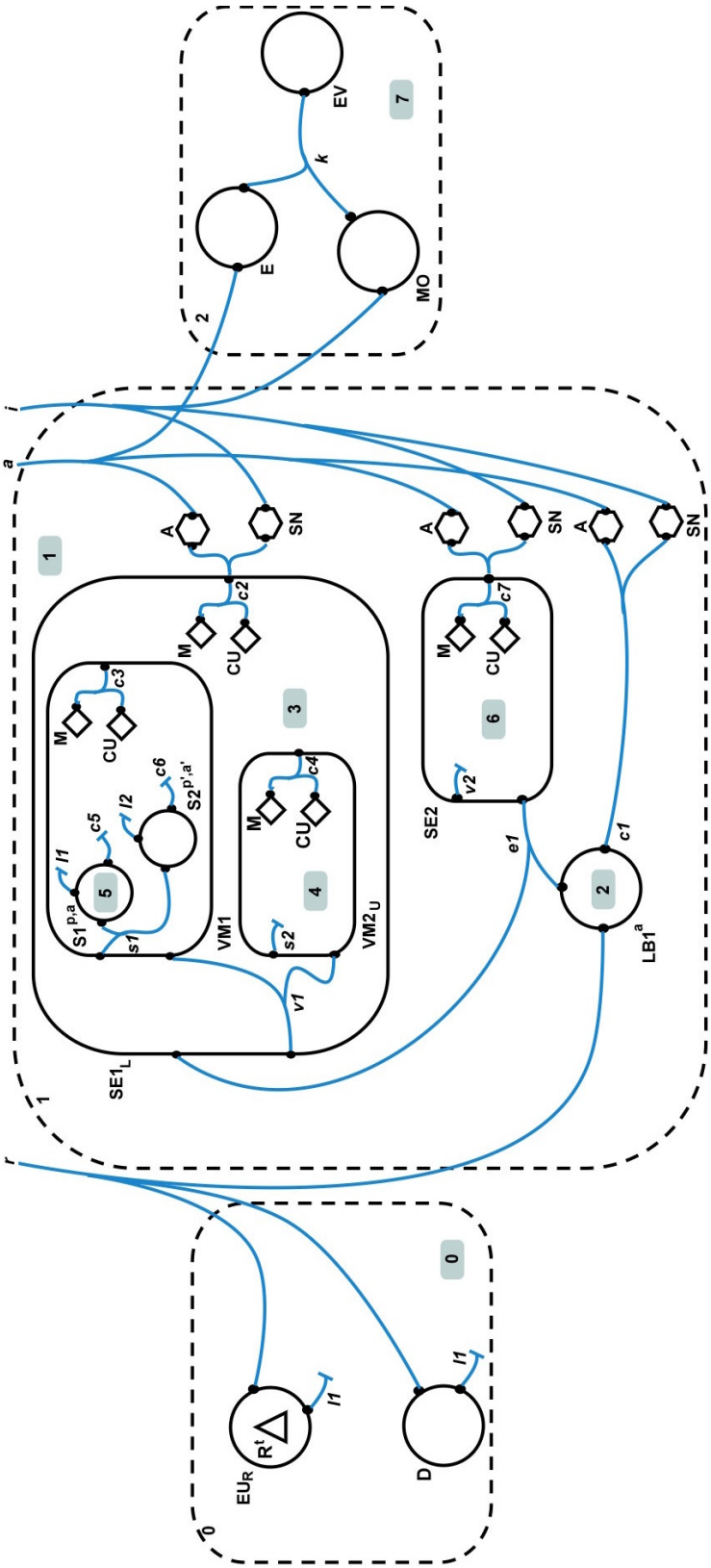


FIGURE 5.6 – Le bigraphe CS résultant du produit parallèle de F, B et E

Exemple. Un bigraph CS qui modélise un système élastique basé cloud avec une interface $\langle m_{CS}, \{\emptyset\}\rangle \rightarrow \langle 3, \{r, a, i\}\rangle$ est obtenue par la composition des trois bigraphs F, B et E en utilisant le produit parallèle ($||$). Considérons les exemples précédents (voir la Figure 5.3, 5.4 et 5.5) leur produit parallèle est montré dans la Figure 5.6. Les trois bigraphes F, B et E sont juxtaposés et liés ensemble à travers leurs noms communs r, a et i, où F et B sont liés par le nom r et B et E par les noms a et i.

5.4 Conclusion

Dans ce chapitre, nous avons présenté une approche de modélisation exhaustive et formelle qui aide à réduire la complexité de la modélisation et de la conception des architectures des systèmes élastiques basés cloud. En particulier, nous avons adopté les systèmes réactifs bigraphiques et leur discipline de typage comme un cadre formel pour la spécification des aspects structurels de ce type de systèmes autonomiques. Le modèle bigraphique présenté est assez générique et modulaire, il permet d'exprimer tous les éléments d'une architecture cloud. En outre, la discipline de typage associée aux bigraphes conçus décrit la sémantique des systèmes élastiques basés cloud en détails et avec une grande précision. Les différents aspects structurels relatifs aux systèmes élastiques basés cloud ont été formalisés en utilisant trois bigraphes indépendants modélisant les différentes parties du système, respectivement le front-end, le back-end et le contrôleur d'élasticité. Ces bigraphes peuvent être composés en utilisant l'opération bigraphique du produit parallèle.

Dans le chapitre suivant, nous nous intéressons aux aspects comportementaux de ce type de systèmes ainsi formalisés.

Chapitre 6

Gestion et planification d'élasticité : approche formelle

Sommaire

6.1 Introduction 83
6.2 Modélisation des interactions front-end/back-end 84
6.3 Modélisation des méthodes d'élasticité 86
6.3.1 Actions d'élasticité 86
6.3.2 Transitions d'états 90
6.4 Stratégie de l'élasticité 91
6.5 Étude de cas 93
6.5.1 Architecture du système élastique cloud 93
6.5.2 Planification de l'élasticité du système élastique cloud 94
6.6 Conclusion 97

6.1 Introduction

Le comportement dynamique des systèmes élastiques basé cloud en temps réels dépend de plusieurs facteurs complexes qui se chevauchent tels que les fluctuations de la charge de travail, l'infrastructure ou la plate-forme cloud fournissant les ressources informatiques, l'état actuel du système et la logique qui contrôle l'allocation de ressources. Par conséquent, la modélisation du comportement dynamique des systèmes élastique basés cloud est une tâche cruciale pour apporter une solution efficace au problème de gestion de l'élasticité. Il s'agit de la spécification du comportement en termes de méthodes et stratégies d'élasticité qui contrôlent la façon dont les ressources informatiques sont allouées dans le système cloud. Cette modélisation sert à déterminer les changements potentiels qui peuvent se produire dans une architecture cloud et donc interdire des évolutions non voulues afin d'éviter des comportements indésirables dans le système.

Nous illustrons dans ce chapitre comment exploiter le modèle bigraphique proposé précédemment pour décrire le comportement dynamique des systèmes élastiques basés cloud en utilisant des règles de réaction bigraphique. Tout d'abord, nous définissons une première catégorie de règles de réaction paramétriques pour modéliser les interactions possibles entre les clients et les applications dans le cloud

(interactions front-end/back-end) tout en respectant le typage Σ_{CS}. Une autre catégorie de règles de réaction aussi importante que la première, est définie pour décrire le comportement élastique des systèmes basés cloud en termes de méthodes d'élasticité, y compris les différentes actions utilisées pour fournir l'élasticité dans les environnements cloud computing à différents niveaux (service, plate-forme, infrastructure et répartition de charge). En outre, une troisième catégorie est définie pour contrôler la nature non-déterministe des règles de réaction. Ensuite, nous couplons les règles de réaction définies avec une logique spatiale appelée BiLog [Conforti 2006] pour modéliser les stratégies d'élasticité. Enfin, nous illustrons notre approche bigraphique pour la spécification des systèmes élastique basés cloud et leur comportement à travers une étude de cas d'un système de réservation de billets qui s'exécute sur une infrastructure Amazon EC2 (Amazon Elastic Compute).

6.2 Modélisation des interactions front-end/back-end

Le comportement d'un système élastique cloud pourrait être observé à travers les interactions qui peuvent exister entre ses deux parties front-end et back-end en termes d'interactions entre les clients et les applications dans le système cloud. Dans notre approche, nous modélisons ces interactions par une première catégorie de règles de réaction faisant intervenir les deux bigraphes correspondants (F et B). Ces règles de réaction paramétriques sont récapitulées dans le Tableau 6.1 en termes de leurs formes algébriques. Une règle de réaction paramétrique prend la forme : $R_i(p) : R \rightarrow R'$, où i est l'indice de la règle, p est une liste de paramètres, R est le redex et R' est le reactum.

Types d'interaction	Règles de réaction
Demande de service	$R1(t) \stackrel{\text{def}}{=} EU_r\|d\|\|B_{rai}\|\|E_{ai} \rightarrow EU_{Rr}.(R^t)\|d\|\|B_{rai}\|\|E_{ai}$
Envoie d'une requête	$R2(t,t',a) \stackrel{\text{def}}{=} EU_{Rr}.(R^t)\|d\|\|B_{rai}\|\|E_{ai} \rightarrow EU_{Rr}\|d\|\|LB^a_r.(R^{t'}\|d')\|id_{ai}\|\|E_{ai}$
Transfert d'une requête	$R3(t,t',p,a,a') \stackrel{\text{def}}{=} F_r\|\|LB^a_r.(R^t\|d)\|id_{ai}\|\|E_{ai} \rightarrow$ $F_r\|\|LB^a_{re}.(d)\|SE_{ev}.(VM_{vs}.(S^{p,a'}_s.(R^{t'}\|d'''\|d'')\|d')\|id_{ai}\|\|E_{ai}$
Allocation de service	$R4(t,p,a) \stackrel{\text{def}}{=} EU_{Rr}\|d\|\|SE_v.(VM_{vs}.(S^{p,a}_s.(R^t\|d''')\|d'')\|d')\|id_{rai}\|\|E_{ai} \rightarrow$ $EU_{Rrl}\|d\|\|SE_v.(VM_{vs}.(S^{p,a}_{sl}.(R^t\|d''')\|d'')\|d')\|id_{rai}\|\|E_{ai}$
Libération d'un service	$R5(t,p,a) \stackrel{\text{def}}{=} EU_{Rrl}\|d\|\|SE_v.(VM_{vs}.(S^{p,a}_{sl}.(R^t\|d'')\|d'')\|d')\|id_{rai}\|\|E_{ai} \rightarrow$ $EU_{Rr}\|d\|\|SE_v.(VM_{vs}.(S^{p,a}_s.(R^t\|d''')\|d'')\|d')\|id_{rai}\|\|E_{ai}$
Déploiement de service	$R6(p,a) \stackrel{\text{def}}{=} D_{Rr}\|d\|\|SE_v.(VM_v.(d'')\|d')\|id_{rai}\|\|E_{ai} \rightarrow$ $D_{Rrl}\|d\|\|SE_v.(VM_{vs}.(S^{p,a}_{sl}\|d'')\|d')\|id_{rai}\|\|E_{ai}$
Suppression d'un service	$R7(p,a) \stackrel{\text{def}}{=} D_{Rrl}\|d\|\|SE_v.(VM_{vs}.(S^{p,a}_{sl}.(d''')\|d'')\|d')\|id_{rai}\|\|E_{ai} \rightarrow$ $D_{Rr}\|d\|\|SE_v.(VM_v.(d'')\|d')\|id_{rai}\|\|E_{ai}$

TABLE 6.1 – Règles de réaction modélisant les interactions front-end/back-end

Nous expliquons dans ce qui suit deux comportements possibles afin de faciliter

la compréhension : l'envoie d'une requête et le déploiement d'un service ou d'une application à travers leurs règles de réaction (respectivement les règles $R2$ et $R6$) illustrées graphiquement dans la Figure 6.1.

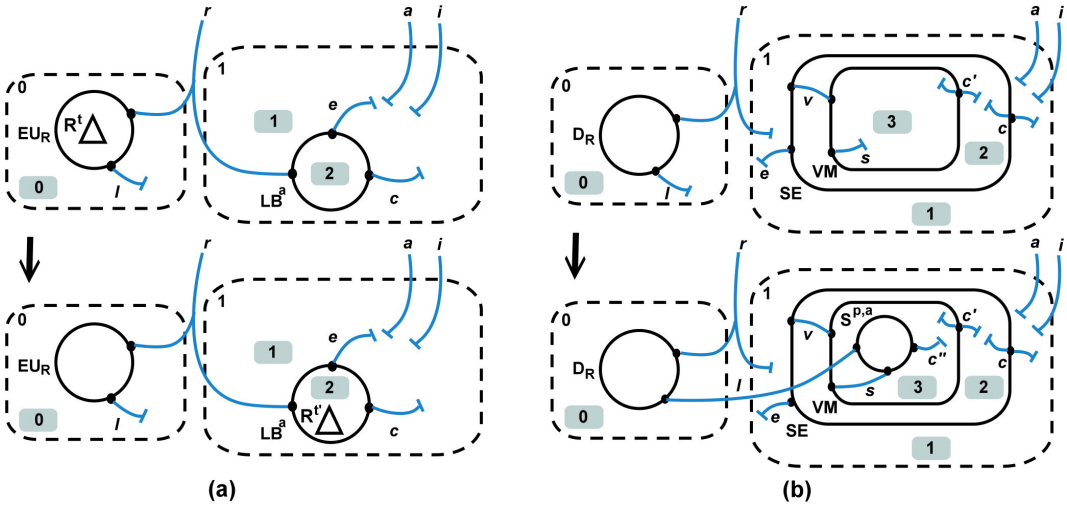


FIGURE 6.1 – Règle de réaction pour l'envoie d'une requête (a), Règle de réaction pour le déploiement de service (b)

La première règle de réaction $R2$ (voir Figure 6.1a), modélise un utilisateur final représenté par un contrôle EU_R envoyant une requête (nœud de contrôle R^t) à un back-end cloud. Sur le côté gauche de la règle (redex), l'utilisateur final est représenté par le nœud EU_R qui contient le nœud de la requête R^t avec un temps d'acheminement initial indiqué par le paramètre t. Les deux nœuds sont encore imbriquées dans la 0-région qui représente le front-end du système élastique basé cloud. Sur le côté droit de la règle (reactum), le nœud de la requête (R^t) est transféré à un équilibreur de charge imbriqué dans la 1-région qui modélise le back-end du système de cloud.

L'équilibreur de charge, spécifié par un nœud de contrôle LB^a, représente la première passe de la requête avant qu'elle soit transmise au service demandé. Noter que le temps d'acheminement de la requête R^t représenté par le paramètre t est remplacé par t' une fois que la réaction est accomplie. Les sites 0 et 1 sont utilisés pour modéliser des nœuds négligés qui ne sont pas impliqués dans la réaction.

La deuxième règle de réaction $R6$ représentée par la Figure 6.1b spécifie un développeur modélisé par un contrôle D_R prêt à déployer une application (ou un service) dans une infrastructure cloud. Précisément, le déploiement de l'application dans une machine virtuelle modélisée par un nœud de contrôle VM, imbriqué dans un nœud de contrôle SE représentant un serveur localisé dans le back-end du système cloud (1-région). L'intention du développeur est exprimé sur le côté gauche de la règle (redex) par un nœud D_R imbriqué dans la 0-région (front-end). Nous tenons à noter qu'après l'exécution de la règle (reactum de la règle), un nouveau nœud avec un contrôle $S^{p,a}$ est créé au sein de la machine virtuelle VM et lié à son développeur

D_R par un l-hyperarc modélisant leur relation.

Le comportement que nous avons formalisé par ce type de règles de réaction est lié aux changements qui peuvent subir les éléments appartenant aux entités front-end et back-end. Un système élastique cloud tel que nous l'avons défini comporte une autre entité qui est responsable de la gestion et la planification de l'élasticité. Ainsi, nous nous intéressons à ce type de comportement dans la section suivante.

6.3 Modélisation des méthodes d'élasticité

Le comportement élastique d'un système élastique cloud est centré autour des méthodes d'élasticité déclenchées en réponse aux changements de la charge de travail. Chaque méthode consiste en un ensemble d'actions d'élasticité qui peuvent être appliquées dans un niveau cloud spécifique (application, plate-forme, infrastructure et répartition de charge). Dans ce travail, nous considérons les trois méthodes utilisée pour fournir l'élasticité dans les environnements cloud : l'élasticité horizontale, l'élasticité verticale et la migration (voir Figure 6.2). La logique qui contrôle le comportement élastique des systèmes basés cloud est modélisée par le bigraphe E qui représente le contrôleur d'élasticité du système. Ce comportement peut être aussi modélisé par des règles de réaction, déclenchées par des éléments du bigraphe E, et agissant sur les états des éléments bigraphiques constituant le back-end.

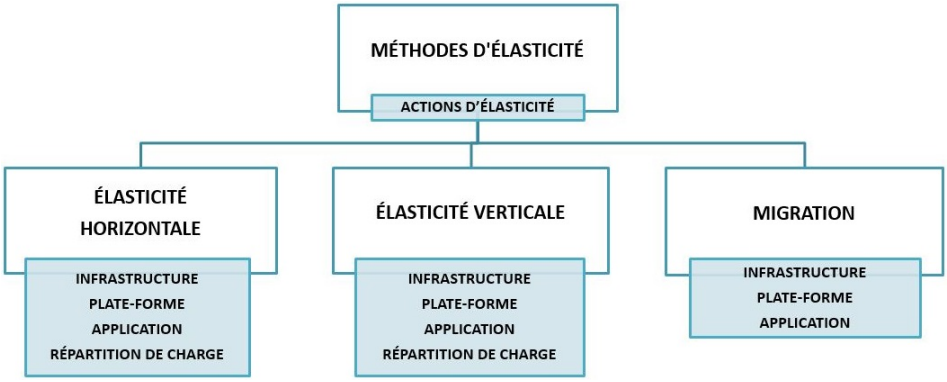


FIGURE 6.2 – Méthode d'élasticité dans un système élastique cloud

6.3.1 Actions d'élasticité

Nous modélisons les différentes méthodes d'élasticité en termes d'actions applicables aux différents niveaux cloud par une deuxième catégorie de règles de réaction. Toutes ces réglés accordant aux systèmes élastiques cloud la capacité d'augmenter/-diminuer le nombre d'instances virtuelles et la quantité de ressources allouées en temps réel sont récapitulées dans le Tableau 6.2. Nous détaillons dans la suite certaines de ces règles jugées pertinentes.

Méthode d'élasticité horizontale	
Niveau infrastructure	Réplication d'une instance de Vm
	$R8 \stackrel{\text{def}}{=} F_r \|\| SE_v.(VM_{Lv}.(d')\|d)\|id_{rai}\|\|E_{ai} \rightarrow$ $F_r \|\| SE_v.(VM_{Lv}.(d')\|VM'_v.(d'')\|d)\|id_{rai}\|\|E_{ai}$
	Consolidation d'une instance de Vm
	$R9 \stackrel{\text{def}}{=} F_r \|\| SE_v.(VM_v.(d')\|VM'_{Uv}.(d'')\|d)\|id_{rai}\|\|E_{ai} \rightarrow$ $F_r \|\| SE_v.(VM_v.(d')\|d)\|id_{rai}\|\|E_{ai}$
Niveau plate-forme	Réplication d'une instance de conteneur
	$R10 \stackrel{\text{def}}{=} F_r \|\| SE_v.(CT_{Lv}.(d')\|d)\|id_{rai}\|\|E_{ai} \rightarrow$ $F_r \|\| SE_v.(CT_{Lv}.(d')\|CT'_v.(d'')\|d)\|id_{rai}\|\|E_{ai}$
	Consolidation d'une instance de conteneur
	$R11 \stackrel{\text{def}}{=} F_r \|\| SE_v.(CT_v.(d')\|CT'_{Uv}.(d'')\|d)\|id_{rai}\|\|E_{ai} \rightarrow$ $F_r \|\| SE_v.(CT_v.(d')\|d)\|id_{rai}\|\|E_{ai}$
Niveau application	Réplication d'une instance de service
	$R12(p,a,p',a') \stackrel{\text{def}}{=} F_r \|\| SE_v.(VM_vs.(S_s^{p,a}.(d'')\|d')\|d)\|id_{rai}\|\|E_{ai} \rightarrow$ $F_r \|\| SE_v.(VM_{vs}.(S_s^{p,a}.(d'')\|S_s^{'p',a'}.(d''')\|d')\|d)\|id_{rai}\|\|E_{ai}$
	Consolidation d'une instance de service
	$R13(p,a,p',a') \stackrel{\text{def}}{=} F_r \|\| SE_v.(VM_{vs}.(S_s^{p,a}.(d'')\|S_s^{'p',a'}.(d''')\|d')\|d)\|id_{rai}\|\|$ $E_{ai} \rightarrow F_r \|\| SE_v.(VM_{vs}.(S_s^{p,a}.(d'')\|d')\|d)\|id_{rai}\|\|E_{ai}$
Niveau répartition de charge	Réplication d'une instance d'équilibreur de charge
	$R14(a,a') \stackrel{\text{def}}{=} F_r \|\| LB_r^a.(d)\|id_{ai}\|\|E_{ai} \rightarrow$ $F_r \|\| LB_r^a.(d)\|LB_r^{'a'}.(d')\|id_{ai}\|\|E_{ai}$
	Consolidation d'une instance d'équilibreur de charge
	$R15(a,a') \stackrel{\text{def}}{=} F_r \|\| LB_r^a.(d)\|LB_r^{'a'}.(d')\|id_{ai}\|\|E_{ai} \rightarrow$ $F_r \|\| LB_r^a.(d)\|id_{ai}\|\|E_{ai}$
Méthode d'élasticité verticale	
Niveau infrastructure	Augmentation/diminution des ressources allouées à une machine virtuelle
	$R16 \stackrel{\text{def}}{=} F_r \|\| SE_v.(VM_{Lvc}.(R_c\|d')\|d)\|id_{rai}\|\|E_{ai} \rightarrow$ $F_r \|\| SE_{vc}.(VM_{Lvc}.(R'_c\|d')\|d)\|id_{rai}\|\|E_{ai}$
Niveau plate-forme	Augmentation/diminution des ressources allouées à un conteneur
	$R17 \stackrel{\text{def}}{=} F_r \|\| SE_v.(CT_{Lvc}.(R_c\|d')\|d)\|id_{rai}\|\|E_{ai} \rightarrow$ $F_r \|\| SE_v.(CT_{Lvc}.(R'_c\|d')\|d)\|id_{rai}\|\|E_{ai}$
Niveau application	Augmentation/diminution des ressources allouées à un service (application)
	$R18(p,a,a') \stackrel{\text{def}}{=} F_r \|\| SE_v.(VM_{vs}.(S_s^{p,a}.(d'')\|d')\|d)\|id_{rai}\|\|E_{ai} \rightarrow$ $F_r \|\| SE_v.(VM_{vs}.(S_s^{p,a'}.(d'')\|d')\|d)\|id_{rai}\|\|E_{ai}$
Niveau répartition de charge	Augmentation/diminution des ressources allouées à un équilibreur de charge
	$R19(a,a') \stackrel{\text{def}}{=} F_r \|\| LB_r^a.(d)\|id_{ai}\|\|E_{ai} \rightarrow$ $F_r \|\| LB_r^{a'}.(d)\|id_{ai}\|\|E_{ai}$
Méthode de migration	
Niveau infrastructure	Migration de machine virtuelle
	$R20 \stackrel{\text{def}}{=} F_r \|\| SE_{Lv}.(VM_v.(d'')\|d)\|SE'.(d')\|id_{rai}\|\|E_{ai} \rightarrow$ $F_r \|\| SE_L.(d)\|SE'_{v'}.(VM_{v'}.(d'')\|d')\|id_{rai}\|\|E_{ai}$
Niveau plate-forme	Redéploiement de conteneur
	$R21 \stackrel{\text{def}}{=} F_r \|\| SE_{Lv}.(CT_v.(d'')\|d)\|SE'.(d')\|id_{rai}\|\|E_ai \rightarrow$ $F_r \|\| SE_L.(d)\|SE'_{v'}.(CT_{v'}.(d'')\|d')\|id_{rai}\|\|E_{ai}$
Niveau application	Redéploiement d'une instance de service (application)
	$R22(p,a) \stackrel{\text{def}}{=} F_r \|\| SE_v.(VM_v.(S_s^{p,a}.(d''')\|d')\|VM'_v.(d'')\|d)\|id_{rai}\|\|E_{ai} \rightarrow$ $F_r \|\| SE_v.(VM_v.(d')\|VM'_{vs'}.(S_{s'}^{p,a}.(d''')\|d'')\|d)\|id_{rai}\|\|E_{ai}$

TABLE 6.2 – Règles de réaction modélisant des actions d'élasticité

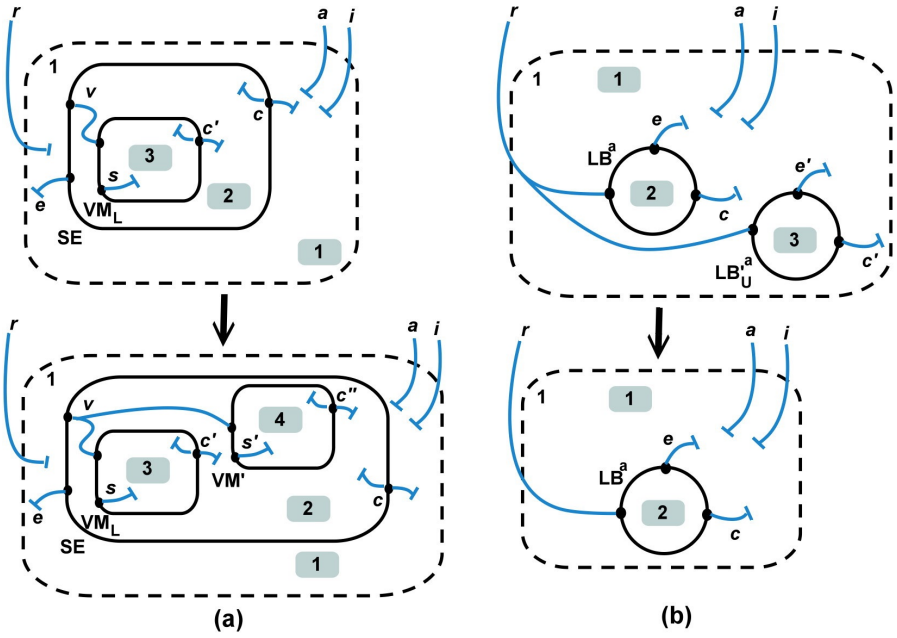


FIGURE 6.3 – Réplication d'une instance de VM (a), Consolidation d'une instance d'équilibreur de charge (b)

La règle de réaction $R8$ décrit la réplication d'une instance de machine virtuelle. Sa représentation graphique est donnée dans la Figure 6.3a. La règle peut être déclenchée lorsqu'une machine virtuelle est chargée ou défaillante. Cette situation est représentée sur le côté gauche de la règle de réaction (redex) par un nœud de machine virtuelle ayant un contrôle VM_L. Ce contrôle modélise une machine virtuelle chargée, imbriqué dans un nœud de serveur de contrôle SE. Les nœuds qui ne sont pas impliqués dans la réaction ont été représentés par les sites $1, 2$ et 3. Dans le côté droit (reactum) de la règle, la machine virtuelle chargée est répliquée en créant une nouvelle instance ayant le contrôle VM' dans le même serveur SE. La nouvelle instance de la machine virtuelle VM' peut être utilisée ultérieurement afin d'alléger la charge sur la machine virtuelle originale VM en divisant la charge entre les deux machines virtuelles VM et VM'.

La consolidation d'une instance d'un équilibreur de charge est modélisée par la règle de réaction $R6$ (voir Figure 6.3b). Cette règle peut être déclenchée lorsqu'une instance d'un équilibreur de charge est sous-utilisée ou n'est plus nécessaire à cause d'une baisse significative du trafic entrant (charge de travail diminuée). Ceci est exprimé sur le redex de la règle par un équilibreur de charge ayant un contrôle $LB_U^{a'}$. Ce nœud représente alors une copie sous-utilisée créé auparavant afin de partager une charge de travail entrante élevée avec l'instance d'équilibreur de charge original codé par le nœud de contrôle LB^a. Nous tenons à noter que les deux équilibreurs de charge avec les nœuds LB^a et $LB_U^{a'}$ sont imbriqués ensemble dans 1-région qui représente le back-end du système élastique basé cloud. Dans le bigraphe reactum, l'équilibreur de charge sous-utilisé $LB_U^{a'}$ disparaît, ce qui signifie qu'il a été consolidé.

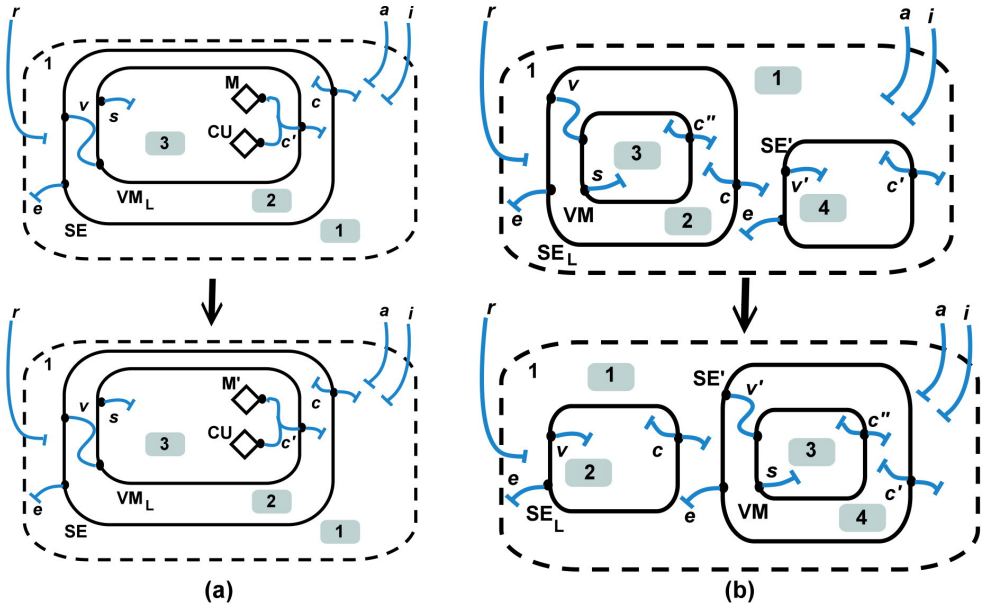


FIGURE 6.4 – Augmentation/diminution des ressources allouées à une VM (a), Migration de machine virtuelle (b)

La règle de réaction R16 permet d'augmenter/diminuer les ressources effectivement allouées à une machine virtuelle chargée (élasticité verticale). Par exemple, la règle sur la Figure 6.4a modélise l'augmentation/diminution de la mémoire allouée à une instance de machine virtuelle chargée modélisée par un nœud de contrôle VM_L et imbriqué dans un nœud de serveur SE. Les deux opérations d'augmentation et de diminution sont exprimées en remplaçant le contrôle de mémoire M par autre contrôle M'. La mémoire est considérée augmentée (scaling up) si la valeur de M' est supposée supérieure à M ($M' > M$) ou diminuée (scalling down) dans le cas contraire. Cette règle peut être appliquée sur tous les contrôles d'un r-sorte. Par exemple, l'augmentation ou la diminution de la CPU allouée à une machine virtuelle ou un conteneur peut être exprimée par le remplacement d'un contrôle CU associé à un nœud VM ou CT par à un autre contrôle CU'.

La règle de réaction $R20$ modélise la migration d'une machine virtuelle d'un serveur hôte (physique ou logique) chargé à un autre serveur moins chargé. La forme graphique de la règle est représentée sur la Figure 6.4b. Cette règle est appliquée quand un serveur est marqué avec le contrôle SE_L sur le côté gauche de la règle de réaction (redex), ce qui signifie que le serveur est actuellement chargé. Dans le côté droit (reactum), la machine virtuelle représentée par un nœud de contrôle VM est relocalisée du serveur chargé SE_L à un autre serveur SE'. Un v'-hyper-arc est créé également pour relier les deux nœuds et d'exprimer leur relation. Nous notons que les deux serveurs avec les nœuds SE_L et SE' sont imbriqués dans le même back-end du système élastique basé coud représenté par 1-région.

6.3.2 Transitions d'états

Les éléments constituant la partie back-end d'un système élastique cloud doivent changer d'état d'une situation à une autre. Par exemple, un serveur (équilibreur de charge, machine virtuelle, ou conteneur) actuellement chargé peut transiter vers un état où il sera non plus chargé. Ainsi, nous proposons dans cette sous-section une troisième catégorie supplémentaire de règles de réaction pour définir un mécanisme de marquage qui permet de transiter entre les états de certains nœuds et contrôler la nature non-déterministe des règles de réaction en imposant des préconditions sur les états des bigraphes, ce qui fournit des restrictions strictes sur l'exécution des règles. Plus précisément, ces transitions sont réalisables en marquant/démarquant quand il est nécessaire les contrôles de $\widehat{bevts}$-sorte chargés ou sous-utilisés.

Un résumé de ces règles de réaction sous leurs formes algébriques est donné dans le Tableau 6.3. Il présente les règles de réaction qui permettent de marquer un serveur, un équilibreur de charge, une machine virtuelle ou un conteneur chargé/sous-utilisé. Ces règles que nous expliquons dans la suite à travers leurs représentations graphiques, peuvent être combinées avec des règles des autres catégories pour modéliser différentes situations comportementales dans un système élastique cloud.

Marquage	Règles de réaction
Serveur chargé	$R23 \stackrel{\mathsf{def}}{=} F_r \vert\vert SE_v.(d) \vert id_{rai} \vert\vert E_a i \rightarrow F_r \vert\vert SE_{Lv}.(d) \vert id_{rai} \vert\vert E_{ai}$
Équilibreur de charge chargé	$R24(a) \stackrel{\mathsf{def}}{=} F_r \vert\vert LB^a_r.(d) \vert id_{ai} \vert\vert E_a i \rightarrow F_r \vert\vert LB^a_{rL}.(d) \vert id_{ai} \vert\vert E_{ai}$
Machine virtuelle chargée	$R25 \stackrel{\mathsf{def}}{=} F_r \vert\vert SE_v.(VM_v.(d')\vert d) \vert id_{rai} \vert\vert E_{ai} \rightarrow$ $F_r \vert\vert SE_v.(VM_{Lv}.(d')\vert d) \vert id_{rai} \vert\vert E_{ai}$
Conteneur chargé	$R26 \stackrel{\mathsf{def}}{=} F_r \vert\vert SE_v.(CT_v.(d')\vert d) \vert id_{rai} \vert\vert E_{ai} \rightarrow$ $F_r \vert\vert SE_v.(CT_{Lv}.(d')\vert d) \vert id_{rai} \vert\vert E_{ai}$
Serveur sous-utilisé	$R27 \stackrel{\mathsf{def}}{=} F_r \vert\vert SE_v.(d) \vert id_{rai} \vert\vert E_{ai} \rightarrow F_r \vert\vert SE_{Uv}.(d) \vert id_{rai} \vert\vert E_{ai}$
Équilibreur de charge sous-utilisé	$R28(a) \stackrel{\mathsf{def}}{=} F_r \vert\vert LB^a_r.(d) \vert id_{ai} \vert\vert E_{ai} \rightarrow F_r \vert\vert LB^a_{rU}.(d) \vert id_{ai} \vert\vert E_{ai}$
Machine virtuelle sous-utilisée	$R29 \stackrel{\mathsf{def}}{=} F_r \vert\vert SE_v.(VM_v.(d')\vert d) \vert id_{rai} \vert\vert E_{ai} \rightarrow$ $F_r \vert\vert SE_v.(VM_{Uv}.(d')\vert d) \vert id_{rai} \vert\vert E_{ai}$
Conteneur sous-utilisé	$R30 \stackrel{\mathsf{def}}{=} F_r \vert\vert SE_v.(CT_v.(d')\vert d) \vert id_{rai} \vert\vert E_{ai} \rightarrow$ $F_r \vert\vert SE_v.(CT_{Uv}.(d')\vert d) \vert id_{rai} \vert\vert E_{ai}$

TABLE 6.3 – Règles de réaction pour le mécanisme de marquage

La Figure 6.5a montre la représentation bigraphique de la règle $R23$ pour marquer un serveur chargé. Dans le côté gauche de la règle de réaction (redex), on peut remarquer un nœud de contrôle SE modélisant un serveur imbriqué dans la 1-région qui représente le back-end d'un système cloud. Comme il est visible sur le reactum de la règle, le mécanisme de marquage consiste simplement à remplacer le contrôle SE

par un autre contrôle SE_L représentant un serveur chargé. Une autre règle pour le marquage d'un équilibreur de charge sous-utilisé $R28$ est donnée dans la Figure 6.5b. De même que la règle précédente, le nœud d'équilibreur de charge avec un contrôle LB^a imbriqué dans la 1-région sur le redex de la règle, est remplacé sur le reactum par un nœud de contrôle LB^a_U représentant un équilibreur de charge sous-utilisé. Le reste des règles de réaction de marquage est basés sur le même principe.

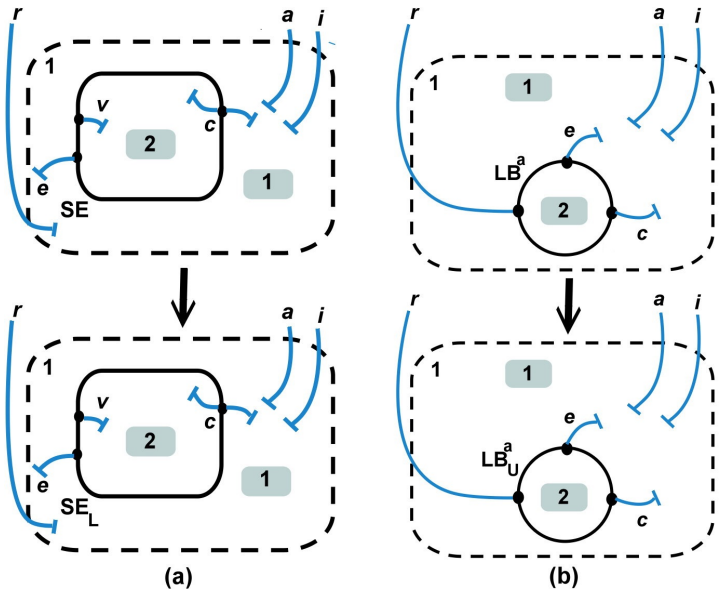


FIGURE 6.5 – Marquage d'un serveur chargé (a), Marquage d'un équilibreur de charge sous-utilisé (b)

6.4 Stratégie de l'élasticité

Nous expliquons dans cette section comment nous pouvons utiliser les règles de réaction définies dans la section précédente avec des prédicat BiLog pour formaliser le comportement global d'un contrôleur d'élasticité en termes de stratégies d'élasticité. En effet, les règles de réaction présentées peuvent servir à simuler différents scénarios d'évolution des systèmes élastiques basés cloud. Chaque scénario consiste en une séquence d'applications de règles de réaction. Par ailleurs, la logique spatiale BiLog peut être utilisée pour décrire les bigraphes à travers des opérateurs modaux qui sont capables d'exprimer les notions orthogonales de localité et de connectivité structurelle des différents termes bigraphiques de la même manière que dans l'algèbre de processus et le *pi*-calcul.

Dans un système basé cloud le comportement élastique est géré et planifié par un contrôleur. En particulier, ce contrôleur d'élasticité décide le plan d'action d'élasticité à déclencher pour ajuster les ressources matérielles allouées au système cloud d'une manière dynamique et transparente. Les décisions peuvent être basées sur plusieurs facteurs tels que les ressources disponibles, la charge de travail actuelle,

l'état du système, etc. Généralement, le comportement d'un contrôleur d'élasticité est défini par une stratégie d'élasticité, qui représente la logique assurant l'élasticité dans un système cloud donné. Une stratégie d'élasticité dans le cloud peut être basée sur une ou plusieurs méthodes d'élasticité. Elle peut être soit réactive ou proactive, à titre d'exemple, une stratégie réactive à base de seuils est de la forme suivante [Galante 2012] :

```
Strategie :
Si CONDITION(s) Alors ACTION(s)
CONDITION :
(1..*)(Si metrique.valeur = seuil)
ACTION :
(*) IaaS actions activées (exemple. replication/consolidation de Vm)
```

Nous combinons dans notre travail, les prédicats BiLog avec les règles de réactions bigraphiques afin de modéliser les stratégies d'élasticité. En plus de détails, nous modélisons les conditions d'une stratégie par des prédicat BiLog sous formes bigraphiques. En outre, les règles de réaction sont utilisées pour représenter les différents plans d'action. Chaque plan consiste (AP) en une séquence d'applications de règles de réaction.

Une stratégie d'élasticité définissant la logique de contrôle dans uns système élastique cloud prend la forme suivante : $STRAT_i : If\ CS \vDash \varphi\ Then\ AP$ où $B_\varphi \stackrel{\text{def}}{=} CS'$, sachant que φ est un prédicat BiLog, B_φ est un modèle bigraphique CS' qui représente le prédicat φ, $CS \vDash \varphi$ est vrai ssi B_φ à une correspondance dans le bigraphe CS et AP est un plan d'action d'élasticité modélisé sous la forme d'une séquence de règles de réaction.

Exemple. Pour une meilleure compréhension, nous présentons un exemple d'une stratégie d'élasticité. La stratégie opère au niveau infrastructure lorsqu'une instance d'une machine virtuelle donnée est déployée dans un serveur chargé en exécutant un plan d'action pour migrer la machine virtuelle sur autre serveur en bon état. Cette stratégie exécute les deux règles $R20$ (migration de machine virtuelle) et $R27$ (marquage d'un serveur sous-utilisé) respectivement si un prédicat spécifique représentant le serveur chargé est vérifié ($SE_{Lv}.(VM_{Lv}.(d'')|d)$), elle prend la forme suivante :

$$STRAT_1 : If\ CS \vDash \varphi\ Then\ R_{20} \rightarrow R_{27},\ B_\varphi \stackrel{\text{def}}{=} (SE_{Lv}.(VM_{Lv}.(d'')|d))$$

Une autre stratégie au niveau service (application) qui permet de répliquer une instance d'une application $R12$. La stratégie permet la réplication d'une instance de service un certain nombre de fois (par exemple, 5 fois) pour gérer un nombre élevé de requêtes (par exemple, le site d qui représente ici "100 requêtes" est de la forme :

$$STRAT_2 : If\ CS \vDash \varphi\ Then\ R^5_{12},\ B_\varphi \stackrel{\text{def}}{=} (LB^a_r.(d))$$

6.5 Étude de cas

Dans cette section, nous illustrons notre approche de modélisation basé BRS à travers une étude de cas d'un système élastique basé cloud qui s'exécute au-dessus d'une infrastructure EC2 (Amazon Elastic Compute Cloud) [Amazon EC2 2006]. Tout d'abord, nous modélisons les aspects structurels par un bigraphe CS représentant l'état initial de l'architecture du système élastique basé cloud. Ensuite, nous proposons quelques scénarios de son exécutions modélisant des différentes situations comportementales. Ces scénarios montrent l'intérêt de prendre en charge la gestion et la planification de l'élasticité dans ce type de système. Les différents scénarios d'exécutions sont modélisées sous la forme d'ensembles de séquences d'applications de règles de réaction choisies parmi celles décrites dans les sections précédentes.

Amazon Elastic Compute Cloud (Amazon EC2) est une infrastructure en tant que service (IaaS, infrastructure as a service) fournissant à la demande, des ressources informatiques redimensionnables à ses clients pour qu'ils puissent héberger et gérer leurs applications et leurs web services. Les ressources en EC2 sont fournies sous forme d'instances de calcul, en particulier des instances de machines virtuelles. Afin de maintenir une qualité de service (QoS) adéquate pour les applications déployées dans une ou plusieurs instances de VM, Amazon EC2 permet de recourir si nécessaire à une ou plusieurs méthodes d'élasticité permettant de supporter une montée potentielle de la charge de travail. Premièrement, la mise à l'échelle horizontale (élasticité horizontale) en créant des instances virtualisées supplémentaires, ou la mise à l'échelle verticale (élasticité verticale) en rajoutant des capacités de calcul (CPU, mémoire, stockage, bande passante, etc.) aux instances existantes. En outre, les machines virtuelles de EC2 peuvent migra d'un serveur à un autre pour de diverses raisons telles que la répartition de charge.

6.5.1 Architecture du système élastique cloud

Dans cette étude de cas, nous considérons un système de réservation de billets (*TicketBook*) [Rong 2014] déployé dans une infrastructure EC2 par un développeur. Ce système est composé de quatre services web atomiques : le service *AirTicketBook* (S1) pour la réservation des billets d'avion, le service *BoatTicketBook* (S2) pour les billets de bateau, le service *TrainTicketBook* (S3) pour des billets de train et le service *BusTicketBook* (S4) pour des billets de bus. Supposant que chacun deux services web sont hébergés sur la même machine virtuelle, et les machines virtuelles hébergeant ces services sont également déployées sur le même serveur. Selon notre approche de formalisation, la représentation graphique du bigraphe B modélisant le back-end d'un système élastique basé cloud est donnée dans la Figure 6.6. Cette Figure montre la structure d'une configuration initiale de back-end du système cloud.

Le nœud de contrôle $LB1^a$ modélise un équilibreur de charge. Il contient un site (site 2) qui modélise les nœuds dissimulés de contrôle R^t représentant la charge de travail actuelle dans le back-end. Par exemple, le site 2 peut représenter 1000 demandes de service (requêtes). Les contrôles $SE1, VM1$ et $VM2$ représentent respectivement un serveur et deux machines virtuelles. Le site 1 est une abstraction

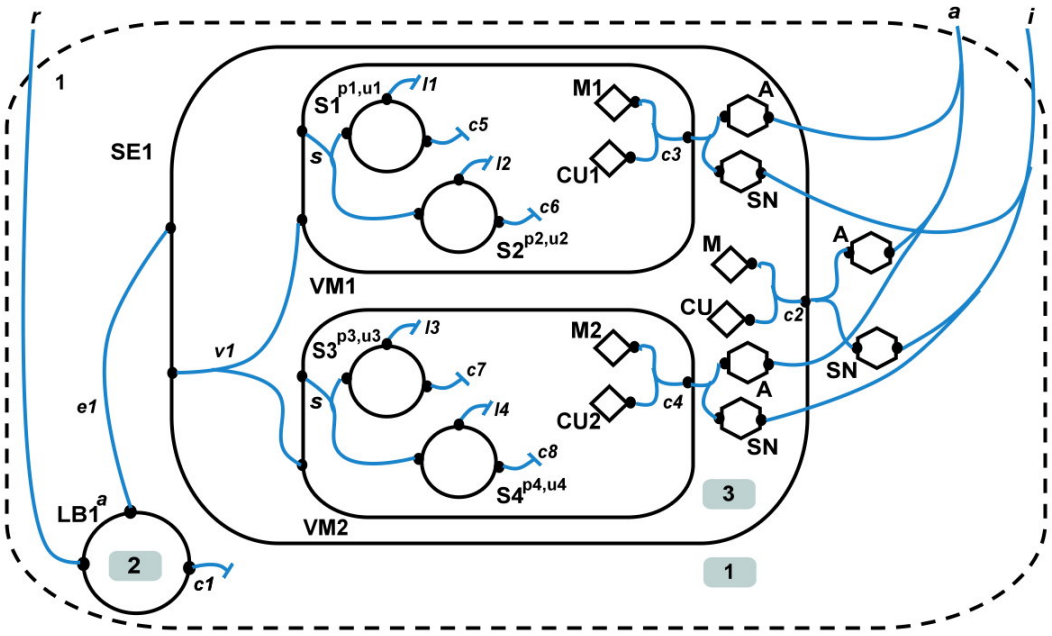


FIGURE 6.6 – Le bigraphe back-end modélisant l'architecture cloud

pour d'autres nœuds de contrôles SE et LB, alors que le site 3 représente une abstraction des nœuds de contrôle VM. Les nœuds de contrôle S ($S1, S2, S3$ et $S4$) modélisent les services web ou les applications. Les nœuds de contrôle SN et A expriment respectivement des sondes (capteurs) et actionneurs. Ils représentent les entités qui surveillent les performances des composants cloud et communiquent avec le contrôleur qui assure l'élasticité du système. Le rôle des sondes (capteurs) est d'observer, de recueillir et de fournir les informations du système (ou un composant spécifique surveillé) à un moniteur. Le rôle des actionneurs est de déclencher des actions d'élasticité selon la logique du contrôleur d'élasticité. Enfin, les contrôles M et CU modélisent respectivement la mémoire et la CPU, qui représentent des ressources informatiques associées à chaque composant du système cloud. Par exemple, dans ce modèle bigraphique, nous pouvons considérer les valeurs de $M, M1$ et $M2$ respectivement par 8Gb, 3Gb et 2Gb, et les valeurs de $CU, CU1$ et $CU2$ respectivement par 2,8 Ghz, 0,8 Ghz et 0,7 Ghz.

6.5.2 Planification de l'élasticité du système élastique cloud

À travers cette étude de cas, nous expliquons comment nous pouvons utiliser les règles de réaction bigraphique définies pour exprimer et exécuter différents plans d'élasticité selon plusieurs scénarios d'évolution du système. Ces scénarios représentent des situations comportementales diverses qui peuvent se produire dans un système élastique basé cloud. Dans cet exemple, nous supposons que le développeur du système *TicketBook* souhaite déployer un nouveau service au sein de ses instances virtuelles allouées. Le service *SubwayTicketBook* (S5) pour les billets de métro. Ce-

pendant, les instances virtuelles actuelles ne sont pas suffisantes pour accueillir ce service web et pour gérer sa charge de travail anticipée, tout en gardant une bonne qualité de service. Dans ce cas, le système de cloud doit recourir à une méthode d'élasticité pour faire évoluer le système (scale up). Ensuite, nous supposons pour une raison quelconque que le développeur veut supprimer un certain service (mettre à jour le service), à titre d'exemple, le service *BoatTicketBook* (S2). Ici, le système doit réduire les capacités de calcul utilisées pour diminuer le coût d'exploitation et capitaliser les ressources disponibles. Dans la suite, nous exprimons la situation d'évolution présentée tout en tenant compte de différents plans d'action possibles basés sur les différentes méthodes d'élasticité.

Scénario 1 (élasticité horizontale). Pour la méthode de la mise à l'échelle horizontale, on définit une séquence de règles de réaction représentant le plan d'action d'élasticité. Premièrement, le système évolue par la réplication d'une instance de machine virtuelle, puis quand il n'est plus nécessaire, le système détruit cette nouvelle instance. À partir d'un bigraphe CS_0 qui représente l'état initial du système cloud décrit dans notre exemple, le plan d'action de l'élasticité représenté par une séquence de règles de réaction pour la mise à l'échelle horizontale est :

$$AP_1 = CS_0 \overset{R6}{\rightarrow} CS_1 \overset{R25}{\rightarrow} CS_2 \overset{R8}{\rightarrow} CS_3 \overset{R22}{\rightarrow} CS_4 \overset{R7}{\rightarrow} CS_5 \overset{R29}{\rightarrow} CS_6 \overset{R9}{\rightarrow} CS_7$$

La première règle à appliquer est $R6$ pour le déploiement d'un service dans une machine virtuelle donnée, à titre d'exemple, dans notre exemple le déploiement du service $S5$ (*SubwayTicketBook*) dans la machine virtuelle $VM2$ (CS_1). Cela augmenterait considérablement les flux entrants dans $VM2$. Par conséquent, la règle $R25$ est appliquée pour marquer la machine virtuelle chargée (CS_2). Pour ce cas, le plan d'action gère la nouvelle charge de travail en évoluant le système cloud horizontalement par l'application de la règle $R8$ qui réplique $VM2$ (CS_3). Ensuite, le service ($S2$) est redéployé dans la nouvelle instance de la machine virtuelle en utilisant $R22$ (CS_4). Après, pour une raison quelconque, le service $S2$ (*BoatTicketBook*) est supprimé par son développeur, cela est réalisé avec la règle $R7$ (CS_5). Dans ce cas, la machine virtuelle est marquée sous-utilisée en appliquant $R29$ (CS_6) et le système supprime l'instance VM supplémentaire en utilisant $R9$ (CS_7).

Scénario 2 (élasticité verticale). La méthode de la mise à l'échelle verticale n'est pas aussi communément adoptée comme l'élasticité horizontale. Elle représente une solution d'élasticité moins efficace. Comme dans le premier scénario, nous définissons une deuxième séquence de règles de réaction pour le plan d'action d'élasticité verticale. Dans ce scenario, le système évolue par l'augmentation des ressources allouées à l'instance de la machine virtuelle (mémoire et CPU) et après la diminution de ces ressources. Le plan d'action pour ce scénario d'évolution représentant l'élasticité verticale est donné par :

$$AP_2 = CS_0 \overset{R6}{\rightarrow} CS_1 \overset{R25}{\rightarrow} CS_2 \overset{R16}{\rightarrow} CS_3 \overset{R7}{\rightarrow} CS_4 \overset{R29}{\rightarrow} CS_5 \overset{R16}{\rightarrow} CS_6$$

D'une manière similaire au scénario d'élasticité horizontale, les deux premières règles à appliquer sont $R6$ (déploiement de service) et $R25$ (Marquage de machine

virtuelle chargée) (CS_1 et CS_2). Cependant, cette fois, plutôt que de répliquer/ consolider la machine virtuelle $VM2$, des ressources additionnelles sont allouées à la machine virtuelle en utilisant la règle $R16$ (CS_3). En outre, lorsque la charge de travail diminuée après la suppression du service $S2$ avec $R7$ (CS_4), la machine virtuelle est marquée sous-utilisée par $R29$ (CS_5) et le système retire les ressources ajoutées à $VM2$ en utilisant la règle $R16$ (CS_6).

Scénario 3 (migration). La méthode de migration peut être utilisée par certaines solutions cloud non élastiques ou des systèmes cloud qui utilisent seulement l'élasticité horizontale pour simuler le comportement obtenu par la méthode d'élasticité verticale. Par exemple, supposons dans notre étude de cas que le serveur $SE1$ hébergeant les deux machines virtuelles $VM1$ et $VM2$ se charge après le déploiement du nouveau service web $S5$ (*SubwayTicketBook*). Cette fois, au lieu d'appliquer les méthodes d'élasticité verticale ou horizontale, une instance de machine virtuelle migre vers un autre serveur (physique ou logique), mieux adapté pour la nouvelle charge entrante. Puis, après la suppression du service web $S2$ (*BoatTicketBook*), la machine virtuelle migre vers son serveur d'origine pour réduire le nombre d'instances allouées au système. Le plan d'action qui modélise ce scénario est donné par la séquence de règles de réaction suivante :

$$AP_3 = CS_0 \overset{R6}{\rightarrow} CS_1 \overset{R25}{\rightarrow} CS_2 \overset{R23}{\rightarrow} CS_3 \overset{R20}{\rightarrow} CS_4 \overset{R7}{\rightarrow} CS_5 \overset{R29}{\rightarrow} CS_6 \overset{R27}{\rightarrow} CS_7 \overset{R20}{\rightarrow} CS_8$$

Les règles de réaction utilisées pour assurer l'élasticité dans le plan d'action de la migration sont légèrement différentes. Dans ce scénario, après l'application de $R6$ (CS_1) et $R25$ (CS_2), le serveur est marqué sous-utilisé par $R23$ (CS_3). Ensuite, $R20$ est utilisé pour faire migrer la machine virtuelle $VM2$ vers un autre serveur moins chargé (CS_4). Après la suppression du service $S2$ avec $R7$ (CS_5), $VM2$ peut migrer éventuellement à son serveur d'origine en utilisant $R20$ de nouveau (CS_8).

Scénario 4 (hybride). Considérons un autre scénario où la migration et les méthodes d'élasticité horizontale et verticale peuvent être utilisées simultanément. Par exemple, supposant que le système cloud est incapable d'allouer des ressources supplémentaires à l'instance de la machine virtuelle parce que le serveur hôte est déjà chargé et il ne peut pas encore libérer des ressources inutilisées. Dans cette situation, l'instance de la machine virtuelle peut migrer vers un serveur moins chargé afin qu'elle puisse acquérir les ressources nécessaires pour s'exécuter proprement. La machine virtuelle peut migrer vers son serveur hôte d'origine quand la charge diminue. Le plan d'action du scénario est exprimé alors par la séquence de règles de réaction suivante :

$$AP_4 = CS_0 \overset{R6}{\rightarrow} CS_1 \overset{R25}{\rightarrow} CS_2 \overset{R20}{\rightarrow} CS_3 \overset{R16}{\rightarrow} CS_4 \overset{R7}{\rightarrow} CS_5 \overset{R29}{\rightarrow} CS_6 \overset{R20}{\rightarrow} CS_7$$

Les deux premières règles de réaction utilisées dans ce plan sont semblables aux précédents scénarios. Cette fois, la machine virtuelle $VM2$ migre vers un autre serveur en utilisant $R20$ (CS_3), ensuite $R16$ peut être appliquée pour allouer des ressources supplémentaires à la machine virtuelle (CS_5). Ensuite, quand la charge de travail diminue, elle migre vers son serveur d'origine $SE1$ en utilisant $R20$ de nouveau (CS_7).

Dans cette étude de cas, le développeur possède plusieurs choix pour gérer l'élasticité dans le système de réservation de billets (*TicketBook*). Par exemple, s'il choisit d'appliquer la méthode d'élasticité verticale, la stratégie d'élasticité qui exécute le plan d'action pour la mise à l'échelle verticale est donnée par :

$$STRAT : If\ CS \vDash \varphi\ Then\ AP_2, \text{ où } B_\varphi \stackrel{\text{def}}{=} (LB^a_r.(d))$$
$$VM2_{v1s2c4}.(d4|S5^{p5,u5}_{l5c9}|M2_{c4}|CU2_{c4})),\ M2.value = 2Gb \text{ et } CU2.value = 0.7Ghz$$

Les scénarios présentés sont quelques exemples qui montrent comment les règles de réaction bigraphiques peuvent être utilisées pour exprimer divers et complexes scénarios d'évolution et plans d'élasticité. Des architectures cloud plus complexes et autres scénarios de comportement peuvent être facilement exprimés à travers le modèle bigraphique proposé. Les règles de réaction définies peuvent être composées facilement pour modéliser formellement différents scénarios. Par exemple, quand un système élastique basé cloud est incapable de retourner à sa configuration initiale après un processus d'adaptation (plasticité) [Gambi 2013a] ou lorsque la mise à l'échelle d'un composant cloud peut déclencher un effet en cascade en imposant le redimensionnement d'autres composants.

6.6 Conclusion

Un système élastique basé cloud et son comportement sont formalisés dans notre approche par un système réactif bigraphique. . La reconfiguration du système cloud est modélisée par des plans d'action d'élasticité qui prennent la forme de séquences d'applications d'une ou de plusieurs règles de réaction. Par ailleurs, les règles de réaction présentées dans ce chapitre peuvent être instanciées pour exprimer le comportement dynamique de n'importe quel systèmes élastiques basé cloud. En particulier, le comportement en termes d'interactions front-end/back-end et méthodes d'élasticité aux différents niveaux (service, plate-forme, infrastructure et répartition de charge), tout en préservant leurs contraintes architecturales. En outre, les règles de réaction définies peuvent être combinées avec des prédicats BiLog afin d'exprimer diverses stratégies d'élasticité.

Chapitre 7

Implémentation et vérification des systèmes élastiques basés cloud

Sommaire

7.1 Introduction 98
7.2 Le Framework MoVeElastic 99
7.3 Implémentation de MoVeElastic 101
7.3.1 Aspects structurels 101
7.3.2 Aspects comportementaux 104
7.3.3 Validation par simulation 107
7.4 Stratégies d'élasticité dans MoVeElastic 109
7.5 Vérification formelle de l'élasticité 111
7.5.1 Vérification par invariants 112
7.5.2 Vérification via le model-checker LTL 116
7.6 Conclusion 118

7.1 Introduction

Les systèmes réactifs bigraphiques représentent un excellent moyen pour modéliser les aspects structurels et comportementaux des systèmes élastiques basés cloud. Néanmoins, les outils développés autour des BRS tels que BPLTool [Højsgaard 2011], BigMC [Perrone 2012] et BigraphER [Sevegnani 2012] ne répondent pas à nos exigences en termes d'expressivité et de performance. À titre d'exemple, l'outil BigMC qui est le seul model-checker conçu pour les systèmes réactifs bigraphiques est très limité en ce qui concerne l'expression des propriété complexes. De plus, il ne supporte pas la logique de typage des BRS. Par ailleurs, l'outil BigraphER fournit un cadre pour manipuler, visualiser et simuler des systèmes réactifs bigraphiques. Cependant, cet outil n'est pas équipé d'un modèle-checker et donc il ne permet pas de réaliser des vérifications formelles sur les modèles bigraphiques obtenus.

Dans ce travail nous optons pour le langage Maude comme l'alternative la plus adéquate à ces outils. Maude [Clavel 2007] est un langage de spécification formelle de haut niveau, fondé sur la logique de réécriture et la logique équationnelle. Ce langage représente un candidat idéal qui permet de prendre en compte de façons naturel les aspects structurels et comportementaux des modèles bigraphiques définis au préalable pour modéliser les systèmes élastiques basés cloud. De plus, l'environnement

Maude permet de valider les spécifications obtenues et analyser le comportement élastique des systèmes cloud à travers un nombre de techniques développées autour de ce langage.

Dans ce chapitre, nous introduisons notre Framework formel MoVeElastic défini en intégrant les systèmes réactifs bigraphiques dans le langage Maude pour proposer un cadre formel permettant la spécification et vérification des systèmes élastiques basés cloud. Dans un premier temps, nous détaillons le principe du Framework MoVeElastic. Ensuite, nous nous intéressons a l'implémentation des aspects structurels et comportementaux des systèmes élastiques basés cloud, ainsi que leur validation par la simulation. Après, nous montrons comment nous avons procédé pour exprimer les stratégies d'élasticité dans MoVeElastic en utilisant un langage de stratégies associé à Maude. Enfin, nous validons les spécifications Maude obtenues par la vérification formelle de l'élasticité. Ceci en proposant deux techniques différentes de vérification formelle, la première est basée sur la technique de vérification par invariants, et la deuxième se base sur le model-checker LTL Maude.

7.2 Le Framework MoVeElastic

La modélisation et la vérification formelles des systèmes élastiques basés cloud sont des critères centraux pour une bonne solution de gestion et de planification de l'élasticité dans les environnements cloud computing. À travers la modélisation de ces systèmes, il sera possible de spécifier leurs architectures et de définir de diverses propriétés de nature désirable ou indésirable liées à leur comportement élastique ou à leur structure interne. Ces propriétés peuvent être vérifiées formellement, ce qui permet d'assurer l'élasticité d'un système cloud et d'interdire des évolutions non voulues. Dans cette section nous donnons un aperçu sur notre Framework MoVeElastic (pour Modeling and Verifying Elastic Cloud Systems) fournissant un cadre formel pour la modélisation des aspects structurels et comportementaux des systèmes basés cloud et la vérification de leur élasticité.

Le Framework MoVeElastic est implémenté en intégrant notre modèle bigraphique dans le langage Maude, cela permet de produire des spécifications Maude équivalentes aux modèles bigraphiques des système élastiques cloud. MoVeElastic consiste en un ensemble de modules Maude de natures différentes couvrant tous les aspects architecturaux et comportementaux. Dans un premier temps, nous proposons un ensemble de modules fonctionnels qui décrivent les opérations de base pour la spécification des différents aspects structurels. Chaque module fonctionnel représente une région spécifique d'un bigraphe modélisant un système élastique basés cloud, qui est lui-même représenté comme un module fonctionnel. Le comportement des systèmes cloud en termes d'interactions front-end/back-end et méthodes d'élasticité est représenté dans un module système contenant trois catégories de règles de réécriture équivalentes aux catégories de règles de réaction présentées dans le chapitre précèdent. Par ailleurs, un autre module de stratégies est introduit pour décrire les différentes stratégies d'élasticité. Enfin, deux modules systèmes supplémentaires sont définis pour effectuer la vérification formelle de l'élasticité. Un premier module

contenant l'état initial d'un système cloud, tandis que le deuxième module système contient les prédicats d'états pertinents utilisés pour exprimer les propriétés d'élasticité.

Notre approche formelle pour la modélisation et la vérification des systèmes élastiques basés cloud ayant abouti au prototype d'un Framework que nous avons nommé MoVeElastic, est définie par les étapes illustrées ci-dessous (voir Figure 7.1) :

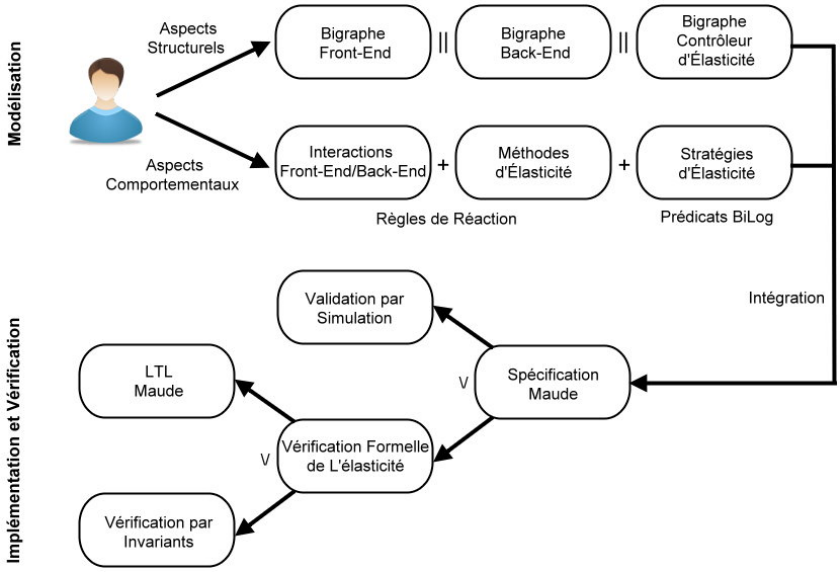


FIGURE 7.1 – Étapes de MoVeElastic

1. **Modélisation d'un système élastique basé cloud** : cette première étape consiste à la spécification des différents aspects conceptuels et architecturaux du système cloud à concevoir. À ce stade, les méthodes et les stratégies utilisées pour assurer l'élasticité dans le système doivent également être énoncées. Cette spécification se fait à travers notre modèle bigraphique pour les systèmes élastiques basés cloud.
2. **Génération d'une description formelle Maude** : la deuxième étape consiste à l'intégration des modèles bigraphiques d'un système cloud obtenus lors de la première étape dans le langage Maude pour avoir des spécifications exécutables et vérifiables.
3. **Validation par simulation** : la troisième étape consiste à la validation des spécifications Maude obtenues via la simulation de leur exécution. Pour ce faire, le système Maude prend en entrée un état initial personnalisé représentant l'architecture du système cloud modélisé et applique des règles de réécriture sur cette configuration. Ceci permet de vérifier le bon fonctionnement non seulement du système élastique cloud, mais aussi des règles de réécriture et des stratégies d'élasticité définies.
4. **Vérification formelle de l'élasticité** : la dernière étape consiste à la vé-

rification des propriétés d'élasticité à l'aide des vérificateurs intégrés dans l'environnement Maude. Dans ce cas, le choix se fait entre deux techniques de vérification possibles qui peuvent être adoptées. La première technique est basée sur la commande search de Maude et la technique de vérification par invariants, alors que la deuxième se base sur le model-checker LTL Maude. Enfin, il faut tirer les conclusions appropriées des résultats de vérification obtenus.

7.3 Implémentation de MoVeElastic

Dans cette section, nous présentons les détails de l'implémentation du Framework MoVeElastic à l'aide du langage de spécification formelle Maude. Nous divisons cette implémentation en deux classes de modules Maude. La première concerne les modules fonctionnels représentant les aspects statiques d'un système élastique cloud. L'autre classe regroupe les modules systèmes qui implémentent les aspects comportementaux de tels systèmes. Une vue sur l'ensemble des modules Maude défini pour la mise en œuvre du Framework MoVeElastic et leurs dépendances est donnée sur la Figure 7.2.

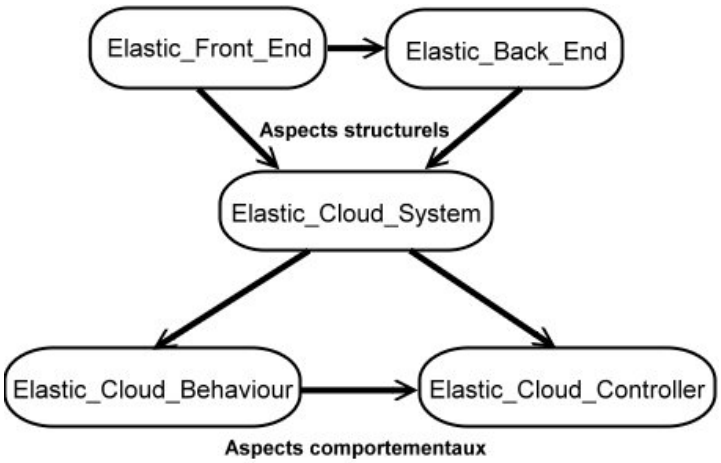


FIGURE 7.2 – Hiérarchie des modules Maude implémentant MoVeElastic

7.3.1 Aspects structurels

Dans MoVeElastic, nous suivons une approche d'implémentation modulaire afin de donner une sémantique opérationnelle aux systèmes élastiques basés cloud. En particulier, nous associons à chaque entité d'un système cloud un module fonctionnel définissant la sémantique du bigraphe qui modélise cette entité. En termes de syntaxe, cette implémentation basé Maude a été conçue pour être aussi proche que possible du langage de termes algébrique des bigraphes [Milner 2008]. Trois

modules fonctionnels sont proposés : `Elastic_Front_End`, `Elastic_Back_End` et `Elastic_Cloud_System`.

Le module fonctionnel `Elastic_Front_End` du Listing 7.1 regroupe les sortes principales, les relations de sous-sorte et les opérations algébriques utilisées pour décrire la structure syntaxique des différents éléments bigraphiques associés au bigraphe front-end. Notamment, les sortes `EndUser` et `Developer` sont introduites pour spécifier les utilisateurs finaux et les développeurs respectivement. La sorte `ClientState` est utilisée pour décrire l'état actuel d'un client (utilisateur final ou développer) qui peut être inactif ou demandeur. `AllocService` et `DepService` sont des sortes qui expriment les services actuellement alloués par un utilisateur final (`AllocService`) ou déployés par un développeur (`DepService`). Enfin, une relation de sous-sorte est mise en place entre les sortes mentionnées.

Listing 7.1– Module fonctionnel Elastic_Front_End

```
fmod Elastic_Cloud_FrontEnd is
protecting NAT .
sorts EndUser Developer ClientState Request  .
sorts EndUserList DeveloperList RequestList .
sorts AllocService DepService .
sorts AllocServiceList DepServiceList .
subsort EndUser < EndUserList .
subsort Developer < DeveloperList .
subsort Request < RequestList .
subsort AllocService < AllocServiceList .
subsort DepService < DepServiceList .
ops req idle : -> ClientState [ctor] .
op EU_:_(_) : Nat ClientState AllocServiceList -> EndUser [ctor] .
op EEL : -> EndUserList [ctor] .
op _,_ : EndUserList EndUserList -> EndUserList [ctor assoc comm id:
    EEL] .
op D_:_(_) : Nat ClientState DepServiceList -> Developer [ctor] .
op EDL : -> DeveloperList [ctor] .
op _,_ : DeveloperList DeveloperList -> DeveloperList [ctor assoc
    comm id: EDL] .
op RQ_ : Nat -> Request [ctor] .
op ERQL : -> RequestList [ctor] .
op _,_ : RequestList RequestList -> RequestList [ctor assoc comm id:
    ERQL] .
endfm
```

De la même manière, nous décrivons les aspects structurels d'un bigraphe back-end cloud à l'aide du module fonctionnel `Elastic_Back_End` (voir Listing 7.2). Ce module importe le module `Elastic_Front_End` et décrit la syntaxe des constructions des éléments de base d'un back-end. Les différents composants architecturaux dans un back-end cloud, y compris les équilibreurs de charge, les serveurs, les machines virtuelles, les conteneurs et les services (applications) sont implémentés respectivement par les sortes `LoadBalancer`, `Server`, `VirtualMachine`, `Container` et `Service`. Leurs listes d'éléments par les sortes `LoadBalancerList`, `ServerList`, `VirtualMachineList`, `ContainerList` et `ServiceList`. Une relation de sous-sorte

est établie entre chaque composants et sa liste d'éléments. La sorte `Resources` représente des ressources de calcul qui peuvent être de sorte `Memory`, `Cpu`, `Storage` et `Bandwidth` expriment respectivement la mémoire, CPU, capacité de stockage et bande passante. `ComponentState` est la sorte utilisée pour indiquer l'état d'un composant cloud (chargé, stable ou sous utilisé). La sorte `ComponentNature` exprime la nature de chaque composant qui peut être représenté par les opérations `original`, `copy`, `destroyed`, `migrated`, `increased` et `decreased`.

Listing 7.2– Module fonctionnel Elastic_Back_End

```
fmod Elastic_Cloud_BackEnd is
protecting NAT .
protecting FLOAT .
including Elastic_Cloud_FrontEnd .
sorts LoadBalancer Server VirtualMachine Container Service .
sorts LoadBalancerList ServerList VirtualMachineList ContainerList
    ServiceList .
sorts ComponentState ComponentNature . sorts Memory Cpu Storage
    Bandwidth Resources .
subsort Memory Cpu Storage Bandwidth < Resources .
subsort LoadBalancer < LoadBalancerList .
subsort Server < ServerList .
subsort VirtualMachine < VirtualMachineList .
subsort Container < ContainerList .
subsort Service < ServiceList .
ops loaded stable underused : -> ComponentState [ctor] .
ops none original copy destroyed migrated increased decreased : ->
    ComponentNature [ctor] .
op M=_Gb : Float -> Memory [ctor] .
op C=_Ghz : Float -> Cpu [ctor] .
op S=_Gb : Float -> Storage [ctor] .
op B=_Mb : Float -> Bandwidth [ctor] .
op ERL : -> Resources [ctor] .
op _,_ : Resources Resources -> Resources [ctor assoc comm id: ERL] .
op LB_:_<_>(_){_} : Nat ComponentState Resources RequestList
    ComponentNature -> LoadBalancer [ctor] .
op EBL : -> LoadBalancerList [ctor] .
op _,_ : LoadBalancerList LoadBalancerList -> LoadBalancerList [ctor
    assoc comm id: EBL] .
op SE_:_<_> : Nat ComponentState Resources -> Server [ctor] .
op ESEL : -> ServerList [ctor] .
op _,_ : ServerList ServerList -> ServerList [ctor assoc comm id:
    ESEL] .
op VM_-_:_<_>{_} : Nat Nat ComponentState Resources ComponentNature
    -> VirtualMachine [ctor] .
op EVL : -> VirtualMachineList [ctor] .
op _,_ : VirtualMachineList VirtualMachineList -> VirtualMachineList
    [ctor assoc comm id: EVL] .
op CT_-_:_<_>{_} : Nat Nat ComponentState Resources ComponentNature
    -> Container [ctor] .
op ECL : -> ContainerList [ctor] .
op _,_ : ContainerList ContainerList -> ContainerList [ctor assoc
    comm id: ECL] .
```

```
op S_-_:_<_>(_){_} : Nat Nat ComponentState Resources RequestList
    ComponentNature -> Service [ctor] .
op ESL : -> ServiceList [ctor] .
op _,_ : ServiceList ServiceList -> ServiceList [ctor assoc comm id:
    ESL] .
op AS_ : Nat -> AllocService [ctor] .
op EASL : -> AllocServiceList [ctor] .
op _,_ : AllocServiceList AllocServiceList -> AllocServiceList [ctor
    assoc comm id: EASL] .
op DS_ : Nat -> DepService [ctor] .
op EDSL : -> DepServiceList [ctor] .
op _,_ : DepServiceList DepServiceList -> DepServiceList [ctor assoc
    comm id: EDSL] .
endfm
```

De façon incrémentale, nous construisons le module fonctionnel `Elastic_Cloud_System` (voir Listing 7.3) qui regroupe les deux modules précédents aux quels nous ajoutons les opérations algébriques nécessaires pour la construction d'un bigraphe cloud complet comprenant les deux parties front-end et back-end.

Les deux modules mentionnés précédemment (Listings 7.1 et 7.2) sont importés par le module fonctionnel `Elastic_Cloud_System` (voir Listing 7.3) afin de regrouper les aspects nécessaires pour définir les opérations algébriques de construction d'un bigraphe cloud.

Listing 7.3– Module fonctionnel Elastic_Cloud_System

```
fmod Elastic_Cloud_System is
including Elastic_Cloud_FrontEnd .
including Elastic_Cloud_BackEnd .
sorts CloudBigraph FrontEnd BackEnd .
op _||_[_] : FrontEnd BackEnd Nat -> CloudBigraph [ctor] .
op _|_|_ : DeveloperList EndUserList RequestList -> FrontEnd [ctor] .
op _|_|_|_ : LoadBalancerList ServerList VirtualMachineList
    ServiceList -> BackEnd [ctor] .
op _|_|_|_ : LoadBalancerList ServerList ContainerList ServiceList ->
     BackEnd [ctor] .
endfm
```

7.3.2 Aspects comportementaux

Dans le Framework MoVeElastic, toutes les règles de réaction définies autour de notre modèle bigraphique proposé pour modéliser les systèmes élastiques cloud sont transformées directement en règles de réécriture respectant la syntaxe du langage Maude. Nous introduisons à ce niveau le module système `Elastic_Cloud_Behaviour` qui regroupe les règles de réécriture correspondant aux trois catégories déjà identifiées auparavant (Chapitre 6). Ce module importe tous les autres modules déjà définis jusqu'à présent. La première catégorie de règles de réécriture conditionnelles définit les différentes interactions entre les clients du système cloud et les applications (interactions front-end/back-end), tels que le déploiement

d'une nouvelle application ou service web par un développeur, demande de service ou l'allocation d'un service par un utilisateur final. La deuxième catégorie exprime des actions d'élasticité qui peuvent être appliquées aux différents niveaux cloud et selon les trois méthodes d'élasticité, la mise à l'échelle horizontale, la mise à l'échelle verticale et la migration. Parmi ces actions d'élasticité, on peut mentionner la réplication d'une machine virtuelle, diminution des ressources allouées à un équilibreur de charge et la migration d'un conteneur. La dernière catégorie est mis en œuvre dans MoVeElastic par un ensemble de règles de réécriture qui permettent de changer, quand il est nécessaire, les états des différents composants cloud.

Nous nous contentons ici de présenter et détailler le fonctionnement de certaines règles de réécriture montrées dans Listing 7.4. Le module `Elastic_Cloud_Behaviour` complet est présenté dans l'Annexe A. La première règle de réécriture conditionnelle dans le module ayant l'étiquette `"send-request"` modélise la soumission d'une demande de service (requête) par un client de type utilisateur final. Cette requête est transmise à un équilibreur de charge avant qu'elle soit transférée au service ou l'application demandé. La deuxième règle `"deploy-service-ct"` définit le déploiement d'un service ou une application par un client de type développeur au sein d'un conteneur. En exécutant cette règle, un nouveau service est créé au sein du conteneur cible. La troisième règle étiquetée `"vm-instance-replication"` décrit la réplication d'une instance de machine virtuelle. Cette règle est appliquée lorsqu'une machine virtuelle est chargée ou défaillante en créant un clone de l'instance de la machine virtuelle. Cette nouvelle copie peut être utilisée ultérieurement pour réduire la charge sur la machine virtuelle originale en redéployant les applications ou les services afin d'équilibrer la charge entre les deux machines virtuelles. La règle avec l'étiquette `"ct-instance-consolidation"` est exécutée lorsqu'une instance d'un conteneur est sous-utilisée ou n'est plus nécessaires à cause d'une diminution de la charge de travail. En outre, la règle `"increase-memory-ct"` permet d'augmenter la mémoire vive actuellement allouée à une instance de conteneur chargée (élasticité verticale). Une autre règle qui représente une action d'élasticité verticale est donnée par l'étiquette `"decrease-memory-lb"`. Cette règle permet de réduire la mémoire vive effectivement allouée à un équilibreur de charge sous utilisé. La règle `"service-instance-migration"` décrit le redéploiement d'une instance de service ou d'application d'une machine virtuelle hôte chargée vers une autre moins chargée. Les deux dernières réglés étiquetées `"marking-vm-underused"` et `"marking-lb-loaded"` sont des règles de marquage qui permettent de changer l'état, respectivement d'une machine virtuelle à l'état sous utilisée et d'un équilibreur à l'état chargé.

Listing 7.4– Module système Elastic_Cloud_Behaviour

```
mod Elastic_Cloud_Behaviour is
including Elastic_Cloud_FrontEnd .
including Elastic_Cloud_BackEnd .
including Elastic_Cloud_System .
```

```
vars nn n1 n2 n3 n4 : Nat . vars f1 f2 f3 : Float .
var b : BackEnd . var f : FrontEnd . var eul : EndUserList .
var dvl : DeveloperList . vars rl1 rl2 : RequestList .
vars r1 r2 r3 : Resources .vars l : LoadBalancerList .
vars se : ServerList . vars v1 v2 : VirtualMachineList .
vars c1 c2 : ContainerList . vars s1 s2 : ServiceList .
var as : AllocServiceList . var ds : DepServiceList .
var cs : ClientState . var cc : ComponentState .
var cn1 cn2 cn3 : ComponentNature .
...
crl [send-request] :
dvl|eul|rl1, RQ n1||l , LB n2:stable <r1> (rl2){cn1}|se|v1|s1 [ nn ]
=>
dvl|eul|rl1||l, LB n2 : stable <r1> (rl2, RQ n1){cn1}|se|v1|s1 [ sd(
    nn, 1) ]
if nn > 0 .
...
crl [deploy-service-ct] :
dvl, D n1 :req(ds, DS n2)|eul|rl1||l|se|c1, CT n3-n4 :stable <r1> {
    cn1}|s1 [ nn ]
=>
dvl, D n1 :req(ds, DS n2)|eul|rl1||l|se|c1, CT n3-n4 :stable <r1> {
    cn1}|s1 ,
S n2 - n3 :stable <M= 0.0 Gb, C= 0.0 Ghz, S= 0.0 Gb, B= 0.0 Mb >(ERQL
    ){original} [ sd(nn, 1) ]
if nn > 0 .
...
crl [vm-instance-replication] :
f||l|se, SE n1 :stable <r1>|v1 , VM n2-n1 :loaded <r2> {original}|s1
    [ nn ]
=>
f||l|se , SE n1 :stable <r1>|v1, VM n2-n1 :loaded <r2> {original} ,
VM (n2 * 10 + 1)-n1 :stable <r2> {copy}|s1 [ sd(nn, 1) ]
...
crl [ct-instance-consolidation] :
f||l|se, SE n1 : stable <r1>|c1, CT n2-n1 : underused <r2> {destroyed
    }|s1 [ nn ]
=>
f||l |se, SE n1 : stable <r1>|c1|s1 [ sd(nn, 1) ]
if nn > 0 .
...
crl [increase-memory-ct] :
f||l|se|c1, CT n1 - n2 :loaded <r1, M= f1 Gb>{cn1}|s1 [ nn ]
=>
f||l|se|c1, CT n1 - n2 :loaded <r1, M= f1 + 0.5 Gb> {increased}|s1 [
    sd(nn, 1) ]
if f1 < 8.0 /\ nn > 0 .
...
crl [decrease-memory-lb] :
f||l, LB n1 :underused <r1, M= f1 Gb> (rl1){cn1}|se|v1|s1 [ nn ]
=>
f||l, LB n1 :underused <r1, M= f1 - 0.5 Gb> (rl1) {decreased}|se|v1|
    s1 [ sd(nn, 1) ]
if f1 > 1.0 /\ nn > 0 .
```

```
...
crl [service-instance-migration-vm] :
f||l|se|v1, VM n1-n2 : cc <r1> {cn1} , VM n3-n2 : stable <r2>{cn2}|
s1, S n4 - n1 :loaded <r3> (rl1) {cn3} [ nn ]
=>
f||l|se|v1, VM n1-n2 : cc <r1> {cn1}, VM n3-n2 : stable <r2>{cn2}|
s1, S n4-n3 :loaded <r3> (rl1) {migrated} [ sd(nn, 1) ]
if nn > 0 .
...
crl [marking-vm-underused] :
f||l|se|v1, VM n1-n2 :stable <r1> {cn1}|s1 [ nn ]
=>
f||l|se|v1, VM n1-n2 :underused <r1> {cn1}|s1 [ sd(nn, 1) ]
if nn > 0 .
...
crl [marking-lb-loaded] :
f||l, LB n1 :stable <r1>(rl1){cn1}|se|v1|s1 [ nn ]
=>
f||l, LB n1 :loaded <r1>(rl1){cn1}|se|v1|s1 [ sd(nn, 1) ]
if nn > 0 .
...
Endm
```

7.3.3 Validation par simulation

Jusqu'à présent nous avons défini les modules fonctionnels et systèmes nécessaires pour représenter la structure architecturale des systèmes élastique basés cloud ainsi que leur comportement dynamique. Cette spécification formelle en langage Maude ainsi obtenue peut maintenant faire l'objet d'une exécution. En effet, en utilisant l'environnement Maude qui est très versatile en matière de simulations, nous pouvons assurer le bon fonctionnement d'une partie du système ou le système complet via l'exécution des règles de réécriture. L'environnement pend en entrée une configuration initiale personnalisée qui représente une architecture cloud initiale (avant son évolution ou réadaptation) et procède à son exécution en appliquant des règles de réécriture afin d'obtenir un état final (anticipé ou non). Cette simulation permet de vérifier le bon fonctionnement de chacune des règles de réécriture du module `Elastic_Cloud_Behaviour` appliquée de manière indépendante, ou bien l'ensemble des règles de réécriture mises en commun pour modéliser un comportement spécifique, désiré ou non.

Exemple. Afin de mieux présenter le principe de validation de la spécification Maude à travers la simulation d'exécution des règles proposées, nous reprenons l'exemple de l'étude de cas présenté dans le chapitre précèdent (section 5.5). Il s'agit d'un système de réservation de billets (*TicketBook*) [Rong 2014] déployé dans une infrastructure cloud (Amazon EC2). La configuration initiale du back-end de ce système élastique basé cloud a été présentée par le bigraphe B de la Figure 6.5. Ici, nous intégrons cette configuration initiale du système cloud dans la spécification Maude proposée en termes d'une expression qui respect la syntaxe définie dans les modules fonctionnels introduits précédemment. Nous avons conçu cette syntaxe pour

être aussi proche que possible du langage de termes algébrique des bigraphes. Cela permet de faciliter l'intégration des modèles bigraphiques des systèmes élastiques cloud dans le langage Maude. La configuration initiale du système cloud dans le langage Maude est montrée dans le Listing 7.5.

Listing 7.5– Configuration initiale du système élastique cloud

```
EDL| EEL|RQ2, RQ11, RQ21, RQ4, RQ5, RQ17, RQ20, RQ1, RQ2, RQ6| LB 1 :
    stable< M= 2.0 Gb, C= 2.1 Ghz, S= 70.0 Gb, B= 6.0 Mb >(ERQL){
    original}|SE1 :stable< M= 16.0 Gb, C= 5.8 Ghz, S= 3000.0 Gb, B=
    60.0 Mb >, SE2 :stable< M= 12.0 Gb, C= 4.4 Ghz, S= 2000.0 Gb, B=
    45.0 Mb >|VM1-1 :stable < M= 3.0 Gb, C= 2.3 Ghz, S= 1000.0 Gb, B=
    20.0 Mb >{original}, VM2-1 :stable < M= 4.0 Gb, C= 2.1 Ghz, S=
    800.0 Gb, B= 20.0 Mb >{original}|S1-1 :stable < M= 0.7 Gb, C= 0.6
    Ghz, S= 70.0 Gb, B= 2.0 Mb >(ERQL){original}, S2-1 :stable < M=
    0.8 Gb, C= 0.8 Ghz, S= 100.0 Gb, B= 1.5 Mb >(ERQL){original}, S3-2
     :stable < M= 1.2 Gb, C= 1.1 Ghz, S= 110.0 Gb, B= 3.0 Mb >(ERQL){
    original}, S4-2 :stable < M= 0.5 Gb, C= 0.5 Ghz, S= 60.0 Gb, B=
    1.0 Mb >(ERQL){original} [n] .
```

```
                     \||||||||||||||||||/
                   --- Welcome to Maude ---
                     /||||||||||||||||||\
           Maude 2.6 built: Mar 31 2011 23:36:02
           Copyright 1997-2010 SRI International
                 Fri Oct 28 13:08:57 2016

Maude>
Maude> rewrite [1000] in Elastic_Cloud_Behaviour : EDL | EEL | RQ 2,RQ 11,RQ 21,RQ 4,
    RQ 5,RQ 17,RQ 20,RQ 1,RQ 2,RQ 6 || LB 1 : stable < M= 2.0 Gb,C=
    2.1000000000000001 Ghz,S= 7.0e+1 Gb,B= 6.0 Mb >(ERQL){original} | SE 1 :
    stable < M= 1.6e+1 Gb,C= 5.7999999999999998 Ghz,S= 3.0e+3 Gb,B= 6.0e+1 Mb
    >,SE 2 : stable < M= 1.2e+1 Gb,C= 4.4000000000000004 Ghz,S= 2.0e+3 Gb,B=
    4.5e+1 Mb > | VM 1 - 1 : stable < M= 3.0 Gb,C= 2.2999999999999998 Ghz,S=
    1.0e+3 Gb,B= 2.0e+1 Mb >{original},VM 2 - 1 : stable < M= 4.0 Gb,C=
    2.1000000000000001 Ghz,S= 8.0e+2 Gb,B= 2.0e+1 Mb >{original} | ((S 3 - 2 :
    stable < M= 1.2 Gb,C= 1.1000000000000001 Ghz,S= 1.1e+2 Gb,B= 3.0 Mb >(
    ERQL){original},S 4 - 2 : stable < M= 5.0e-1 Gb,C= 5.0e-1 Ghz,S= 6.0e+1 Gb,
    B= 1.0 Mb >(ERQL){original}),S 2 - 1 : stable < M= 8.0000000000000004e-1
    Gb,C= 8.0000000000000004e-1 Ghz,S= 1.0e+2 Gb,B= 1.5 Mb >(ERQL){original}),S
    1 - 1 : stable < M= 6.9999999999999996e-1 Gb,C= 5.9999999999999998e-1 Ghz,
    S= 7.0e+1 Gb,B= 2.0 Mb >(ERQL){original}[1000] .
rewrites: 3000 in 30002ms cpu (71ms real) (99 rewrites/second)
result CloudBigraph: EDL | EEL | ERQL || LB 1 : stable < M= 2.0 Gb,C=
    2.1000000000000001 Ghz,S= 7.0e+1 Gb,B= 6.0 Mb >(ERQL){original} | SE 1 :
    stable < M= 1.6e+1 Gb,C= 5.7999999999999998 Ghz,S= 3.0e+3 Gb,B= 6.0e+1 Mb
    >,SE 2 : stable < M= 1.2e+1 Gb,C= 4.4000000000000004 Ghz,S= 2.0e+3 Gb,B=
    4.5e+1 Mb > | VM 1 - 1 : stable < M= 3.0 Gb,C= 2.2999999999999998 Ghz,S=
    1.0e+3 Gb,B= 2.0e+1 Mb >{original},VM 2 - 1 : stable < M= 4.0 Gb,C=
    2.1000000000000001 Ghz,S= 8.0e+2 Gb,B= 2.0e+1 Mb >{original} | S 1 - 1 :
    stable < M= 6.9999999999999996e-1 Gb,C= 5.9999999999999998e-1 Ghz,S= 7.0e+1
    Gb,B= 2.0 Mb >(ERQL){original},S 2 - 1 : stable < M= 8.0000000000000004e-1
    Gb,C= 8.0000000000000004e-1 Ghz,S= 1.0e+2 Gb,B= 1.5 Mb >(ERQL){original},S
    3 - 2 : stable < M= 1.2 Gb,C= 1.1000000000000001 Ghz,S= 1.1e+2 Gb,B= 3.0 Mb
    >(ERQL){original},S 4 - 2 : stable < M= 5.0e-1 Gb,C= 5.0e-1 Ghz,S= 6.0e+1
    Gb,B= 1.0 Mb >(ERQL){original}[0]

Maude>
```

FIGURE 7.3 – Résultats de la simulation du système cloud

En utilisant cette configuration comme état initial de la commande rewrite, nous pouvons faire appel à l'interpréteur par défaut de Maude pour exécuter les règles de réécriture définies et simuler différents scénarios d'évolution du système élastique cloud. Ces règles de réécriture sont appliquées par le moteur d'exécution de Maude d'une façon équitable et selon une stratégie par défaut (de haut en bas) pour réécrire

l'état initial en entrée. La réécriture termine lorsque le nombre d'applications de règles atteint une limite spécifiée dans la commande.

Le résultat de la simulation dans ce cas d'étude est donné par le capture d'écran dans la Figure 7.3. On y voit apparaître un nouveau terme bigraphique constituent le résultat de la réécriture qui a été limitée à milles pas pour permettre de visualiser les états intermédiaires des composants cloud, ainsi que les requêtes transférées. D'où, l'expression `CloudBigraph` représente la nouvelle configuration du système cloud après l'application des règles de réécriture.

7.4 Stratégies d'élasticité dans MoVeElastic

Dans le Framework MoVeElastic nous implémentons les stratégies d'élasticité définies pour contrôler l'exécution des règles de réaction bigraphiques représentant des actions relatives aux trois méthodes d'élasticité par l'intermédiaire du langage de stratégies de Maude.

À travers ce langage de stratégies, nous définissons trois stratégies d'élasticité principales chacune implémente une méthode d'élasticité particulaire (élasticité horizontale, élasticité verticale et migration). Un nouveau type de module Maude est alors introduit, il s'agit d'un module de stratégies appelé : `Elastic_Cloud_Controller` présenté dans le Listing 7.6. Nous tenons à noter que ces différentes stratégies sont construites à partir des stratégies élémentaires, qui peuvent être des autres stratégies plus simples ou bien des applications de règles de réécriture introduit précédemment dans le module `Elastic_Cloud_Behaviour` pour spécifier des actions d'élasticité.

Listing 7.6– Module de strategies Elastic_Cloud_Controller

```
load maude-strat.maude
(smod Elastic_Cloud_Controller is
protecting Elastic_Cloud_Behaviour .
protecting Elastic_Cloud_FrontEnd .
protecting Elastic_Cloud_BackEnd .
protecting Elastic_Cloud_System .
strat enduser-request-plan : @ CloudBigraph .
sd enduser-request-plan :=
(marking-req-eu ; (make-service-request ; send-request ;
proxy-request ; destroy-request) ! ; marking-idle-eu) * .
strat horizontal-scale-up : @ CloudBigraph .
sd horizontal-scale-up :=
(marking-vm-loaded ; (vm-instance-replication)
! ; unmarking-vm-loaded) ! .
strat horizontal-scale-down : @ CloudBigraph .
sd horizontal-scale-down :=
(marking-vm-underused ; (vm-instance-to-be-destroyed ; vm-instance-
    consolidation) ! ; unmarking-vm-underused) ! .
strat horizontal-scale-strategy : @ CloudBigraph .
sd horizontal-scale-strategy :=
(enduser-request-plan | (horizontal-scale-up ; horizontal-scale-down)
     !) * .
```

```
strat vertical-scale-up : @ CloudBigraph .
sd vertical-scale-up :=
(marking-vm-loaded ; (increase-memory-vm | increase-cpu-vm) ! ;
    unmarking-vm-loaded) ! .
strat vertical-scale-down : @ CloudBigraph .
sd vertical-scale-down :=
(marking-vm-underused ; (decrease-memory-vm | decrease-cpu-vm) ! ;
    unmarking-vm-underused) ! .
strat vertical-scale-strategy : @ CloudBigraph .
sd vertical-scale-strategy :=
(enduser-request-plan | (vertical-scale-up ; vertical-scale-down) !)
    * .
strat migration-scale-up : @ CloudBigraph .
sd migration-scale-up :=
(marking-vm-loaded ; (vm-instance-migration) ! ; unmarking-vm-loaded)
     ! .
strat migration-scale-down : @ CloudBigraph .
sd migration-scale-down :=
(marking-vm-underused ; (vm-instance-migration) ! ; unmarking-vm-
    underused) ! .
strat migration-strategy : @ CloudBigraph .
sd migration-strategy :=
(enduser-request-plan | (migration-scale-up ; migration-scale-down)
    !) * .
endsm)
```

La première stratégie introduite `enduser-request-plan` permet de gérer les cycles de vie des requêtes des utilisateurs finaux dans un système cloud, depuis leur création, passant par leur envois au système cloud, ensuite le transfert vers les applications cibles et enfin la destruction après leur satisfaction. Cette stratégie permet de générer et simuler des fluctuations de charge de travail dans le système élastique basé cloud. Elle est utilisée pour concevoir les autres stratégies d'élasticité.

Nous définissons ensuite les deux stratégies `horizontal-scale-up` et `horizontal-scale-down` qui permettent de spécifier à quel moment le système cloud doit répliquer une machine virtuelle et aussi quand il doit supprimer une instance de machine virtuelle. Ces deux stratégies sont utilisées pour décrire la mise à l'échelle horizontale (`horizontal-scale-strategy`) dans le système cloud. De manière similaire, deux autres stratégies `vertical-scale-up` et `vertical-scale-down` sont définies pour décrire la stratégie de la mise à l'échelle verticale (`vertical-scale-strategy`). Elles sont utilisées pour augmenter et diminuer quand il est nécessaire, la quantité de ressources allouées à une instance de machine virtuelle. La dernière stratégie `migration-strategy` spécifie la méthode de migration de machine virtuelle. Elle se base principalement sur les deux stratégies `migration-scale-up` et `migration-scale-down` qui indiquent dans quelle circonstance une instance de machine virtuelle doit migrer d'un serveur hôte a un autre (par exemple, pour être en mesure de mettre la machine virtuelle à l'échelle) et dans quelle circonstance elle doit retourner vers son serveur hôte original.

Dans ce module de stratégies Maude, nous avons défini un ensemble de stratégies qui peuvent gérer l'élasticité au niveau cloud infrastructure en contrôlant l'ap-

plication des règles de réécriture. Par ailleurs, il est possible d'introduire d'autres stratégies d'élasticité dans le module `Elastic_Cloud_Controller` selon le besoin et le contexte d'application. À titre d'exemple, on peut définir d'autres stratégies qui opèrent dans les autres niveaux cloud tels que les niveaux de répartition de charge, plate-forme ou application. De plus, on peut spécifier des stratégies hybrides basées sur deux ou trois méthodes d'élasticité employées ensemble pour assurer l'élasticité dans un systèmes élastique basé cloud.

Malgré les grands potentiels fournis par ce langage de stratégies associé au Maude, il reste qu'un prototype qui est toujours en cours de développement. Pour le moment, on ne peut pas inclure les stratégies d'élasticité définie dans la procédure de vérification de l'élasticité en utilisant les différentes techniques de base fournit dans l'environnement Maude. Ce langage a été mis en œuvre comme un ensemble de modules connexes définis sur Full Maude dans un seul fichier maude-strat.maude qui n'a pas été encore intégré dans Core Maude.

7.5 Vérification formelle de l'élasticité

Avec l'émergence fréquente de nouvelles technologies dans l'industrie comme le cloud computing et les systèmes élastiques basés cloud, la conception des systèmes informatiques robustes et performants est devenue de plus en plus une tâche complexe et difficile. La vérification de ces systèmes complexes est une étape inévitable et essentielle pour éviter les erreurs critiques et assurer la fiabilité des systèmes conçus. La méthode la plus courante pour vérifier l'exactitude d'une conception d'un système donné est la simulation. La technique de vérification basée sur la simulation est simple et facile à mettre en œuvre. Cependant, l'inconvénient majeur de cette technique est le fait qu'il est impossible de couvrir tous les états et les chemins potentiels d'exécutions du système à vérifier. Par exemple, dans des environnements hautement dynamiques comme les systèmes élastiques basés cloud ou le nombre d'états est indéterminé, la vérification de tous les chemins d'exécution possibles en utilisant la simulation aurait besoin d'un temps infini.

L'utilisation d'une technique de vérification formelle pour assurer l'absence d'erreurs dans un système donné représente la méthode la plus efficace. La vérification formelle de modèles ou le "model-checking" [Clarke 1996, Chechik 2001] consiste à analyser d'une manière automatique un modèle qui représente une abstraction d'un système pour déterminer si une série de propriétés est satisfaite par ce modèle du système. Les propriétés représentent l'expression d'exigences envers un système, elles sont définies généralement sous la forme de formules de logique temporelle. Plus précisément, la technique de model-checking consiste à faire une recherche exhaustive et automatique au sein de l'ensemble des états possibles du système afin de vérifier s'ils répondent à des propriétés (désirables ou indésirables) exprimées en logique temporelle ou fournir un contre-exemple montrant le chemin qui a conduit à la violation d'une ou plusieurs propriétés.

L'objectif principal de cette section est de valider la spécification Maude proposée à travers la vérification formelle des propriétés d'élasticité dans les systèmes

élastiques basés cloud selon les trois méthodes d'élasticité : la mise à l'échelle verticale, la mise à l'échelle horizontale et la migration. Pour atteindre cet objectif, nous utilisons deux techniques de vérification formelle offertes par Maude. La première technique est basée sur la technique de vérification par invariants et la commande search de Maude, alors que la deuxième technique se base sur le model-checker LTL Maude. Ainsi, nous développons deux autres modules systèmes `Elastic_Cloud_Intial_State` et `Elastic_Cloud_Preds` assurant la vérification formelle d'un système élastique cloud dans MoveElastic (voir Figure 7.4). Dans la suite, nous explicitons chacune de ces deux techniques à travers l'exemple de l'étude de cas considéré auparavant.

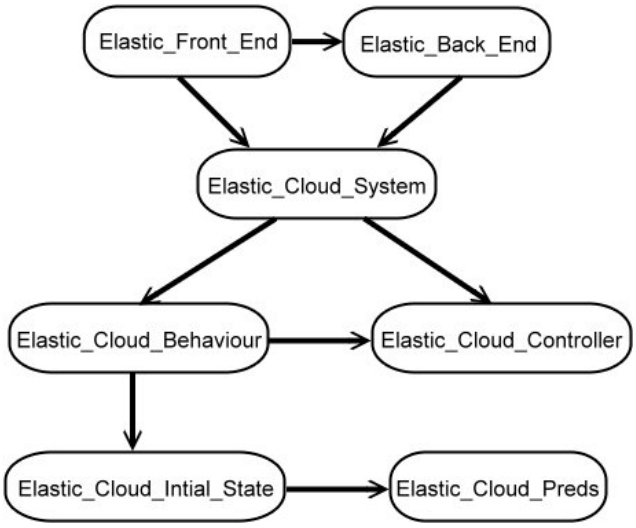


FIGURE 7.4 – Dépendance entre les modules d'implémentation et les modules de vérification

7.5.1 Vérification par invariants

La technique de vérification par invariants est une méthode de vérification formelle simple, mais très utile, qui peut être mise en œuvre dans l'environnement Maude en utilisant la commande `search` [Clavel 2011]. Cette dernière permet d'explorer à partir d'une configuration initiale représentant un point de départ du processus de vérification, les différents états d'une spécification Maude exécutable en fonction de l'application des règles de réécriture. Ceci afin de vérifier si certains de ces états correspondent à des patterns donnés, satisfaisant certaines conditions, peuvent être accessibles à partir de la configuration initiale. Ces patterns sont appelés des invariants. Un invariant sur les états d'un système est défini comme un prédicat qui est vrai dans tous les états du système qui sont accessibles à partir d'une certaine classe d'états initiaux [Clavel 2011]. Ainsi, la commande `search` de Maude permet de vérifier l'atteignabilité aussi bien que la non-atteignabilité de certains états.

Dans la suite nous montrons à travers trois étapes, comment utiliser cette tech-

nique de vérification pour vérifier les différentes méthodes d'élasticité dans un système élastique basé cloud.

Première étape. Avant de passer à la vérification, cette étape consiste à constituer une configuration initiale pour représenter l'état du système cloud. Dans ce contexte, la configuration initiale est déclarée dans le module système `Elastic_Cloud_Intial_State` (voir Listing 7.7).

Listing 7.7– Module système Elastic_Cloud_Intial_State pour la vérification par invariants

```
mod Elastic_Cloud_Initial_State is
protecting Elastic_Cloud_Behaviour .
protecting Elastic_Cloud_FrontEnd .
protecting Elastic_Cloud_BackEnd .
protecting Elastic_Cloud_System .
ops Cloud-Initial-State : -> CloudBigraph .
eq Cloud-Initial-State = EDL | EEL |
RQ 1 , RQ 3 , RQ 2 , RQ 4 , RQ 5 , RQ 6 , RQ 9 , RQ 10 , RQ 7 , RQ 0
    , RQ 8 ||
LB 1 : stable < M= 2.0 Gb , C= 2.1 Ghz , S= 70.0 Gb , B= 6.0 Mb > (
    ERQL ){ original } |
SE 1 : stable < M= 16.0 Gb , C= 5.8 Ghz , S= 3000.0 Gb , B= 60.0 Mb >
     ,
SE 2 : stable < M= 12.0 Gb , C= 4.4 Ghz , S= 2000.0 Gb , B= 45.0 Mb >
     |
VM 1 - 1 : stable < M= 3.0 Gb , C= 2.3 Ghz , S= 1000.0 Gb , B= 20.0
    Mb >{ original } ,
VM 2 - 1 : stable < M= 4.0 Gb , C= 2.1 Ghz , S= 800.0 Gb , B= 20.0 Mb
     >{ original } |
S 1 - 1 : stable < M= 0.7 Gb , C= 0.6 Ghz , S= 70.0 Gb , B= 2.0 Mb >(
     ERQL ){ original } ,
S 2 - 1 : stable < M= 0.8 Gb , C= 0.8 Ghz , S= 100.0 Gb , B= 1.5 Mb
    >( ERQL ){ original } ,
S 3 - 2 : stable < M= 1.2 Gb , C= 1.1 Ghz , S= 110.0 Gb , B= 3.0 Mb
    >( ERQL ){ original } ,
S 4 - 2 : stable < M= 0.5 Gb , C= 0.5 Ghz , S= 60.0 Gb , B= 1.0 Mb >(
     ERQL ){ original } [1000] .
Endm
```

Deuxième étape. À ce stade, nous définissons un ensemble de prédicats pertinents qui servent plus tard à l'expression des propriétés soumises à la vérification. Dans notre exemple, les différents prédicats introduits pour représenter les propriétés d'élasticité sont déclarés dans le module `Elastic_Cloud_Preds` donné dans le Listing 7.8.

Listing 7.8– Module système Elastic_Cloud_Preds pour la vérification par invariants

```
mod Elastic_Cloud_Preds is
protecting Elastic_Cloud_Initial_State .
protecting Elastic_Cloud_Behaviour .
```

```
protecting Elastic_Cloud_FrontEnd .
protecting Elastic_Cloud_BackEnd .
protecting Elastic_Cloud_System .
protecting BOOL .
vars nn n1 n2 n3 n4 n5 n6 n7 n8 n9 : Nat .
vars f1 f2 f3 : Float . var b : BackEnd .
var f : FrontEnd . var eul : EndUserList .
var dvl : DeveloperList . vars rl1 rl2 rl3 rl4 rl5 : RequestList .
vars r1 r2 r3 r4 r5 r6 r7 r8 r9 : Resources . vars l :
    LoadBalancerList .
vars se : ServerList . vars v1 v2 : VirtualMachineList .
vars c1 c2 : ContainerList . vars s1 s2 : ServiceList .
var as : AllocServiceList . var ds : DepServiceList .
var cs : ClientState . var cc : ComponentState .
ops scale-up-horizontally scale-down-horizontally
scalling-with-migration scale-up-vertically scale-down-vertically :
    CloudBigraph -> Bool [ctor] .
eq scale-up-horizontally( f || l | se | v1 , VM n1 - n2 : stable < r1
     >{ copy } | s1 [ nn ] ) = true .
eq scale-up-horizontally( CB:CloudBigraph ) = false [owise] .
eq scale-down-horizontally( f || l | se | v1 , VM n1 - n2 : underused
     < r1 >{ destroyed } | s1 [ nn ] ) = true .
eq scale-down-horizontally( CB:CloudBigraph ) = false [owise] .
eq scale-up-vertically( f || l | se | v1 , VM n1 - n2 : stable < r1
    >{ increased } | s1 [ nn ] ) = true .
eq scale-up-vertically( CB:CloudBigraph ) = false [owise] .
eq scale-down-vertically( f || l | se | v1 , VM n1 - n2 : underused <
     r1 >{ decreased } | s1 [ nn ] ) = true .
eq scale-down-vertically( CB:CloudBigraph ) = false [owise] .
eq scalling-with-migration( f || l | se | v1 , VM n1 - n2 : stable <
    r1 >{ migrated } | s1 [ nn ] ) = true .
eq scalling-with-migration( CB:CloudBigraph ) = false [owise] .
endm
```

Troisième étape. La dernière phase consiste à lancer la vérification via la commande `search` pour tester l'atteignabilité de certains états à partir d'une configuration initiale (`Cloud-Initial-State`) après l'exécution d'un nombre de règles de réécriture représentant des différentes actions d'élasticités. Dans ce contexte, l'élasticité horizontale (mise à l'échelle horizontale) est vérifiée en assurant que le système cloud est en train de répliquer/supprimer les instances de machines virtuelles (ou conteneurs, équilibreur de charges, applications dans autre scénarios) selon les fluctuations de la charge de travail. Nous vérifions l'élasticité verticale (mise à l'échelle verticale) en cherchant les états qui correspondant à des configuration intermédiaires représentant l'augmentation/diminution des ressources allouée aux instances dans le système élastique basé cloud. Enfin, la migration est vérifiée lorsqu'une ou plusieurs instances de machines virtuelles, conteneurs, équilibreurs de charge et applications sont transférées d'un hôte chargé à un autre hôte moins chargé. Ces différentes vérifications sont exprimées par les requêtes du Listing 7.9.

Listing 7.9– Requêtes search pour la vérification des propriétés d'élasticité

```
search in Elastic_Cloud_Preds : Cloud-Initial-State
=>* CB:CloudBigraph such that scale-up-horizontally (CB:CloudBigraph)
     == true .
search in Elastic_Cloud_Preds : Cloud-Initial-State
=>* CB:CloudBigraph such that scale-down-horizontally (CB:
    CloudBigraph) == true .
search in Elastic_Cloud_Preds : Cloud-Initial-State
=>* CB:CloudBigraph such that scale-up-vertically (CB:CloudBigraph)
    == true .
search in Elastic_Cloud_Preds : Cloud-Initial-State
=>* CB:CloudBigraph such that scale-down-vertically (CB:CloudBigraph)
     == true .
search in Elastic_Cloud_Preds : Cloud-Initial-State
=>* CB:CloudBigraph such that scalling-with-migration (CB:
    CloudBigraph) == true .
```

Les résultats de la vérification de la mise à l'échelle horizontale au niveau infrastructure sont donnés dans la Figure 7.5. Ces résultats montrent qu'il est possible après un certain nombre de réécritures, d'atteindre un état où une instance d'une machine virtuelle a été répliquée ou supprimée. Ceci pour assurer un bon niveau de qualité de service tout en réduisant le coût d'exploitation.

```
Maude> search in Elastic_Cloud_Preds : Cloud-Initial-State =>* CB:CloudBigraph such
that scale-up-horizontally(CB:CloudBigraph) == true = true .

Solution 1 (state 313)
states: 314  rewrites: 1789 in 2674921186ms cpu (64ms real) (0 rewrites/second)
CB:CloudBigraph --> EDL | EEL | RQ 0,RQ 1,RQ 2,RQ 3,RQ 4,RQ 5,RQ 6,RQ 7,RQ 8,RQ
9,RQ 10 || LB 1 : stable < M= 2.0 Gb,C= 2.1000000000000001 Ghz,S= 7.0e+1
Gb,B= 6.0 Mb >(ERQL){original} | SE 1 : stable < M= 1.6e+1 Gb,C=
5.7999999999999998 Ghz,S= 3.0e+3 Gb,B= 6.0e+1 Mb >,SE 2 : stable < M=
1.2e+1 Gb,C= 4.4000000000000004 Ghz,S= 2.0e+3 Gb,B= 4.5e+1 Mb > | VM 1 - 1
: loaded < M= 3.0 Gb,C= 2.2999999999999998 Ghz,S= 1.0e+3 Gb,B= 2.0e+1 Mb >{
original},VM 2 - 1 : stable < M= 4.0 Gb,C= 2.1000000000000001 Ghz,S= 8.0e+2
Gb,B= 2.0e+1 Mb >{original},VM 11 - 1 : stable < M= 3.0 Gb,C=
2.2999999999999998 Ghz,S= 1.0e+3 Gb,B= 2.0e+1 Mb >{copy} | S 1 - 1 : stable
< M= 6.9999999999999996e-1 Gb,C= 5.9999999999999998e-1 Ghz,S= 7.0e+1 Gb,B=
2.0 Mb >(ERQL){original},S 2 - 1 : stable < M= 8.0000000000000004e-1 Gb,C=
8.0000000000000004e-1 Ghz,S= 1.0e+2 Gb,B= 1.5 Mb >(ERQL){original},S 3 - 2
: stable < M= 1.2 Gb,C= 1.1000000000000001 Ghz,S= 1.1e+2 Gb,B= 3.0 Mb >(
ERQL){original},S 4 - 2 : stable < M= 5.0e-1 Gb,C= 5.0e-1 Ghz,S= 6.0e+1 Gb,
B= 1.0 Mb >(ERQL){original}[8]

Solution 2 (state 329)
states: 330  rewrites: 1921 in 2674921186ms cpu (71ms real) (0 rewrites/second)
CB:CloudBigraph --> EDL | EEL | RQ 0,RQ 1,RQ 2,RQ 3,RQ 4,RQ 5,RQ 6,RQ 7,RQ 8,RQ
9,RQ 10 || LB 1 : stable < M= 2.0 Gb,C= 2.1000000000000001 Ghz,S= 7.0e+1
Gb,B= 6.0 Mb >(ERQL){original} | SE 1 : stable < M= 1.6e+1 Gb,C=
5.7999999999999998 Ghz,S= 3.0e+3 Gb,B= 6.0e+1 Mb >,SE 2 : stable < M=
1.2e+1 Gb,C= 4.4000000000000004 Ghz,S= 2.0e+3 Gb,B= 4.5e+1 Mb > | VM 1 - 1
: stable < M= 3.0 Gb,C= 2.2999999999999998 Ghz,S= 1.0e+3 Gb,B= 2.0e+1 Mb >{
original},VM 2 - 1 : loaded < M= 4.0 Gb,C= 2.1000000000000001 Ghz,S= 8.0e+2
Gb,B= 2.0e+1 Mb >{original},VM 21 - 1 : stable < M= 4.0 Gb,C=
2.1000000000000001 Ghz,S= 8.0e+2 Gb,B= 2.0e+1 Mb >{copy} | S 1 - 1 : stable
< M= 6.9999999999999996e-1 Gb,C= 5.9999999999999998e-1 Ghz,S= 7.0e+1 Gb,B=
2.0 Mb >(ERQL){original},S 2 - 1 : stable < M= 8.0000000000000004e-1 Gb,C=
8.0000000000000004e-1 Ghz,S= 1.0e+2 Gb,B= 1.5 Mb >(ERQL){original},S 3 - 2
: stable < M= 1.2 Gb,C= 1.1000000000000001 Ghz,S= 1.1e+2 Gb,B= 3.0 Mb >(
ERQL){original},S 4 - 2 : stable < M= 5.0e-1 Gb,C= 5.0e-1 Ghz,S= 6.0e+1 Gb,
B= 1.0 Mb >(ERQL){original}[8]

Solution 3 (state 787)
states: 788  rewrites: 5243 in 2674921186ms cpu (164ms real) (0
rewrites/second)
```

FIGURE 7.5 – Résultats de la vérification de l'élasticité horizontale

7.5.2 Vérification via le model-checker LTL

Le système Maude est équipé de plusieurs autres outils et techniques d'analyse formelle qui permettent de réaliser des vérifications plus profondes et plus exhaustives. Dans cette section, nous illustrons une autre technique de vérification dans laquelle nous employons le model-checker Maude basé sur la logique temporelle linéaire LTL. Ce model-checker prend en entrée un état initial et une formule LTL et retourne soit le booléen true si la formule est satisfaite, ou un contre-exemple quand elle n'est pas satisfaite [Clavel 2011].

L'environnement Maude prend en charge ce type de vérification via un ensemble de modules connexes, introduits pour permettre la définition des formules LTL et vérification de leur satisfaction. Ces différents modules sont regroupés dans le fichier model-checker.maude, auquel nous avons ajouter d'autres modules spécifiques au domaine d'applications que nous expliquerons à travers les trois étapes suivantes :

Première étape. Comme nous l'avons fait déjà pour la technique de vérification par invariants, avant de procéder à la vérification, nous définissons le module `Elastic_Cloud_Intial_State` qui ressemble à celui défini antérieurement (voir Listing 7.7). Il contient la configuration initiale du système cloud.

Deuxième étape. Nous introduisons dans cette étape le module `Elastic_Cloud_Preds` (voir Listing 7.10) qui regroupe les différents prédicats pertinents du système. Particulièrement, dans ce module nous déclarons les prédicats `scale-up-horizontally`, `scale-down-horizontally`, `scale-up-vertically`, `scale-down-vertically` et `scalling-with-migration` qui sont utilisés pour exprimer les propriétés associées aux trois méthodes d'élasticité horizontale, verticale et migration définies dans le Listing 7.11. À titre d'exemple, le prédicat `scale-up-vertically` exprime une configuration quelconque où le système cloud est mis à l'échelle pour répondre à une augmentation dans la charge de travail, et le prédicat `scalling-with-migration` spécifie un état ou une instance virtuelle a été migrée d'un hôte chargé à un autre hôte moins chargé.

Listing 7.10– Module système Elastic_Cloud_Preds pour LTL Maude

```
load model-checker.maude
mod Elastic_Cloud_Preds is
including MODEL-CHECKER . including LTL-SIMPLIFIER .
including SATISFACTION . protecting Elastic_Cloud_FrontEnd .
protecting Elastic_Cloud_BackEnd . protecting Elastic_Cloud_System .
protecting Elastic_Cloud_Behaviour .
subsort CloudBigraph < State .
vars nn n1 n2 n3 n4 n5 n6 n7 n8 n9 : Nat .
vars f1 f2 f3 : Float . var b : BackEnd .
var f : FrontEnd . var eul : EndUserList .
var dvl : DeveloperList . vars rl1 rl2 rl3 rl4 rl5 : RequestList .
vars r1 r2 r3 r4 r5 r6 r7 r8 r9 : Resources . vars l :
    LoadBalancerList . vars se : ServerList . vars v1 v2 :
    VirtualMachineList .
vars c1 c2 : ContainerList . vars s1 s2 : ServiceList .
var as : AllocServiceList . var ds : DepServiceList .
var cs : ClientState . var cc : ComponentState .
```

```
var CB : CloudBigraph . var P : Prop .
op scale-up-horizontally : -> Prop .
op scale-down-horizontally : -> Prop .
op scale-up-vertically : -> Prop .
op scale-down-vertically : -> Prop .
op scalling-with-migration : -> Prop .
eq f || l | se | v1 , VM n1 - n2 : stable < r1 >{ copy } | s1 [ nn ]
    |= scale-up-horizontally = true .
eq f || l | se | v1 , VM n1 - n2 : underused < r1 >{ destroyed } | s1
     [ nn ] |= scale-down-horizontally = true .
eq f || l | se | v1 , VM n1 - n2 : stable < r1 >{ increased } | s1 [
    nn ] |= scale-up-vertically  = true .
eq f || l | se | v1 , VM n1 - n2 : underused < r1 >{ decreased } | s1
     [ nn ] |= scale-down-vertically  = true .
eq f || l | se | v1 , VM n1 - n2 : stable < r1 >{ migrated } | s1 [
    nn ] |= scalling-with-migration  = true .
eq CB |= P = false [owise] .
endm
```

Listing 7.11– Formule LTL exprimant des propriétés d'élasticité

```
[] (scale-up- horizontally -> <> scale-down- horizontally) \/ (scale-
    down- horizontally -> <> scale-up- horizontally) --- Propriété 1
    ---
[] (scale-up-vertically -> <> scale-down-vertically) \/ (scale-down-
    vertically -> <> scale-up-vertically) --- Propriété 2 ---
[] (scalling-with-migration -> <> scalling-with-migration) ---
    Propriété 3 ---
```

Troisième étape. Une fois que tous les éléments nécessaires pour entamer de la vérification ont tous été définis, nous pouvons passer à la dernière étape qui consiste à lancer l'exécution de la fonction `ModelCheck` du model-cheker LTL pour obtenir les résultats de vérification de chacune des propriétés exprimées en logique temporelle linéaire (LTL). Le moteur de vérification de Maude prend une configuration initiale et une formule LTL comme paramètres et renvoie soit le booléen vrai si la formule est satisfaite, ou un contre-exemple, si elle est violée.

```
reduce in Elastic_Cloud_Initial_State : modelCheck(Cloud-Initial-State, [](
    scale-up-vertically -> <> scale-down-vertically) \/ (scale-down-vertically
    -> <> scale-up-vertically)) .
rewrites: 3690077 in 4834116879ms cpu (7ms real) (0 rewrites/second)
result Bool: true
```

FIGURE 7.6 – Résultat de vérification d'élasticité verticale sous LTL Maude

Le résultat (vrai) de vérification de la propriété d'élasticité verticale (propriété 2 dans le Listing 7.11) est montrée dans la Figure 7.6. La commande exprime qu'à partir de la configuration initiale Cloud-Initial-State, cette propriété tente de vérifier s'il est toujours possible pour un système élastique basé cloud de diminuer ou augmenter éventuellement la quantité de ressources informatiques allouée à une instance cloud après avoir déjà augmenté ou diminué ces ressources au préalable. Nous

notons que `[]`, `->`, `<>` et $\vee$ représentent respectivement les opérateurs temporels henceforth, implies , eventually et or.

Par contre, si nous tentons de vérifier la négation de cette propriété d'élasticité verticale (propriété 2 dans le Listing 7.11), on obtient le résultat montré dans la Figure 7.7. Ce résultat est constitué de plusieurs contre-exemples montrant qu'il y a des états intermédiaires prouvant que le système élastique basé cloud augmente/-diminue verticalement l'allocation de ces ressources lorsqu'il est nécessaire.

```
reduce in Elastic_Cloud_Initial_State : modelCheck(Cloud-Initial-State, ~ ([](
    scale-up-vertically -> <> scale-down-vertically) \/ (scale-down-vertically
    -> <> scale-up-vertically))) .
rewrites: 48894 in 8360962608ms cpu (32ms real) (0 rewrites/second)
result ModelCheckResult: counterexample({EDL | EEL | RQ 0,RQ 1,RQ 2,RQ 3,RQ 4,
    RQ 5,RQ 6,RQ 7,RQ 8,RQ 9,RQ 10 || LB 1 : stable < M= 2.0 Gb,C=
    2.1000000000000001 Ghz,S= 7.0e+1 Gb,B= 6.0 Mb >(ERQL){original} | SE 1 :
    stable < M= 1.6e+1 Gb,C= 5.7999999999999998 Ghz,S= 3.0e+3 Gb,B= 6.0e+1 Mb
    >,SE 2 : stable < M= 1.2e+1 Gb,C= 4.4000000000000004 Ghz,S= 2.0e+3 Gb,B=
    4.5e+1 Mb > | VM 1 - 1 : stable < M= 3.0 Gb,C= 2.2999999999999998 Ghz,S=
    1.0e+3 Gb,B= 2.0e+1 Mb >{original},VM 2 - 1 : stable < M= 4.0 Gb,C=
    2.1000000000000001 Ghz,S= 8.0e+2 Gb,B= 2.0e+1 Mb >{original} | S 1 - 1 :
    stable < M= 6.9999999999999996e-1 Gb,C= 5.9999999999999998e-1 Ghz,S= 7.0e+1
    Gb,B= 2.0 Mb >(ERQL){original},S 2 - 1 : stable < M= 8.0000000000000004e-1
    Gb,C= 8.0000000000000004e-1 Ghz,S= 1.0e+2 Gb,B= 1.5 Mb >(ERQL){original},S
    3 - 2 : stable < M= 1.2 Gb,C= 1.1000000000000001 Ghz,S= 1.1e+2 Gb,B= 3.0 Mb
    >(ERQL){original},S 4 - 2 : stable < M= 5.0e-1 Gb,C= 5.0e-1 Ghz,S= 6.0e+1
    Gb,B= 1.0 Mb >(ERQL){original}[100],'send-request} {EDL | EEL | RQ 1,RQ 2,
    RQ 3,RQ 4,RQ 5,RQ 6,RQ 7,RQ 8,RQ 9,RQ 10 || LB 1 : stable < M= 2.0 Gb,C=
    2.1000000000000001 Ghz,S= 7.0e+1 Gb,B= 6.0 Mb >(RQ 0){original} | SE 1 :
    stable < M= 1.6e+1 Gb,C= 5.7999999999999998 Ghz,S= 3.0e+3 Gb,B= 6.0e+1 Mb
    >,SE 2 : stable < M= 1.2e+1 Gb,C= 4.4000000000000004 Ghz,S= 2.0e+3 Gb,B=
    4.5e+1 Mb > | VM 1 - 1 : stable < M= 3.0 Gb,C= 2.2999999999999998 Ghz,S=
    1.0e+3 Gb,B= 2.0e+1 Mb >{original},VM 2 - 1 : stable < M= 4.0 Gb,C=
    2.1000000000000001 Ghz,S= 8.0e+2 Gb,B= 2.0e+1 Mb >{original} | S 1 - 1 :
    stable < M= 6.9999999999999996e-1 Gb,C= 5.9999999999999998e-1 Ghz,S= 7.0e+1
    Gb,B= 2.0 Mb >(ERQL){original},S 2 - 1 : stable < M= 8.0000000000000004e-1
    Gb,C= 8.0000000000000004e-1 Ghz,S= 1.0e+2 Gb,B= 1.5 Mb >(ERQL){original},S
    3 - 2 : stable < M= 1.2 Gb,C= 1.1000000000000001 Ghz,S= 1.1e+2 Gb,B= 3.0 Mb
```

FIGURE 7.7 – Partie des résultats de vérification d'élasticité verticale par la négation

7.6 Conclusion

Dans ce dernier chapitre de contribution, nous avons proposé un Framework MoVeElastie qui fournit un cadre dans lequel les systèmes basés cloud peuvent être modélisés et leurs comportements élastiques peuvent être vérifiés. Ce Framework est basé sur un couplage judicieux entre les systèmes réactifs bigraphiques et le langage Maude permettant de profiter de leurs avantages complémentaires. Pour cela, nous avons présenté dans un premier temps, un aperçu sur le principe de notre Framework MoVeElastic mis en œuvre en utilisant le langage de spécification formelle Maude. Nous avons détaillé ensuite, l'implémentation des aspects structurels et comportementaux des systèmes élastiques basés cloud et leur validation à travers la simulation. Après, nous avons illustré comment le langage de stratégies de Maude peut être utilisé dans MoVeElastic pour la spécification des différentes stratégies d'élasticité. Particulièrement, nous avons présenté trois stratégies d'élasticité pour la mise à l'échelle horizontale, la mise à l'échelle verticale, et la migration. Dans ce Framework, nous avons aussi procédé à la vérification formelle de l'élasti-

cité en proposant deux techniques de vérification formelle basées sur la technique de vérification par invariants et le model-checker LTL Maude.

Chapitre 8

Conclusion générale

Sommaire

8.1 Contributions 121
8.2 Perspectives 123

Le contexte général des travaux de cette thèse se positionne dans le domaine du cloud computing, qui a acquis très rapidement, ces dernières années, une grande popularité dans les domaines académique et industriel pour plusieurs raisons, par exemple, le modèle de facturation "pay-as-you-go", le déploiement facile et rapide des applications dans des infrastructures flexibles et évolutives et la libération des utilisateurs des préoccupations liées à la gestion du matériel.

Le cloud computing est caractérisé par la propriété d'élasticité qui n'est rien d'autre que le degré auquel un système basé cloud est capable de s'adapter de manière dynamique aux changements de la charge de travail, en approvisionnant et libérant des ressources, de telle façon à ce que les ressources fournies soient conformes à la demande du système. Cette propriété d'adaptation dynamique de la consommation de ressources en termes d'élasticité a rendu le paradigme très attrayant pour les différents types d'utilisateurs.

L'émergence des systèmes élastiques basés cloud a conduit à un nouveau type de systèmes autonomiques dans lesquels l'élasticité est un principe de conception clé. L'avantage principal de ces systèmes est leur capacité d'adapter, de manière autonomique, la consommation des ressources informatiques afin de maintenir une bonne qualité de service (QoS), tout en minimisant les coûts de fonctionnement. Dans les environnements cloud, de nombreuses solutions académiques et commerciales ont été proposées pour automatiser la gestion et la planification de deux types d'élasticité : horizontale et verticale. La mise en œuvre de l'élasticité dans un système cloud est basée sur deux types de contrôleurs d'élasticité : réactifs et proactifs, adoptant une boucle de contrôle MAPE-K (Monitor, Analyse, Plan, Execute) proposée par IBM pour les systèmes autonomiques. Néanmoins, toutes les solutions existantes actuellement, proposées par des grands acteurs du marché cloud ou bien venant du monde de recherche, sont encore immatures. En effet, d'une part, les solutions commerciales présentent beaucoup de limites et de contraintes pour leurs utilisateurs, ce qui influe sur la gestion de l'élasticité et par conséquent sur la disponibilité, la performance et parfois même sur le coût. Et d'autre part, certaines propositions du monde académique sont très coûteuses, difficiles à mettre en œuvre voire inapplicables dans les environnements cloud.

Pour une bonne gestion de l'élasticité, il est indispensable de s'appuyer sur un modèle qui permet de décrire les architectures des systèmes élastiques basés cloud, ainsi que leur comportement. La modélisation est une tâche impérative qui permet de déterminer et comprendre les changements structurels et les dépendances comportementales dans un système élastique basé cloud afin d'éviter l'émergence des comportements provoquant des situations indésirables telles que l'instabilité dans l'allocation des ressources et la dégradation de la qualité de service. Dans ce contexte, les méthodes formelles caractérisées par leur efficacité, fiabilité et précision, présentent une solution efficace pour relever ce défi et pour éliminer les ambiguïtés sémantiques et la complexité provenant de la tâche de modélisation. Ceci permet de raisonner rigoureusement sur les spécifications formelles obtenues pour démontrer leur validité à travers plusieurs techniques de simulations et vérifications formelles.

La modélisation et l'analyse formelle des systèmes cloud sont des problématiques d'actualités qui ont attiré l'attention du milieu académique et abouti à plusieurs approches en se basant sur des formalismes différents. Cependant, à cause de leur complexité, peu de travaux ont abordé les aspects comportementaux en termes d'élasticité et d'approvisionnement autonomique dans le cloud. Néanmoins, quelques tentatives de recherche récentes ont orienté leurs efforts vers la modélisation et la vérification formelle des aspects comportementaux relatifs aux systèmes élastiques basés cloud. Lors de notre étude des travaux existants dans ce domaine, nous avons soulevé l'absence d'une approche générique et complète pour modéliser et analyser ces systèmes. Ces approches formelles se concentrent essentiellement sur le niveau infrastructure du modèle cloud et l'élasticité horizontale. En outre, la majorité de ces approches proposent des solutions simples et limitées puisqu'elles ignorent les aspects architecturaux ainsi que les dépendances structurelles et comportementales internes dans les systèmes élastiques basés cloud. De plus, elles ne sont pas exhaustives, parfois non formelles, et souvent ne prennent pas en compte, lors de la modélisation des systèmes élastiques basés cloud, tous les aspects structurels et comportementaux.

8.1 Contributions

Pour pallier à ce manque et répondre à ces limitations, la contribution principale de cette thèse est la proposition d'une approche générique et exhaustive qui permet de réduire la complexité provenant des tâches de description et d'analyse des systèmes élastiques basés cloud et leur comportement. De plus et en particulier, nous avons proposé un cadre formel basé sur un couplage judicieux entre les systèmes réactifs bigraphiques et le langage Maude. Ce couplage permet d'une part la modélisation des aspects conceptuels et architecturaux des systèmes élastiques basés cloud, et d'autre part, la vérification de leur comportement élastique. Les principales contributions de cette thèse peuvent se résumer comme suit.

Tout d'abord, nous avons entamé notre travail par une étude des travaux existants qui abordent l'élasticité dans le cloud computing. Cette étude a été divisée en deux parties traitant respectivement les solutions de gestion autonomique de l'élas-

ticité et les approches formelles pour la modélisation et l'analyse des systèmes élastiques basés cloud. Dans un premier temps, nous avons présenté un certain nombre de solutions académiques qui proposent des mécanismes et des contrôleurs pour assurer l'élasticité dans des environnements cloud. Ces solutions sont souvent basées sur la boucle MAPE-K pour la planification et la gestion autonomique des ressources informatiques en termes d'élasticité horizontale et verticale à différents niveaux cloud. Ensuite, nous avons étudié différentes approches formelles de modélisation et d'analyse des systèmes élastiques basés cloud. Nous avons présenté et évalué les approches proposées en mettant en évidence quelques caractéristiques comme : le niveau de prise en charge de détails architecturaux, l'existence d'une technique d'analyse formelle, les stratégies d'élasticité, les méthodes d'élasticité et les niveaux du modèle cloud couverts. Enfin, nous avons établi une synthèse des approches formelles présentées pour récapituler les carences et les lacunes de chacune de ces approches selon les caractéristiques ainsi définies. Cette étude nous a permis de mieux cerner les différentes limites et besoins liés à la modélisation et l'analyse des systèmes élastiques basés cloud.

La première contribution de cette thèse consiste à proposer un modèle exhaustif et formel basé sur les systèmes réactifs bigraphiques et la logique de typage. Dans ce modèle, nous avons essayé de couvrir tous les aspects conceptuels et architecturaux relatifs aux systèmes élastiques basés cloud pour permettre une modélisation beaucoup plus fine de ces systèmes. Le modèle bigraphique ainsi obtenu permet de décrire les éléments architecturaux dans un système cloud, les relations et les dépendances entre ces éléments et aussi les contraintes structurelles et relationnelles qui en régissent le comportement élastique. Ainsi, la structure d'un bigraphe modélisant un système élastique basé cloud comprend trois bigraphes élémentaires et indépendants qui interagissent les uns avec les autres à travers leurs interfaces. Ces bigraphes modélisent respectivement le front-end, le back-end et le contrôleur d'élasticité d'un système élastique basé cloud. Nous avons défini la structure du contrôleur d'élasticité en adoptant le principe de la boucle de contrôle autonomique MAPE-K (Monitor, Analyse, Plan, Execute). Pour assurer une représentation précise de la sémantique des systèmes élastiques basés cloud, nous avons utilisé la logique de typage associée aux systèmes réactifs bigraphiques. Ceci nous a permis d'assurer à travers des contraintes sur la localité et la connectivité que les modèles décrits ont une structure significative.

La deuxième contribution consiste à la définition d'un ensemble règles de réaction paramétriques pour fournir des mécanismes nécessaires à la modélisation de tous les aspects du comportement dynamique des systèmes élastiques basés cloud, tout en tenant compte des points de vue client et cloud. Pour ce faire, deux catégories principales de règles de réaction bigraphiques ont été définies. Dans un premier temps, nous avons modélisé les interactions client/application (front-end/back-end) dans le cloud à travers une première catégorie. Ensuite, le comportement élastique des systèmes basés cloud en termes de méthodes d'élasticité applicables aux différents niveaux du modèle cloud computing (service, plate-forme, infrastructure et répartition de charge) a été défini à travers une deuxième catégorie. Nous avons

introduit une catégorie supplémentaire pour imposer un contrôle sur l'exécution des règles de réaction et apprivoiser leur nature non déterministe. La modélisation des stratégies d'élasticité a été réalisée en combinant les règles de réaction définies avec des prédicats BiLog.

Les deux premières contributions théoriques ont été implémentées dans un prototype d'un Framework, nommé MoVeElastic. Ce Framework nous a permis de valider l'approche proposée en présentant un cadre formel complet pour la modélisation des systèmes basés cloud et la vérification de leur comportement élastique. MoVeElastic a été implémenté à travers un couplage judicieux entre les systèmes réactifs bigraphiques et le langage Maude. Ce couplage judicieux nous a permis d'en tirer les avantages de chacun de ces deux formalismes. Nous avons validé notre approche de deux manières différentes : la simulation des spécifications exécutables obtenues et la validation par la vérification formelle des propriétés. Pour cette dernière, nous avons procédé à deux techniques de vérification fournies par l'environnement Maude. La première est basée sur la technique de vérification par invariants, alors que la deuxième est basée sur la vérification par modèle à travers le model-checker LTL Maude.

8.2 Perspectives

Les perspectives à court et à long terme de ce travail scientifique sont multiples et prometteuses. Nous soulignons ici quatre perspectives pertinentes : la première concerne l'amélioration de la prise en compte des stratégies d'élasticité de notre approche, la seconde est la définition et l'évaluation de différents types de stratégies d'élasticité, la troisième se rapporte au développement d'un outil complètement assisté pour concevoir et analyser les systèmes élastiques basés cloud, et la dernière concerne la vérification des propriétés liées à l'élasticité.

— L'amélioration de la prise en compte des stratégies d'élasticité. L'une des pistes de recherche consiste à raffiner la spécification des stratégies d'élasticité. La représentation de ces stratégies dans notre approche a été supportée par le langage de stratégies associé à Maude. Cependant, ce langage est un prototype qui est toujours en cours de développement. Par conséquent, on n'a pas pu inclure dans la procédure de vérification de l'élasticité les stratégies d'élasticité définies dans cette thèse. Il serait donc intéressant d'améliorer la prise en compte des stratégies d'élasticité dans notre approche, une fois la version finale du langage est prête, ou à travers l'exploitation de la méta-programmation et la caractéristique de réflexion offertes par le langage Maude.

— La définition et l'évaluation des stratégies d'élasticité. une des perspectives à long terme est la spécification et l'analyse des stratégies d'élasticité proactives et/ou réactives en fonction des règles de réaction ainsi définies dans la thèse. L'idée est de définir et d'évaluer différents types de stratégies d'élasticité, et de déterminer quel type de stratégie est le plus approprié pour un type d'événements (internes ou externes) dans le système cloud.

— Le développement d'un outil complètement assisté. Il serait intéressant de développer un outil qui est complètement automatisé et assisté depuis la phase de modélisation jusqu'à l'analyse des résultats obtenus de la vérification des propriétés. La première fonctionnalité nécessaire pour l'outil consiste en un modeleur graphique permettant de créer des modèles par intention en s'appuyant sur la technologie EMF et le langage Java. Deux autres composants pour l'outil seront développés en Java, le premier composant est un traducteur qui permettra la génération des spécifications Maude à partir des modèles bigraphiques conçus alors que le deuxième composant, permettra d'analyser les résultats des vérifications selon des critères bien définis.

— La vérification des propriétés liées à l'élasticité. Une autre perspective importante de notre travail est d'identifier, d'exprimer et de vérifier différentes propriétés liées à l'élasticité. Parmi ces propriétés on peut mentionner, la plasticité, qui désigne un système cloud qui est incapable de retourner à sa configuration initiale après une procédure d'adaptation. La résonance, qui représente l'apparition d'oscillations permanentes dans l'allocation des ressources en réponse aux changements de la charge de travail. La réflexivité qui se réfère à la vitesse de la réaction d'un système élastique cloud lors d'un changement de la charge. Enfin, la précision du système élastique basé cloud à allouer la quantité de ressources adéquate pour son bon fonctionnement.

Bibliographie

[Agrawal 2011] Divyakant Agrawal, Amr El Abbadi, Sudipto Das et Aaron J. Elmore. *Database Scalability, Elasticity, and Autonomy in the Cloud*. In Proceedings of the 16th International Conference on Database Systems for Advanced Applications - Volume Part I, DASFAA'11, pages 2–15, Berlin, Heidelberg, 2011. Springer-Verlag. (Cité en page 30.)

[Al-Shishtawy 2013] Ahmad Al-Shishtawy et Vladimir Vlassov. *ElastMan : Autonomic Elasticity Manager for Cloud-based Key-value Stores*. In Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing, HPDC '13, pages 115–116, New York, NY, USA, 2013. ACM. (Cité en page 33.)

[Ali-Eldin 2012] Ahmed Ali-Eldin, Johan Tordsson et Erik Elmroth. *An adaptive hybrid elasticity controller for cloud infrastructures*. In 2012 IEEE Network Operations and Management Symposium, pages 204–212. IEEE, 2012. (Cité en pages 4 et 35.)

[Amazon EC2 2006] Amazon EC2. http://aws.amazon.com/en/ec, 2006. Dernière consultation le 09.12.2015. (Cité en pages 9 et 93.)

[Amrhein 2009] Dustin Amrhein et Scott Quint. *Cloud computing for the enterprise : Part 1 : Capturing the cloud*. Rapport technique UCB/EECS-2009-28, IBM, Apr 2009. (Cité en page 10.)

[Amziani 2013a] Mourad Amziani, Kais Klai, Tarek Melliti et Samir Tata. *Time-based evaluation of service-based business process elasticity in the cloud*. In Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on, volume 1, pages 573–580. IEEE, 2013. (Cité en pages 40, 44 et 45.)

[Amziani 2013b] Mourad Amziani, Tarek Melliti et Samir Tata. *Formal modeling and evaluation of service-based business process elasticity in the cloud*. In Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE), 2013 IEEE 22nd International Workshop on, pages 284–291. IEEE, 2013. (Cité en pages 39, 41, 44 et 45.)

[Amziani 2013c] Mourad Amziani, Tarek Melliti et Samir Tata. *Formal modeling and evaluation of stateful service-based business process elasticity in the cloud*. In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", pages 21–38. Springer, 2013. (Cité en pages 40, 44 et 45.)

[Andersen 2007] Paul Andersen. *What is Web 2.0 ? : ideas, technologies and implications for education*. http://www.jisc.ac.uk/media/documents/techwatch/tsw0701b.pdf, 2007. Dernière consultation le 12.01.2016. (Cité en page 22.)

[Armbrust 2009] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica et Matei Zaharia. *Above the Clouds : A Berkeley View of Cloud Computing.* Rapport technique UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009. (Cité en page 10.)

[Bantz 2003] D. F. Bantz, C. Bisdikian, D. Challener, J. P. Karidis, S. Mastrianni, A. Mohindra, D. G. Shea et M. Vanover. *Autonomic Personal Computing.* IBM Syst. J., vol. 42, no. 1, pages 165–176, Janvier 2003. (Cité en page 26.)

[Benzadri 2014] Zakaria Benzadri, Faiza Belala et Chafia Bouanaka. Towards a formal model for cloud computing, pages 381–393. Springer International Publishing, Cham, 2014. (Cité en page 39.)

[Berns 2009] Andrew Berns et Sukumar Ghosh. *Dissecting Self-* Properties.* In Proceedings of the 2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO '09, pages 10–19, Washington, DC, USA, 2009. IEEE Computer Society. (Cité en page 26.)

[Bersani 2014] Marcello M Bersani, Domenico Bianculli, Schahram Dustdar, Alessio Gambi, Carlo Ghezzi et Srđan Krstić. *Towards the formalization of properties of cloud-based elastic systems.* In Proceedings of the 6th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems, pages 38–47. ACM, 2014. (Cité en pages , 41, 44, 45, 69 et 70.)

[Blake 2011] James Blake et Nathaniel Borenstein. *Cloud Computing Standards : Where's the Beef?* IEEE Internet Computing, vol. 15, no. undefined, pages 74–78, 2011. (Cité en page 11.)

[Booth 2004] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris et David Orchard. *Web Services Architecture.* Rapport technique, W3C Working Group Note, 2004. Dernière consultation le 22.01.2016. (Cité en page 20.)

[Bundgaard 2008] Mikkel Bundgaard, Arne John Glenstrup, Thomas Hildebrandt, Espen Højsgaard et Henning Niss. *Formalizing WS-BPEL and higher order mobile embedded business processes in the bigraphical programming languages (BPL) tool.* Rapport technique, Technical Report TR-2008-103, IT University of Copenhagen, 2008. (Cité en page 59.)

[Burbeck 2000] Steve Burbeck. *The tao of e-business services : the evolution of Web applications into service-oriented components with Web services.* Rapport technique, IBM DeveloperWorks, 2000. Dernière consultation le 24.01.2016. (Cité en page 20.)

[Buyya 2008] Rajkumar Buyya. *Market-Oriented Cloud Computing : Vision, Hype, and Reality of Delivering Computing As the 5th Utility.* In Proceedings of the 2008 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '08, pages 1–, Washington, DC, USA, 2008. IEEE Computer Society. (Cité en page 10.)

[CERN 2006] CERN. *The Grid*. http://cdsweb.cern.ch/record/976156/files/it-brochure-2006-002.pdf, 2006. Dernière consultation le 07.12.2015. (Cité en page 9.)

[Chechik 2001] Marsha Chechik et John Gannon. *Automatic analysis of consistency between requirements and designs*. IEEE Transactions on Software Engineering, vol. 27, no. 7, pages 651–672, 2001. (Cité en page 111.)

[Clarke 1996] Edmund M Clarke et Jeannette M Wing. *Formal methods : State of the art and future directions*. ACM Computing Surveys (CSUR), vol. 28, no. 4, pages 626–643, 1996. (Cité en page 111.)

[Clavel 2007] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer et Carolyn Talcott. All about maude - a high-performance logical framework : How to specify, program and verify systems in rewriting logic. Springer-Verlag, Berlin, Heidelberg, 2007. (Cité en pages 47, 60, 66 et 98.)

[Clavel 2011] Manuel Clavel, Francisco Duran, Steven Eker, Patrick Lincoln, Narciso Marti-Oliet, José Meseguer et Carolyn Talcott. *Maude Manual (Version 2.6)(January 2011)*, 2011. (Cité en pages 47, 60, 66, 112 et 116.)

[Cloud Industry Forum 2012] Cloud Industry Forum. *UK Cloud adoption and trends for 2013*. Rapport technique, High Wycombe HP13 6DG,, 2012. (Cité en page 1.)

[Conforti 2006] Giovanni Conforti, Damiano Macedonio et Vladimiro Sassone. *BiLog : Spatial logics for bigraphs*. (Under revision), 2006. (Cité en pages 47, 59 et 84.)

[Coutinho 2013] Emanuel Ferreira Coutinho, Danielo Gonçalves Gomes et José Neuman de Souza. *An analysis of elasticity in cloud computing environments based on allocation time and resources*. In Cloud Computing and Communications (LatinCloud), 2nd IEEE Latin American Conference on, pages 7–12. IEEE, 2013. (Cité en page 30.)

[Coutinho 2015] Emanuel Ferreira Coutinho, Flávio Rubens de Carvalho Sousa, Paulo Antonio Leal Rego, Danielo Gonçalves Gomes et José Neuman de Souza. *Elasticity in cloud computing : a survey*. annals of telecommunications - annales des télécommunications, vol. 70, no. 7, pages 289–309, 2015. (Cité en page 32.)

[Creasy 1981] R. J. Creasy. *The Origin of the VM/370 Time-sharing System*. IBM J. Res. Dev., vol. 25, no. 5, pages 483–490, Septembre 1981. (Cité en page 17.)

[DiMare 2006] Jay DiMare. *Changing the way industries work : Impact of SOA - IBM*. Rapport technique, IBM, 2006. Dernière consultation le 22.01.2016. (Cité en page 20.)

[Dustdar 2011] Schahram Dustdar, Yike Guo, Benjamin Satzger et Hong-Linh Truong. *Principles of Elastic Processes*. IEEE Internet Computing, vol. 15, no. 5, pages 66–71, 2011. (Cité en pages 2 et 3.)

[Dustdar 2015] Schahram Dustdar, Alessio Gambi, Willibald Krenn et Dejan Nickovic. *A pattern-based formalization of cloud-based elastic systems.* In Proceedings of the Seventh International Workshop on Principles of Engineering Service-Oriented and Cloud Systems, pages 31–37. IEEE Press, 2015. (Cité en pages 41, 44, 45 et 69.)

[Eker 2007] Steven Eker, Narciso Martí-Oliet, José Meseguer et Alberto Verdejo. *Deduction, strategies, and rewriting.* Electronic Notes in Theoretical Computer Science, vol. 174, no. 11, pages 3–25, 2007. (Cité en pages 47, 63 et 64.)

[Ellson 2001] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North et Gordon Woodhull. *Graphviz-open source graph drawing tools.* In International Symposium on Graph Drawing, pages 483–484. Springer, 2001. (Cité en page 60.)

[Elsborg 2009] Ebbe Elsborg. *Bigraphs : Modelling, Simulation, and Type Systems.* PhD thesis, PhD thesis, IT University of Copenhagen, 2009. (Cité en page 59.)

[Foley 2008] John Foley. *A Definition of Cloud Computing.* `http://www.informationweek.com/cloud-computing/blog/archives/2008/09/a_definition_of.html`, 2008. (Cité en page 10.)

[Freitas 2012] Leo Freitas et Paul Watson. *Formalising Workflows Partitioning over Federated Clouds : Multi-level Security and Costs.* In Proceedings of the 2012 IEEE Eighth World Congress on Services, SERVICES '12, pages 219–226, Washington, DC, USA, 2012. IEEE Computer Society. (Cité en page 39.)

[Galante 2012] Guilherme Galante et Luis Carlos E. de Bona. *A Survey on Cloud Computing Elasticity.* In Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing, UCC '12, pages 263–270, Washington, DC, USA, 2012. IEEE Computer Society. (Cité en pages , 3, 30, 31, 32 et 92.)

[Gambi 2013a] Alessio Gambi, Antonio Filieri et Schahram Dustdar. *Iterative test suites refinement for elastic computing systems.* In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pages 635–638. ACM, 2013. (Cité en pages , 42, 44, 45 et 97.)

[Gambi 2013b] Alessio Gambi, Waldemar Hummer et Schahram Dustdar. *Testing elastic systems with surrogate models.* In Combining Modelling and Search-Based Software Engineering (CMSBSE), 2013 1st International Workshop on, pages 8–11. IEEE, 2013. (Cité en pages 43, 44 et 45.)

[Gambi 2016] Alessio Gambi, Mauro Pezze et Giovanni Toffetti. *Kriging-Based Self-Adaptive Cloud Controllers.* IEEE Transactions on Services Computing, vol. 9, no. 3, pages 368–381, May 2016. (Cité en page 3.)

[Garfinkel 1999] Simson Garfinkel et Harold Abelson. Architects of the information society : 35 years of the laboratory for computer science at mit. MIT Press, Cambridge, MA, USA, 1999. (Cité en page 8.)

[Gartner 2008] Gartner. http://www.gartner.com/newsroom/id/707508, 2008. Dernière consultation le 12.01.2016. (Cité en page 10.)

[Geelan 2008] Jeremy Geelan. *Twenty-One Experts Define Cloud Computing*. http://cloudcomputing.sys-con.com/node/612375, 2008. Dernière consultation le 14.01.2016. (Cité en pages 9 et 10.)

[Herbst 2013] Nikolas Roman Herbst, Samuel Kounev et Ralf Reussner. *Elasticity in cloud computing : What it is, and what it is not*. In Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13), pages 23–27, 2013. (Cité en page 30.)

[Hoenisch 2013a] Philipp Hoenisch, Stefan Schulte et Schahram Dustdar. *Workflow Scheduling and Resource Allocation for Cloud-Based Execution of Elastic Processes*. In Proceedings of the 2013 IEEE 6th International Conference on Service-Oriented Computing and Applications, SOCA '13, pages 1–8, Washington, DC, USA, 2013. IEEE Computer Society. (Cité en page 36.)

[Hoenisch 2013b] Philipp Hoenisch, Stefan Schulte, Schahram Dustdar et Srikumar Venugopal. *Self-Adaptive Resource Allocation for Elastic Process Execution*. In Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, CLOUD '13, pages 220–227, Washington, DC, USA, 2013. IEEE Computer Society. (Cité en page 36.)

[Højsgaard 2011] Espen Højsgaard et Arne J Glenstrup. *The bpl tool : A tool for experimenting with bigraphical reactive systems*. Bigraphical Languages and their Simulation, page 85, 2011. (Cité en pages 59 et 98.)

[Horn 2001] Paul Horn. *Autonomic Computing : IBM's Perspective on the State of Information Technology*. Rapport technique, IBM, 2001. (Cité en page 26.)

[Huebscher 2008] Markus C. Huebscher et Julie A. McCann. *A Survey of Autonomic Computing&Mdash ;Degrees, Models, and Applications*. ACM Comput. Surv., vol. 40, no. 3, pages 7 :1–7 :28, Août 2008. (Cité en page 26.)

[Hwang 2014] Ren-Hung Hwang, Chung-Nan Lee, Yi-Ru Chen et Da-Jing Zhang-Jian. *Cost optimization of elasticity cloud resource subscription policy*. IEEE Transactions on Services Computing, vol. 7, no. 4, pages 561–574, 2014. (Cité en pages et 34.)

[IBM Group 2005] IBM Group . *An architectural blueprint for autonomic computing*. IBM White paper, 2005. (Cité en pages , 28, 29 et 70.)

[IBM 2007] IBM. *Google and IBM Announced University Initiative to Address Internet-Scale Computing Challenges*. http://www-03.ibm.com/press/us/en/pressrelease/22414.wss, 2007. Dernière consultation le 12.01.2016. (Cité en page 9.)

[Jacob 2004] Bart Jacob, Richard Lanyon, Hogg Devaprasad et Amr F Yassin. A practical guide to the ibm autonomic computing toolkit. IBM redbooks. IBM Corporation, International Technical Support Organization, 2004. (Cité en pages 4 et 26.)

[Kaur 2014] Pankaj Deep Kaur et Inderveer Chana. *A resource elasticity framework for QoS-aware execution of cloud applications*. Future Generation Computer Systems, vol. 37, pages 14–25, 2014. (Cité en pages et 37.)

[Kephart 2003] Jeffrey O. Kephart et David M. Chess. *The Vision of Autonomic Computing*. Computer, vol. 36, no. 1, pages 41–50, Janvier 2003. (Cité en page 26.)

[Kikuchi 2014] Shinji Kikuchi et Kunihiko Hiraishi. *Improving reliability in management of cloud computing infrastructure by formal methods*. In 2014 IEEE Network Operations and Management Symposium (NOMS), pages 1–7. IEEE, 2014. (Cité en page 39.)

[Kleinrock 2005] Leonard Kleinrock. *A Vision for the Internet*. ST Journal for Research, vol. 2, no. 1, pages 4–5, November 2005. (Cité en page 8.)

[Knauth 2011] Thomas Knauth et Christof Fetzer. *Scaling Non-elastic Applications Using Virtual Machines*. In Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing, CLOUD '11, pages 468–475, Washington, DC, USA, 2011. IEEE Computer Society. (Cité en page 32.)

[Kranas 2012] Pavlos Kranas, Vasileios Anagnostopoulos, Andreas Menychtas et Theodora Varvarigou. *ElaaS : An Innovative Elasticity As a Service Framework for Dynamic Management Across the Cloud Stack Layers*. In Proceedings of the 2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS), CISIS '12, pages 1042–1049, Washington, DC, USA, 2012. IEEE Computer Society. (Cité en pages et 38.)

[Letondeur 2014] Loïc Letondeur, Xavier Etchevers, Thierry Coupaye, Fabienne Boyer et Noël De Palma. *Architectural Model and Planification Algorithm for the Self-Management of Elastic Cloud Applications*. In Proceedings of the 2014 International Conference on Cloud and Autonomic Computing, ICCAC '14, pages 172–179, Washington, DC, USA, 2014. IEEE Computer Society. (Cité en pages et 33.)

[Liu 2009] Ling Liu et M. Tamer Zsu. Encyclopedia of database systems. Springer Publishing Company, Incorporated, 1st édition, 2009. (Cité en page 22.)

[Loff 2014] João Loff et João Garcia. *Vadara : Predictive Elasticity for Cloud Applications*. In Proceedings of the 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, CLOUDCOM '14, pages 541–546, Washington, DC, USA, 2014. IEEE Computer Society. (Cité en pages , 36 et 37.)

[Mainsah 2002] Evaristus Mainsah. *Autonomic computing : the next era of computing*. Electronics Communication Engineering Journal, vol. 14, no. 1, pages 2–3, Feb 2002. (Cité en page 26.)

[Martí-Oliet 2009] Narciso Martí-Oliet, José Meseguer et Alberto Verdejo. *A rewriting semantics for Maude strategies*. Electronic Notes in Theoretical Computer Science, vol. 238, no. 3, pages 227–247, 2009. (Cité en pages 47, 63 et 64.)

[Mell 2011] Peter M. Mell et Timothy Grance. *The NIST Definition of Cloud Computing*. Rapport technique, NIST, Gaithersburg, MD, United States, 2011. (Cité en page 11.)

[Milner 1997] Robin Milner. The definition of standard ml : revised. MIT press, 1997. (Cité en page 59.)

[Milner 2006] Robin Milner. *Pure bigraphs : Structure and dynamics*. Information and computation, vol. 204, no. 1, pages 60–122, 2006. (Cité en page 47.)

[Milner 2008] Robin Milner. *Bigraphs and Their Algebra*. Electron. Notes Theor. Comput. Sci., vol. 209, pages 5–19, Avril 2008. (Cité en pages 47, 48, 51, 56, 57, 69 et 101.)

[Milner 2009] Robin Milner. The space and motion of communicating agents. Cambridge University Press, New York, NY, USA, 1st édition, 2009. (Cité en pages 47, 49, 54 et 69.)

[Mohamed 2015] Mohamed Mohamed, Mourad Amziani, Djamel Belaïd, Samir Tata et Tarek Melliti. *An autonomic approach to manage elasticity of business processes in the Cloud*. Future Generation Computer Systems, vol. 50, pages 49–61, 2015. (Cité en page 40.)

[Naskos 2014] Athanasios Naskos, Emmanouela Stachtiari, Anastasios Gounaris, Panagiotis Katsaros, Dimitrios Tsoumakos, Ioannis Konstantinou et Spyros Sioutas. *Cloud elasticity using probabilistic model checking*. arXiv preprint arXiv :1405.4699, 2014. (Cité en pages , 40, 41, 44 et 45.)

[OpenShift 2011] OpenShift. `https://openshift.redhat.com/app/`, 2011. Dernière consultation le 12.01.2016. (Cité en page 9.)

[OpenStack 2010] OpenStack. `http://www.openstack.org/`, 2010. Dernière consultation le 12.01.2016. (Cité en page 9.)

[O'Reilly 2009] Tim O'Reilly et John Battelle. Web squared : Web 2.0 five years on. " O'Reilly Media, Inc.", 2009. (Cité en page 22.)

[Parkhill 1966] Douglas F. Parkhill. The challenge of the computer utility. Numéro p. 246 de The Challenge of the Computer Utility. Addison-Wesley Publishing Company, 1966. (Cité en page 8.)

[Peltz 2003] Chris Peltz. *Web Services Orchestration and Choreography*. Computer, vol. 36, no. 10, pages 46–52, Octobre 2003. (Cité en page 19.)

[Perrone 2012] Gian Perrone, Søren Debois et Thomas T Hildebrandt. *A model checker for bigraphs*. In Proceedings of the 27th Annual ACM Symposium on Applied Computing, pages 1320–1325. ACM, 2012. (Cité en page 98.)

[Pooja 2014] Pooja et A. Pandey. *Virtual machine performance measurement*. In Engineering and Computational Sciences (RAECS), 2014 Recent Advances in, pages 1–3, March 2014. (Cité en page 18.)

[Popek 1974] Gerald J. Popek et Robert P. Goldberg. *Formal Requirements for Virtualizable Third Generation Architectures*. Commun. ACM, vol. 17, no. 7, pages 412–421, Juillet 1974. (Cité en page 17.)

[Rady 2013] Mariam Rady. *Formal definition of service availability in cloud computing using OWL.* In International Conference on Computer Aided Systems Theory, pages 189–194. Springer, 2013. (Cité en page 39.)

[Rong 2014] Mei Rong. *Modeling and analysis BPEL-based web services composition using XYZ.* In Computer Science & Education (ICCSE), 2014 9th International Conference on, pages 1083–1088. IEEE, 2014. (Cité en pages 93 et 107.)

[Schulte 2012] Stefan Schulte, Philipp Hoenisch, Srikumar Venugopal et Schahram Dustdar. *Introducing the Vienna Platform for Elastic Processes.* In International Conference on Service-Oriented Computing, pages 179–190. Springer, 2012. (Cité en pages , 35 et 36.)

[Sevegnani 2012] Michele Sevegnani. *Bigraphs with sharing and applications in wireless networks.* PhD thesis, University of Glasgow, 2012. (Cité en page 98.)

[Sevegnani 2015] Michele Sevegnani et Muffy Calder. *Bigraphs with sharing.* Theoretical Computer Science, vol. 577, pages 43–73, 2015. (Cité en page 59.)

[Sevegnani 2016] Michele Sevegnani et Muffy Calder. *BigraphER : rewriting and analysis engine for bigraphs.* 2016. (Cité en page 59.)

[Smith 2005] Jim Smith et Ravi Nair. Virtual machines : Versatile platforms for systems and processes (the morgan kaufmann series in computer architecture and design). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. (Cité en page 17.)

[Soundararajan 2010] Vijayaraghavan Soundararajan et Kinshuk Govil. *Challenges in Building Scalable Virtualized Datacenter Management.* SIGOPS Oper. Syst. Rev., vol. 44, no. 4, pages 95–102, Décembre 2010. (Cité en page 1.)

[Vaquero 2009] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres et Maik Lindner. *A Break in the Clouds : Towards a Cloud Definition.* SIGCOMM Comput. Commun. Rev., vol. 39, no. 1, pages 50–55, Décembre 2009. (Cité en page 10.)

[Vaquero 2011] Luis M. Vaquero, Luis Rodero-Merino et Rajkumar Buyya. *Dynamically Scaling Applications in the Cloud.* SIGCOMM Comput. Commun. Rev., vol. 41, no. 1, pages 45–52, 2011. (Cité en pages 3 et 32.)

[Wang 2010] Lizhe Wang, Gregor von Laszewski, Andrew Younge, Xi He, Marcel Kunze, Jie Tao et Cheng Fu. *Cloud Computing : a Perspective Study.* New Generation Computing, vol. 28, no. 2, pages 137–146, 2010. (Cité en page 10.)

Annexe A

Module comportement Maude

Listing A.1– Module Elastic_Cloud_Behaviour

```
mod Elastic_Cloud_Behaviour is

including Elastic_Cloud_FrontEnd .
including Elastic_Cloud_BackEnd .
including Elastic_Cloud_System .

vars nn n1 n2 n3 n4 : Nat .
vars f1 f2 f3 : Float .
var b : BackEnd .
var f : FrontEnd .
var eul : EndUserList .
var dvl : DeveloperList .
vars rl1 rl2 : RequestList .
vars r1 r2 r3 : Resources .
vars l : LoadBalancerList .
vars se : ServerList .
vars v1 v2 : VirtualMachineList .
vars c1 c2 : ContainerList .
vars s1 s2 : ServiceList .
var as : AllocServiceList .
var ds : DepServiceList .
var cs : ClientState .
var cc : ComponentState .
var cn1 cn2 cn3 : ComponentNature .

---Client/Cloud-Intercation---

crl [make-service-request] :
dvl | eul , EU n1 : req ( as ) | rl1 || b [ nn ]
=>
dvl | eul , EU n1 : req ( as ) | rl1 , RQ n1 || b [ sd(nn, 1) ]
if nn > 0 .

crl [send-request] :
dvl | eul | rl1 , RQ n1 || l , LB n2 : stable < r1 >( rl2 ){ cn1 } |
    se | v1 | s1 [ nn ]
=>
dvl | eul | rl1 || l , LB n2 : stable < r1 >( rl2 , RQ n1 ){ cn1 } |
    se | v1 | s1 [ sd(nn, 1) ]
if nn > 0 .

crl [proxy-request] :
```

```
f || l , LB n1 : stable < r1 >( rl1 , RQ n2 ){ cn1 } | se | v1 | s1 ,
     S n3 - n4 : stable < r2 >( rl2 ){ cn2 } [ nn ]
=>
f || l , LB n1 : stable < r1 >( rl1 ){ cn1 } | se | v1 | s1 , S n3 -
    n4 : stable < r2 >( rl2 , RQ n2 ){ cn2 } [ sd(nn, 1) ]
if nn > 0 .

crl [destroy-request] :
f || l | se | v1 | s1 , S n1 - n2 : stable < r2 >( rl2 , RQ n3  ){
    cn1 } [ nn ]
=>
f || l | se | v1 | s1 , S n1 - n2 : stable < r2 >( rl2 ){ cn1 } [ sd(
    nn, 1) ]
if nn > 0 .

crl [allocate-service] :
dvl | eul , EU n1 : req ( as ) | rl1 || l | se | v1 | s1 , S n2 - n3
    : stable < r2 >( rl2 , RQ n1 ){ cn1 } [ nn ]
=>
dvl | eul , EU n1 : req ( as , AS n2 ) | rl1 || l | se | v1 | s1 , S
    n2 - n3 : stable < r2 >( rl2 ){ cn1 } [ sd(nn, 1) ]
if nn > 0 .

crl [deallocate-service] :
dvl | eul , EU n1 : req ( as , AS n2 ) | rl1 || l | se | v1 | s1 , S
    n2 - n3 : stable < r2 >( rl2 ){ cn1 } [ nn ]
=>
dvl | eul , EU n1 : req ( as ) | rl1 || l | se | v1 | s1 , S n2 - n3
    : stable < r2 >( rl2 ){ cn1 } [ sd(nn, 1) ]
if nn > 0 .

crl [deploy-service-vm] :
dvl , D n1 : req ( ds , DS n2 ) | eul | rl1 || l | se | v1 , VM n3 -
    n4 : stable < r1 >{ cn1 } | s1 [ nn ]
=>
dvl , D n1 : req ( ds , DS n2 ) | eul | rl1 || l | se | v1 , VM n3 -
    n4 : stable < r1 >{ cn1 } | s1 ,
S n2 - n3 : loaded < M= 0.0 Gb , C= 0.0 Ghz , S= 0.0 Gb , B= 0.0 Mb
    >( ERQL ){ original } [ sd(nn, 1) ]
if nn > 0 .

crl [undeploy-service-vm] :
dvl , D n1 : req ( ds , DS n2 ) | eul | rl1 || l | se | v1 , VM n3 -
    n4 : stable < r1 >{ cn1 } | s1 ,
S n2 - n3 : cc < r2 >( rl2 ){ original } [ nn ]
=>
dvl , D n1 : req ( ds , DS n2 ) | eul | rl1 || l | se | v1 , VM n3 -
    n4 : stable < r1 >{ cn1 }| s1 [ sd(nn, 1) ]
if nn > 0 .

crl [deploy-service-ct] :
dvl , D n1 : req ( ds , DS n2 ) | eul | rl1 || l | se | c1 , CT n3 -
    n4 : stable < r1 >{ cn1 } | s1 [ nn ]
=>
```

```
dvl , D n1 : req ( ds , DS n2 ) | eul | rl1 || l | se | c1 , CT n3 -
    n4 : stable < r1 >{ cn1 } | s1 ,
S n2 - n3 : stable < M= 0.0 Gb , C= 0.0 Ghz , S= 0.0 Gb , B= 0.0 Mb
    >( ERQL ){ original } [ sd(nn, 1) ]
if nn > 0 .

crl [undeploy-service-ct] :
dvl , D n1 : req ( ds , DS n2 ) | eul | rl1 || l | se | c1 , CT n3 -
    n4 : stable < r1 >{ cn1 } | s1 ,
S n2 - n3 : cc < r2 >( rl2 ) { original } [ nn ]
=>
dvl , D n1 : req ( ds , DS n2 ) | eul | rl1 || l | se | c1 , CT n3 -
    n4 : stable < r1 >{ cn1 } | s1 [ sd(nn, 1) ]
if nn > 0 .

---Marking-EndUser/Developer---

crl [marking-idle-eu] :
dvl | eul , EU n1 : req ( as ) | rl1 || b [ nn ]
=>
dvl | eul , EU n1 : idle ( as ) | rl1 || b [ sd(nn, 1) ]
if nn > 0 .

crl [marking-req-eu] :
dvl | eul , EU n1 : idle ( as ) | rl1 || b [ nn ]
=>
dvl | eul , EU n1 : req ( as ) | rl1 || b [ sd(nn, 1) ]
if nn > 0 .

crl [marking-idle-dv] :
dvl , D n1 : req ( ds ) | eul | rl1 || b [ nn ]
=>
dvl , D n1 : idle ( ds ) | eul | rl1 || b [ sd(nn, 1) ]
if nn > 0 .

crl [marking-req-dv] :
dvl , D n1 : idle ( ds ) | eul | rl1 || b [ nn ]
=>
dvl , D n1 : req ( ds ) | eul | rl1 || b [ sd(nn, 1) ]
if nn > 0 .

---Horizontal-Scale---

crl [lb-instance-replication] :
f || l , LB n1 : loaded < r1 >( rl1 ){ original } | se | v1 | s1 [ nn
    ]
=>
f || l , LB n1 : loaded < r1 >( rl1 ){ original } , LB ( n1 * 10 + 1
    ) : stable < r1 >( ERQL ){ copy } | se | v1 | s1 [ sd(nn, 1) ]
if nn > 0 .

crl [lb-instance-to-be-destroyed] :
f || l , LB n1 : underused < r1 >( rl1 ){ copy } | se | v1 | s1 [ nn
    ]
```

```
=>
f || l , LB n1 : underused < r1 >( rl1 ){ destroyed } | se | v1 | s1
    [ sd(nn, 1) ]
if nn > 0 .

crl [lb-instance-consolidation] :
f || l , LB n1 : underused < r1 >( rl1 ){ destroyed } | se | v1 | s1
    [ nn ]
=>
f || l | se | v1 | s1 [ sd(nn, 1) ]
if nn > 0 .

crl [vm-instance-replication] :
f || l | se , SE n1 : stable < r1 > | v1 , VM n2 - n1 : loaded < r2
    >{ original } | s1 [ nn ]
=>
f || l | se , SE n1 : stable < r1 > | v1 , VM n2 - n1 : loaded < r2
    >{ original } ,
VM ( n2 * 10 + 1 ) - n1 : stable < r2 >{ copy } | s1 [ sd(nn, 1) ]
if nn > 0 .

crl [vm-instance-to-be-destroyed] :
f || l | se , SE n1 : stable < r1 > | v1 , VM n2 - n1 : underused <
    r2 >{ copy } | s1 [ nn ]
=>
f || l | se , SE n1 : stable < r1 > | v1 , VM n2 - n1 : underused <
    r2 >{ destroyed } | s1 [ sd(nn, 1) ]
if nn > 0 .

crl [vm-instance-consolidation] :
f || l | se , SE n1 : stable < r1 > | v1 , VM n2 - n1 : underused <
    r2 >{ destroyed } | s1 [ nn ]
=>
f || l | se , SE n1 : stable < r1 > | v1 | s1 [ sd(nn, 1) ]
if nn > 0 .

crl [ct-instance-replication] :
f || l | se , SE n1 : stable < r1 > | c1 , CT n2 - n1 : loaded < r2
    >{ original } | s1 [ nn ]
=>
f || l | se , SE n1 : stable < r1 > | c1 , CT n2 - n1 : loaded < r2
    >{ original } ,
CT ( n2 * 10 + 1 ) - n1 : stable < r2 >{ copy } | s1 [ sd(nn, 1) ]
if nn > 0 .

crl [ct-instance-to-be-destroyed] :
f || l | se , SE n1 : stable < r1 > | c1 , CT n2 - n1 : underused <
    r2 >{ copy } | s1 [ nn ]
=>
f || l | se , SE n1 : stable < r1 > | c1 , CT n2 - n1 : underused <
    r2 >{ destroyed } | s1 [ sd(nn, 1) ]
if nn > 0 .

crl [ct-instance-consolidation] :
```

```
f || l | se , SE n1 : stable < r1 > | c1 , CT n2 - n1 : underused <
    r2 >{ destroyed } | s1 [ nn ]
=>
f || l | se , SE n1 : stable < r1 > | c1 | s1 [ sd(nn, 1) ]
if nn > 0 .

crl [sevice-instance-replication] :
f || l | se | v1 , VM n1 - n2 : stable < r1 >{ cn1 } | s1 , S n3 - n1
     : loaded < r2 >( rl2 ){ original } [ nn ]
=>
f || l | se | v1 , VM n1 - n2 : stable < r1 >{ cn1 } | s1 ,
S n3 - n1 : loaded < r2 >( rl2 ){ original } , S ( n3 * 10 + 1 ) - n1
     : stable < r2 >( ERQL ){ copy } [ sd(nn, 1) ]
if nn > 0 .

crl [service-instance-to-be-destroyed] :
f || l | se | v1 , VM n1 - n2 : stable < r1 >{ cn1 } | s1 , S n3 - n1
     : underused < r2 >( rl2 ){ copy } [ nn ]
=>
f || l | se | v1 , VM n1 - n2 : stable < r1 >{ cn1 } | s1 , S n3 - n1
     : underused < r2 >( rl2 ){ destroyed } [ sd(nn, 1) ]
if nn > 0 .

crl [service-instance-consolidation] :
f || l | se | v1 , VM n1 - n2 : stable < r1 >{ cn1 } | s1 , S n3 - n1
     : underused < r2 >( rl2 ){ destroyed } [ nn ]
=>
f || l | se | v1 , VM n1 - n2 : stable < r1 >{ cn1 } | s1 [ sd(nn, 1)
     ]
if nn > 0 .

---Migration---

crl [vm-instance-migration] :
f || l | se , SE n1 : cc < r1 > , SE n2 : stable < r2 > | v1 , VM n3
    - n1 : loaded < r3 >{ cn1 } | s1 [ nn ]
=>
f || l | se , SE n1 : cc < r1 > , SE n2 : stable < r2 > | v1 , VM n3
    - n2 : loaded < r3 >{ migrated } | s1 [ sd(nn, 1) ]
if nn > 0 .

crl [ct-instance-migration] :
f || l | se , SE n1 : cc < r1 > , SE n2 : stable < r2 > | c1 , CT n3
    - n1 : loaded < r3 >{ cn1 } | s1 [ nn ]
=>
f || l | se , SE n1 : cc < r1 > , SE n2 : stable < r2 > | c1 , CT n3
    - n2 : loaded < r3 >{ migrated } | s1 [ sd(nn, 1) ]
if nn > 0 .

crl [service-instance-migration] :
f || l | se | v1 , VM n1 - n2 : cc < r1 >{ cn1 } , VM n3 - n2 :
    stable < r2 >{ cn2 } |
s1 , S n4 - n1 : loaded < r3 >( rl1 ){ cn3 } [ nn ]
=>
```

```
f || l | se | v1 , VM n1 - n2 : cc < r1 >{ cn1 } , VM n3 - n2 :
    stable < r2 >{ cn2 } |
s1 , S n4 - n3 : loaded < r3 >( rl1 ){ migrated } [ sd(nn, 1) ]
if nn > 0 .

---Vertical-Scale-Load-Balancer---

crl [increase-memory-lb] :
f || l , LB n1 : loaded < r1 , M= f1 Gb >( rl1 ){ cn1 } | se | v1 |
    s1 [ nn ]
=>
f || l , LB n1 : loaded < r1 , M= f1 + 0.5 Gb >( rl1 ){ increased } |
     se | v1 | s1 [ sd(nn, 1) ]
if f1 < 8.0 /\ nn > 0 .

crl [decrease-memory-lb] :
f || l , LB n1 : underused < r1 , M= f1 Gb >( rl1 ){ cn1 } | se | v1
    | s1 [ nn ]
=>
f || l , LB n1 : underused < r1 , M= f1 - 0.5 Gb >( rl1 ){ decreased
    } | se | v1 | s1 [ sd(nn, 1) ]
if f1 > 1.0 /\ nn > 0 .

crl [increase-cpu-lb] :
f || l , LB n1 : loaded < r1 , C= f1 Ghz >( rl1 ){ cn1 } | se | v1 |
    s1 [ nn ]
=>
f || l , LB n1 : loaded < r1 , C= f1 + 0.3 Ghz >( rl1 ){ increased }
    | se | v1 | s1 [ sd(nn, 1) ]
if f1 < 3.0 /\ nn > 0 .

crl [decrease-cpu-lb] :
f || l , LB n1 : underused < r1 , C= f1 Ghz >( rl1 ){ cn1 } | se | v1
     | s1 [ nn ]
=>
f || l , LB n1 : underused < r1 , C= f1 - 0.3 Ghz >( rl1 ){ decreased
     } | se | v1 | s1 [ sd(nn, 1) ]
if f1 > 0.6 /\ nn > 0 .

crl [increase-storage-lb] :
f || l , LB n1 : loaded < r1 , S= f1 Gb >( rl1 ){ cn1 } | se | v1 |
    s1 [ nn ]
=>
f || l , LB n1 : loaded < r1 , S= f1 + 10.0 Gb >( rl1 ){ increased }
    | se | v1 | s1 [ sd(nn, 1) ]
if f1 < 2000.0 /\ nn > 0 .

crl [decrease-storage-lb] :
f || l , LB n1 : underused < r1 , S= f1 Gb >( rl1 ){ cn1 } | se | v1
    | s1 [ nn ]
=>
f || l , LB n1 : underused < r1 , S= f1 - 10.0 Gb >( rl1 ){ decreased
     } | se | v1 | s1 [ sd(nn, 1) ]
if f1 > 10.0 /\ nn > 0 .
```

```
crl [increase-bandwidth-lb] :
f || l , LB n1 : loaded < r1 , B= f1 Mb >( rl1 ){ cn1 } | se | v1 |
    s1 [ nn ]
=>
f || l , LB n1 : loaded < r1 , B= f1 + 1.0 Mb >( rl1 ){ increased } |
     se | v1 | s1 [ sd(nn, 1) ]
if f1 < 30.0 /\ nn > 0 .

crl [decrease-bandwidth-lb] :
f || l , LB n1 : underused < r1 , B= f1 Mb >( rl1 ){ cn1 } | se | v1
    | s1 [ nn ]
=>
f || l , LB n1 : underused < r1 , B= f1 - 1.0 Mb >( rl1 ){ decreased
    } | se | v1 | s1 [ sd(nn, 1) ]
if f1 > 5.0 /\ nn > 0 .

---Vertical-Scale-Virtual-Machine---

crl [increase-memory-vm] :
f || l | se | v1 , VM n1 - n2 : loaded < r1 , M= f1 Gb >{ cn1 } | s1
    [ nn ]
=>
f || l | se | v1 , VM n1 - n2 : loaded < r1 , M= f1 + 0.5  Gb >{
    increased } | s1 [ sd(nn, 1) ]
if f1 < 8.0 /\ nn > 0 .

crl [decrease-memory-vm] :
f || l | se | v1 , VM n1 - n2 : underused < r1 , M= f1 Gb >{ cn1 } |
    s1 [ nn ]
=>
f || l | se | v1 , VM n1 - n2 : underused < r1 , M= f1 - 0.5 Gb >{
    decreased } | s1 [ sd(nn, 1) ]
if f1 > 1.0 /\ nn > 0 .

crl [increase-cpu-vm] :
f || l | se | v1 , VM n1 - n2 : loaded < r1 , C= f1 Ghz >{ cn1 } | s1
     [ nn ]
=>
f || l | se | v1 , VM n1 - n2 : loaded < r1 , C= f1 + 0.3 Ghz >{
    increased } | s1 [ sd(nn, 1) ]
if f1 < 3.0 /\ nn > 0 .

crl [decrease-cpu-vm] :
f || l | se | v1 , VM n1 - n2 : underused < r1 , C= f1 Ghz >{ cn1 } |
     s1 [ nn ]
=>
f || l | se | v1 , VM n1 - n2 : underused < r1 , C= f1 - 0.3 Ghz >{
    decreased } | s1 [ sd(nn, 1) ]
if f1 > 0.6 /\ nn > 0 .

crl [increase-storage-vm] :
f || l | se | v1 , VM n1 - n2 : loaded < r1 , S= f1 Gb >{ cn1 } | s1
    [ nn ]
```

```
=>
f || l | se | v1 , VM n1 - n2 : loaded < r1 , S= f1 + 10.0  Gb >{
    increased } | s1 [ sd(nn, 1) ]
if f1 < 2000.0 /\ nn > 0 .

crl [decrease-storage-vm] :
f || l | se | v1 , VM n1 - n2 : underused < r1 , S= f1 Gb >{ cn1 } |
    s1 [ nn ]
=>
f || l | se | v1 , VM n1 - n2 : underused < r1 , S= f1 - 10.0 Gb >{
    decreased } | s1 [ sd(nn, 1) ]
if f1 > 10.0 /\ nn > 0 .

crl [increase-bandwidth-vm] :
f || l | se | v1 , VM n1 - n2 : loaded < r1 , B= f1 Mb >{ cn1 } | s1
    [ nn ]
=>
f || l | se | v1 , VM n1 - n2 : loaded < r1 , B= f1 + 1.0 Mb >{
    increased } | s1 [ sd(nn, 1) ]
if f1 < 30.0 /\ nn > 0 .

crl [decrease-bandwidth-vm] :
f || l | se | v1 , VM n1 - n2 : underused < r1 , B= f1 Mb >{ cn1 } |
    s1 [ nn ]
=>
f || l | se | v1 , VM n1 - n2 : underused < r1 , B= f1 - 1.0 Mb >{
    decreased } | s1 [ sd(nn, 1) ]
if f1 > 5.0 /\ nn > 0 .

---Vertical-Scale-Container---

crl [increase-memory-ct] :
f || l | se | c1 , CT n1 - n2 : loaded < r1 , M= f1 Gb >{ cn1 } | s1
    [ nn ]
=>
f || l | se | c1 , CT n1 - n2 : loaded < r1 , M= f1 + 0.5  Gb >{
    increased } | s1 [ sd(nn, 1) ]
if f1 < 8.0 /\ nn > 0 .

crl [decrease-memory-ct] :
f || l | se | c1 , CT n1 - n2 : underused < r1 , M= f1 Gb >{ cn1 } |
    s1 [ nn ]
=>
f || l | se | c1 , CT n1 - n2 : underused < r1 , M= f1 - 0.5 Gb >{
    decreased } | s1 [ sd(nn, 1) ]
if f1 > 1.0 /\ nn > 0 .

crl [increase-cpu-ct] :
f || l | se | c1 , CT n1 - n2 : loaded < r1 , C= f1 Ghz >{ cn1 } | s1
     [ nn ]
=>
f || l | se | c1 , CT n1 - n2 : loaded < r1 , C= f1 + 0.3 Ghz >{
    increased } | s1 [ sd(nn, 1) ]
if f1 < 3.0 /\ nn > 0 .
```

```
crl [decrease-cpu-ct] :
f || l | se | c1 , CT n1 - n2 : underused < r1 , C= f1 Ghz >{ cn1 } |
     s1 [ nn ]
=>
f || l | se | c1 , CT n1 - n2 : underused < r1 , C= f1 - 0.3 Ghz >{
    decreased } | s1 [ sd(nn, 1) ]
if f1 > 0.6 /\ nn > 0 .

crl [increase-storage-ct] :
f || l | se | c1 , CT n1 - n2 : loaded < r1 , S= f1 Gb >{ cn1 } | s1
    [ nn ]
=>
f || l | se | c1 , CT n1 - n2 : loaded < r1 , S= f1 + 10.0  Gb >{
    increased } | s1 [ sd(nn, 1) ]
if f1 < 2000.0 /\ nn > 0 .

crl [decrease-storage-ct] :
f || l | se | c1 , CT n1 - n2 : underused < r1 , S= f1 Gb >{ cn1 } |
    s1 [ nn ]
=>
f || l | se | c1 , CT n1 - n2 : underused < r1 , S= f1 - 10.0 Gb >{
    decreased } | s1 [ sd(nn, 1) ]
if f1 > 10.0 /\ nn > 0 .

crl [increase-bandwidth-ct] :
f || l | se | c1 , CT n1 - n2 : loaded < r1 , B= f1 Mb >{ cn1 } | s1
    [ nn ]
=>
f || l | se | c1 , CT n1 - n2 : loaded < r1 , B= f1 + 1.0 Mb >{
    increased } | s1 [ sd(nn, 1) ]
if f1 < 30.0 /\ nn > 0 .

crl [decrease-bandwidth-ct] :
f || l | se | c1 , CT n1 - n2 : underused < r1 , B= f1 Mb >{ cn1 } |
    s1 [ nn ]
=>
f || l | se | c1 , CT n1 - n2 : underused < r1 , B= f1 - 1.0 Mb >{
    decreased } | s1 [ sd(nn, 1) ]
if f1 > 5.0 /\ nn > 0 .

---Vertical-Scale-Service---

crl [increase-memory-service] :
f || l | se | v1 | s1 , S n1 - n2 : loaded < r1 , M= f1 Gb  >( rl1 ){
     cn1 } [ nn ]
=>
f || l | se | v1 | s1 , S n1 - n2 : loaded < r1 , M= f1 + 0.5  Gb  >(
     rl1 ){ increased } [ sd(nn, 1) ]
if f1 < 3.0 /\ nn > 0 .

crl [decrease-memory-service] :
f || l | se | v1 | s1 , S n1 - n2 : underused < r1 , M= f1 Gb >( rl1
    ){ cn1 } [ nn ]
```

```
=>
f || l | se | v1 | s1 , S n1 - n2 : underused < r1 , M= f1 - 0.5 Gb
    >( rl1 ){ decreased } [ sd(nn, 1) ]
if f1 > 0.5 /\ nn > 0 .

crl [increase-cpu-service] :
f || l | se | v1 | s1 , S n1 - n2 : loaded < r1 , C= f1 Ghz >( rl1 ){
     cn1 } [ nn ]
=>
f || l | se | v1 | s1 , S n1 - n2 : loaded < r1 , C= f1 + 0.3 Ghz >(
    rl1 ){ increased } [ sd(nn, 1) ]
if f1 < 2.0 /\ nn > 0 .

crl [decrease-cpu-service] :
f || l | se | v1 | s1 , S n1 - n2 : underused < r1 , C= f1 Ghz >( rl1
     ){ cn1 } [ nn ]
=>
f || l | se | v1 | s1 , S n1 - n2 : underused < r1 , C= f1 - 0.3 Ghz
    >( rl1 ){ decreased } [ sd(nn, 1) ]
if f1 > 0.3 /\ nn > 0 .

crl [increase-storage-service] :
f || l | se | v1 | s1 , S n1 - n2 : loaded < r1 , S= f1 Gb >( rl1 ){
    cn1 } [ nn ]
=>
f || l | se | v1 | s1 , S n1 - n2 : loaded < r1 , S= f1 + 10.0  Gb >(
     rl1 ){ increased } [ sd(nn, 1) ]
if f1 < 500.0 /\ nn > 0 .

crl [decrease-storage-service] :
f || l | se | v1 | s1 , S n1 - n2 : underused < r1 , S= f1 Gb >( rl1
    ){ cn1 } [ nn ]
=>
f || l | se | v1 | s1 , S n1 - n2 : underused < r1 , S= f1 - 10.0  Gb
     >( rl1 ){ decreased } [ sd(nn, 1) ]
if f1 > 5.0 /\ nn > 0 .

crl [increase-bandwidth-service] :
f || l | se | v1 | s1 , S n1 - n2 : loaded < r1 , B= f1 Mb >( rl1 ){
    cn1 } [ nn ]
=>
f || l | se | v1 | s1 , S n1 - n2 : loaded < r1 , B= f1 + 1.0 Mb >(
    rl1 ){ increased } [ sd(nn, 1) ]
if f1 < 5.0 /\ nn > 0 .

crl [decrease-bandwidth-service] :
f || l | se | v1 | s1 , S n1 - n2 : underused < r1 , B= f1 Mb >( rl1
    ){ cn1 } [ nn ]
=>
f || l | se | v1 | s1 , S n1 - n2 : underused < r1 , B= f1 - 1.0 Mb
    >( rl1 ){ decreased } [ sd(nn, 1) ]
if f1 > 1.0 /\ nn > 0 .

---Marking-Unmarking-End-User/Developer---
```

```
crl [marking-lb-loaded] :
f || l , LB n1 : stable < r1 >( rl1 ){ cn1 } | se | v1 | s1 [ nn ]
=>
f || l , LB n1 : loaded < r1 >( rl1 ){ cn1 } | se | v1 | s1 [ sd(nn,
    1) ]
if nn > 0 .

crl [unmarking-lb-loaded] :
f || l , LB n1 : loaded < r1 >( rl1 ){ cn1 } | se | v1 | s1 [ nn ]
=>
f || l , LB n1 : stable < r1 >( rl1 ){ cn1 } | se | v1 | s1 [ sd(nn,
    1) ]
if nn > 0 .

crl [marking-lb-underused] :
f || l , LB n1 : stable < r1 >( rl1 ){ cn1 } | se | v1 | s1 [ nn ]
=>
f || l , LB n1 : underused < r1 >( rl1 ){ cn1 } | se | v1 | s1 [ sd(
    nn, 1) ]
if nn > 0 .

crl [unmarking-lb-underused] :
f || l , LB n1 : underused < r1 >( rl1 ){ cn1 } | se | v1 | s1 [ nn ]
=>
f || l , LB n1 : stable < r1 >( rl1 ){ cn1 } | se | v1 | s1 [ sd(nn,
    1) ]
if nn > 0 .

crl [marking-vm-loaded] :
f || l | se | v1 , VM n1 - n2 : stable < r1 >{ cn1 } | s1 [ nn ]
=>
f || l | se | v1 , VM n1 - n2 : loaded < r1 >{ cn1 } | s1 [ sd(nn, 1)
    ]
if nn > 0 .

crl [unmarking-vm-loaded] :
f || l | se | v1 , VM n1 - n2 : loaded < r1 >{ cn1 } | s1 [ nn ]
=>
f || l | se | v1 , VM n1 - n2 : stable < r1 >{ cn1 } | s1 [ sd(nn, 1)
    ]
if nn > 0 .

crl [marking-vm-underused] :
f || l | se | v1 , VM n1 - n2 : stable < r1 >{ cn1 } | s1 [ nn ]
=>
f || l | se | v1 , VM n1 - n2 : underused < r1 >{ cn1 } | s1 [ sd(nn,
    1) ]
if nn > 0 .

crl [unmarking-vm-underused] :
f || l | se | v1 , VM n1 - n2 : underused < r1 >{ cn1 } | s1 [ nn ]
=>
f || l | se | v1 , VM n1 - n2 : stable < r1 >{ cn1 } | s1 [ sd(nn, 1)
```

```
    ]
if nn > 0 .

crl [marking-ct-loaded] :
f || l | se | c1 , CT n1 - n2 : stable < r1 >{ cn1 } | s1 [ nn ]
=>
f || l | se | c1 , CT n1 - n2 : loaded < r1 >{ cn1 } | s1 [ sd(nn, 1)
    ]
if nn > 0 .

crl [unmarking-ct-loaded] :
f || l | se | c1 , CT n1 - n2 : loaded < r1 >{ cn1 } | s1 [ nn ]
=>
f || l | se | c1 , CT n1 - n2 : stable < r1 >{ cn1 } | s1 [ sd(nn, 1)
    ]
if nn > 0 .

crl [marking-ct-underused] :
f || l | se | c1 , CT n1 - n2 : stable < r1 >{ cn1 }| s1 [ nn ]
=>
f || l | se | c1 , CT n1 - n2 : underused < r1 >{ cn1 } | s1 [ sd(nn,
    1) ]
if nn > 0 .

crl [unmarking-ct-underused] :
f || l | se | c1 , CT n1 - n2 : underused < r1 >{ cn1 } | s1 [ nn ]
=>
f || l | se | c1 , CT n1 - n2 : stable < r1 >{ cn1 } | s1 [ sd(nn, 1)
    ]
if nn > 0 .

crl [marking-service-loaded] :
f || l | se | v1 | s1 , S n1 - n2 : stable < r1 >( rl1 ){ cn1 } [ nn
    ]
=>
f || l | se | v1 | s1 , S n1 - n2 : loaded < r1 >( rl1 ){ cn1 } [ sd(
    nn, 1) ]
if nn > 0 .

crl [unmarking-service-loaded] :
f || l | se | v1 | s1 , S n1 - n2 : loaded < r1 >( rl1 ){ cn1 } [ nn
    ]
=>
f || l | se | v1 | s1 , S n1 - n2 : stable < r1 >( rl1 ){ cn1 } [ sd(
    nn, 1) ]
if nn > 0 .

crl [marking-service-underused] :
f || l | se | v1 | s1 , S n1 - n2 : stable < r1 >( rl1 ){ cn1 } [ nn
    ]
=>
f || l | se | v1 | s1 , S n1 - n2 : underused < r1 >( rl1 ){ cn1 } [
    sd(nn, 1) ]
if nn > 0 .
```

```
crl [unmarking-service-underused] :
f || l | se | v1 | s1 , S n1 - n2 : underused < r1 >( rl1 ){ cn1 } [
    nn ]
=>
f || l | se | v1 | s1 , S n1 - n2 : stable < r1 >( rl1 ){ cn1 } [ sd(
    nn, 1) ]
if nn > 0 .

endm
```