Modélisation Formelle des Systèmes Fog : vers l'Analyse et la Validation de leur Comportement

Souad Marir

Université Constantine 2 Abdelhamid Mehri - Faculté des Nouvelles Technologies de l'Information et de la Communication - Département des Technologies Logicielles et Systèmes d'Information Université de Pau et des Pays de l'Adour - Ecole doctorale des Sciences Exactes et leurs Applications (ED SEA 211)

Modélisation Formelle des Systèmes Fog : vers l'Analyse et la Validation de leur Comportement

THÈSE

pour l'obtention du titre de

Docteur en informatique

préparée dans le cadre d'une cotutelle entre

Université de Constantine 2 - Abdelhamid Mehri

et

Université de Pau et des Pays de l'Adour

Présentée par

MARIR Souad

Composition du jury

Président :	Pr. Mahmoud BOUFAIDA	Université Constantine 2 – Abdelhamid Mehri, Algérie
Rapporteurs :	Pr. Mustapha ABDI	Université d'Oran 1 – Ahmed Ben Bella, Algérie
	Pr. Abdelkrim ABDELLI	USTHB, Algérie
Examinateur :	Dr. Messaoud MAAROUK	Université de Khenchela – Abbes Leghrour, Algérie
Directeurs :	Pr. Faiza BELALA	Université Constantine 2 – Abdelhamid Mehri, Algérie
	Dr. Nabil HAMEURLAIN	Université de Pau et des Pays de l'Adour, France

Laboratoire d'Informatique Répartie (LIRE)
Laboratoire d'Informatique de l'Université de Pau et des Pays de l'Adour (LIUPPA EA 3000)

Remerciements

Ce travail a été effectué entre le **L**aboratoire d'**I**nformatique **RE**partie de l'université de Constantine 2 Abdelhamid Mehri, à la faculté des NTIC (nouvelles technologies de l'information et de la communication), au département de TLSI (technologies des logiciels et des systèmes d'information) et le **L**aboratoire d'**I**nformatique de l'**U**niversité de **P**au et des **P**ays de l'**A**dour que je remercie.

Je tiens à témoigner mes remerciements et toute ma reconnaissance à Madame la Professeur Belala Faiza à l'université de Constantine 2 qui a Dirigé ce travail, pour m'avoir soutenu jusqu'à l'aboutissement de mes travaux. Je la remercie vivement pour son aide efficace, son expérience et son entière disponibilité.

Je remercie également monsieur Nabil Hameurlain, maître de conférences habilité à diriger des recherches à l'université de Pau et des Pays de l'Adour (UPPA) qui a co-dirigé cette thèse, pour m'avoir accueilli au sein du LIUPPA. Sa rigueur et ses précieux conseils m'ont permit d'atteindre mes objectifs.

Je tiens à remercier très sincèrement monsieur le Professeur Chikhi Salim, qui m'a accueilli au sein de son laboratoire de **M**odélisation et d'**I**mplémentation des **S**ystèmes **C**omplexes de l'université de Constantine 2. Pour m'avoir traitée comme un membre à part entière de son laboratoire, et de poursuivre le plus dur de mes travaux, ma reconnaissance envers lui est éternelle.

Je tiens à remercier monsieur Mustapha Abdi, professeur à l'université d'Oran 1 − Ahmed Ben Bella , et monsieur Abdelkrim Abdelli, professeur à l'université des Sciences et de la Technologie Houari Boumediene (USTHB), pour l'honneur qu'ils m'ont fait en acceptant d'être rapporteurs de cette thèse, et pour le temps qu'ils ont consacré à mon travail de recherche.

Je tiens à remercier monsieur Mahmoud Boufaida, professeur à l'université Constantine 2 − Abdelhamid Mehri, pour avoir accepté de présider mon jury de thèse, ainsi que monsieur Messaoud Maarouk, maître de conférence à l'université de Khenchela − Abbes Leghrour, pour avoir accepté d'être dans mon jury de thèse en tant qu'examinateur. Je les remercie pour l'honneur qu'ils me font, pour leur participation scientifique ainsi que pour le temps qu'ils ont consacré à ma recherche.

Je remercie également tous les membres des laboratoires LIRE et LIUPPA.

Je tiens à remercier toute personne qui a contribué, de près ou de loin, à ce modeste travail.

Dédicaces

Je dédie ce modeste travail,

à Ma mère, mon pilier dans la vie qui m'a soutenue, aidée et épaulée tout le long de mes dures épreuves, je n'y serai jamais arrivée sans elle ;

à Ma sœur, Amel, mon amie, qui me protège et m'est d'une grande aide depuis le début ;

à Mon fiancé, Amine, qui m'encourage et qui croit en moi depuis le premier jour ;

à Mes partenaires Bouchera, Hanene, Khadidja, Roumeissa et Yasmina qui ont été là pour moi dans les instants les plus difficiles ;

à Mes amis Elise, Eva et Ivan qui ont égayé mon séjour à Pau et qui ont toujours été à mon écoute ;

à Toutes les personnes qui m'ont aidé à atteindre mon rêve, tous mes proches qui me sont chers, à tout les miens.

Pour finir, et surtout, à la mémoire de mon père, mon confident, avec lequel j'aurai aimé partagé ma joie, puisse-t-il être heureux de cet accomplissement.

Résumé

Le Fog computing est un nouveau paradigme émergent présentant un nombre important de défis. Basés sur les systèmes IoT (Internet of Things), les systèmes Fog sont définis afin de rapprocher les traitements et le stockage des données des dispositifs IoT. De ce fait, le temps de latence diminue et l'élaboration de systèmes critiques à l'aide de ce paradigme se trouve être un choix judicieux. Cependant, développer un système aussi complexe se révèle être une tâche particulièrement difficile. Par conséquent, la conception d'un système Fog est cruciale, notamment du point de vue de l'hétérogénéité, de la dispersion géographique, de la réactivité, de la portabilité des services et de l'interopérabilité entre ses différents éléments.

Dans le cadre de cette thèse, nous proposons un langage de modélisation spécifique au Fog computing noté Fog-DSML afin de relever les différents défis des systèmes Fog. Ce langage englobe 1) une syntaxe abstraite composée de diagrammes représentant les entités d'un système Fog, 2) une syntaxe concrète représentée par une architecture multi-couches des systèmes Fog fournissant une description graphique et conceptuelle d'un système Fog et 3) une sémantique formelle à base des BiAgents* (*Bigraphical Agents*) qui est une extension des systèmes bigraphiques (BRS) par des agents intelligents pour fournir une description rigoureuse des aspects physiques, virtuels et comportementaux des systèmes Fog.

L'approche proposée est validée par une étude de cas : système d'inspection des bagages LIS (*Luggage Inspection System*). Nous procédons aussi à l'encodage des différents concepts BiAgents* en langage de stratégie Maude et nous les implémentons à travers différents scénarios d'exécution. De plus, nous vérifions formellement les propriétés pertinentes des systèmes Fog à savoir la portabilité des données et leur interopérabilité. L'intégration des stratégies Maude dans la réécriture des états du système Fog a permis de guider l'exécution du modèle et son analyse.

Mots-clés : Fog Computing, Modélisation formelle, BiAgents*, Vérification formelle, Maude stratégie, Fog-DSML, DSML.

Abstract

Fog computing is a new emerging paradigm with a variety of significant challenges. Based on IoT (Internet of Things) systems, Fog systems are defined to bring data processing and storage closer to IoT devices. As a result, latency is reduced and the development of critical systems using this paradigm is a wise choice. However, developing such a complex system turns out to be a particularly challenging task. Therefore, the design of a Fog system is crucial, especially from the point of view of heterogeneity, geographical dispersion, reactivity, service portability and interoperability between its different components.

In this thesis, we propose a modelling language specific to Fog computing noted Fog-DSML in order to address the various challenges of Fog systems. This language includes 1) an abstract syntax consisting in a class diagram representing the entities of a Fog system, 2) a concrete syntax represented by a multi-layer architecture of Fog systems providing a graphical and conceptual description of a Fog system and 3) a formal semantics based on BiAgents* (*Bigraphical Agents*) which is an extension of bigraphic systems (BRS) by intelligent agents to provide a rigorous description of the physical, virtual and behavioural aspects of Fog systems.

The proposed approach is validated through a case study : LIS (Luggage Inspection System). We also proceed to the encoding of the different BiAgents* concepts in Maude strategy language and implement them through different execution scenarios. Furthermore, we formally verify the relevant properties of Fog systems, which are data portability and interoperability. The integration of Maude strategies in the rewriting of the Fog system states aime to guide the model execution and its analysis.

Mots-clés : Fog Computing, Formal modelling, BiAgents*, Formal verification, Maude strategy, Fog-DSML, DSML.

ملخص

تعد الحوسبة الضبابية نموذجًا جديدًا ناشئًا يقدم عددًا كبيرًا من التحديات. استنادًا إلى أنظمة IoT (إنترنت الأشياء)، يتم تعريف أنظمة الضباب لجعل معالجة البيانات وتخزينها أقرب إلى أجهزة إنترنت الأشياء. نتيجة لذلك، يتناقص زمن الوصول وبناء أنظمة مهمة باستخدام هذا النموذج يعد اختيارًا حكيمًا. ومع ذلك، فإن تطوير مثل هذا النظام المعقد هو مهمة صعبة بشكل خاص. لذلك، يعد تصميم نظام الضباب أمرًا بالغ الأهمية، لا سيما من وجهة نظر عدم التجانس والتشتت الجغرافي والاستجابة وإمكانية نقل الخدمات وإمكانية التشغيل البيني بين عناصره المختلفة.

نقترح من خلال هذه الأطروحة لغة نمذجة خاصة بحوسبة الضباب برمز Fog-DSML من أجل مواجهة التحديات المختلفة لأنظمة الضباب. تتضمن هذه اللغة 1) بنية مجردة تتكون من الرسوم البيانية التي تمثل كيانات نظام الضباب بالإضافة إلى سلوكها العام، 2) بناء جملة ملموس يمثله بنية متعددة الطبقات لأنظمة الضباب توفر وصفًا رسميا ومفاهيميًا لنظام الضباب و3) دلالات رسمية تعتمد على (BiAgents * Bigraphical Agents) وهي امتداد للأنظمة ثنائية البيان (BRS) بواسطة مفوضين أذكياء لتوفير وصف دقيق للجوانب المادية والافتراضية والسلوكية لأنظمة الضباب.

يتم التحقق من صحة النهج المقترح من خلال دراسة حالة: LIS (نظام فحص الأمتعة). ننتقل أيضًا إلى ترميز رموز BiAgents * المختلفة بلغة استراتيجية Maude وتنفيذ مفاهيمهم من خلال سيناريوهات التنفيذ المختلفة. بالإضافة إلى ذلك، نتحقق رياضيا من صحة الخصائص ذات الصلة لأنظمة الضباب ، أي قابلية نقل البيانات وقابلية التشغيل البيني. ساعد دمج استراتيجيات Maude في إعادة كتابة حالات نظام الضباب في توجيه تنفيذ النموذج وتحليله.

الكلمات الرئيسية: الحوسبة الضبابية، النمذجة الرسمية، *BiAgents، التحقق الرياضي، Maude Strategy, Fog-DSML, DSML

"Liveness property : Something good will happen."

Bowen Alpern and Fred Shneider

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction générale

Sommaire

1.1 Contexte

À l'ère des technologies de l'information, les données sont la principale marchandise, et le fait de posséder plus de données génère généralement plus de valeur dans les entreprises axées sur les données. Selon l'International Data Corporation (IDC), la quantité de données numériques générées a dépassé 1 zettaoctet en 2010 [Gantz and Reinsel, 2011]. En outre, 2,5 exaoctets de nouvelles données sont générés chaque jour depuis 2012 [Mcafee and Brynjolfsson, 2012]. Ces appareils connectés constituent l'Internet des objets (IoT) et sont susceptibles de générer une quantité massive de données. Avec cette quantité astronomique de données, les architectures de réseau mobile actuelles auront du mal à gérer l'élan et l'ampleur de ces données. Dans les mises en œuvre actuelles des applications basées sur le Cloud, la plupart des données qui nécessitent un stockage, une analyse et une prise de décision sont envoyées vers les centres de données du Cloud [Ravandi and Papapanagiotou, 2017].

À mesure que la vitesse et le volume des données augmentent, le transfert des "big data" des dispositifs IoT vers le Cloud pourrait ne pas être efficace, voir impossible dans certains cas en raison des contraintes de bande passante. D'autre part, avec l'émergence des applications sensibles au temps et à l'emplacement (telles que la surveillance des patients, la fabrication en temps réel, les voitures à conduite autonome, les flottes de drones ou l'assistance cognitive), le Cloud distant ne sera pas en mesure de répondre aux exigences de latence ultra-faible de ces applications, de fournir des services sensibles à l'emplacement ou de s'adapter à l'ampleur des données produites par ces applications [Zhang et al., 2015]. En outre, dans certaines applications, l'envoi des données dans le Cloud peut ne pas être une solution réalisable en raison de problèmes de confidentialité. Afin de résoudre les problèmes liés aux applications à large bande passante, géographiquement dispersées, à très faible latence et sensibles à la confidentialité, il est absolument nécessaire d'adopter un paradigme informatique qui se rapproche des dispositifs connectés.
Le Fog computing a été proposé par l'industrie et le monde universitaire [IEEE, 2018], [Bonomi et al., 2012] pour répondre aux problèmes susmentionnés et satisfaire le besoin d'un paradigme informatique plus proche des dispositifs connectés. Le Fog computing comble le fossé entre le Cloud et les dispositifs IoT en permettant le calcul, le stockage, la mise en réseau et la gestion des données sur les nœuds de réseau à proximité immédiate des dispositifs IoT. Par conséquent, toutes ces opérations se produisent le long du chemin entre les dispositifs IoT et le Cloud, alors que les données sont transférées vers le Cloud à partir des dispositifs IoT. D'autres paradigmes informatiques similaires au Fog computing, tels que l'Edge computing, le "Cloud of Things" et les "Cloudlets", ont été proposés par la communauté des chercheurs pour résoudre les problèmes mentionnés.
Le Fog computing est un paradigme représentant l'intermédiaire entre le Cloud et les objets connectés, permettant des exécutions en bordure du réseau. Ainsi, le réseau de calcul, la prise de décision, le stockage et la gestion des données se font entre les objets et le Cloud. OpenFog Consortium [IEEE, 2018] définit le Fog computing comme étant des plateformes horizontales et verticales. La plateforme horizontale permet de disperser les opérations informatiques entre les plateformes applicatives et les secteurs et la verticale encourage les applications isolées.
Ce nouveau domaine de l'informatique a donné lieu à d'intenses recherches et études au cours de la dernière décennie. De nombreux chercheurs travaillent à sa normalisation puisque l'OpenFog consortum a établi la norme IEEE Std 1934-2018 pour une définition d'une architecture Fog et ses piliers. Ils se sont rencontrés sur ses principales caractéristiques [Ai et al., 2018, Yi et al., 2015] :

— Faible latence et conscience de l'emplacement : Le Fog computing permet de connaître

l'emplacement en permettant le déploiement de nœuds Fog à divers endroits. En outre, en raison de la proximité du Fog avec les dispositifs terminaux, il a une latence réduite lors du traitement des données.

— Dispersion géographique : Contrairement au Cloud centralisé, le Fog fournit des services dispersés et des applications qui peuvent être installées n'importe où.

— Scalabilité : Le Fog fournit des ressources de calcul et de stockage distribuées qui peuvent gérer des périphériques terminaux massifs.

— Mobilité : L'une des caractéristiques les plus importantes des systèmes Fog est leur capacité à se connecter directement aux appareils mobiles, offrant les technologies de mobilité.

— Interactions : Au lieu du traitement par paquets utilisé dans le Cloud, les systèmes Fog permettent des interactions en temps réel entre les nœuds Fog.

— Hétérogénéité : Différents fabricants créent des nœuds Fog ou des dispositifs terminaux, qui viennent dans une variété de types et doivent être installés selon leurs plateformes.

— Interopérabilité : Les composants Fog peuvent interagir entre eux et s'exécuter dans différents domaines et fournisseurs de services.

— Portabilité : Les données traitées peuvent se déplacer à travers divers systèmes et sous-systèmes d'exécution sans avoir à les réécrire entièrement ou partiellement

1.2 Problématique

Les systèmes basés sur le Fog computing prennent en charge une variété d'applications de l'IoT, y compris les transports, l'agriculture, les villes intelligentes, les bâtiments intelligents, les soins de santé, l'hôtellerie et les services financiers, pour n'en citer que quelques-uns. En outre, ce paradigme fournit des méthodes efficaces pour surmonter de nombreuses limitations des architectures informatiques existantes liées aux systèmes Cloud et IoT.
Il existe plusieurs efforts de modélisation et de compréhension du comportement dynamique et imprévisible des systèmes Fog. Nous pouvons en distinguer deux types [Mouradian et al., 2017] : les approches basées architectures et celles basées algorithmes. En effet, certains travaux tels que [Asadi et al., 2021], [Xiao and Krunz, 2021] et [Roig et al., 2021c] se concentrent sur les aspects physiques des systèmes Fog en utilisant des architectures, une modélisation graphique ou une modélisation formelle. D'autres se concentrent sur l'étude comportementale des systèmes Fog, tel que [Abd Elaziz et al., 2021], avec des algorithmes.
Les solutions existantes ne décrivent pas globalement les systèmes Fog avec leurs différentes caractéristiques et exigences. En effet, certaines se focalisent sur l'aspect structurel d'un système Fog et d'autres sur l'aspect comportemental. Certaines se concentrent sur les dispositifs alors que d'autres se focalisent sur l'aspect réseau ou encore sur l'aspect sémantique selon les visions des systèmes IoT. Certaines approches aussi cherchent à garantir le coté sécurité des systèmes Fog et d'autres sur l'interopérabilité des services et des données partagées entre les différentes couches de ces systèmes. Certaines se focalisent sur la garantie d'un temps de réponse minimum et d'autres sur la décongestion de la bande passante.
La gestion et la coordination des dispositifs IoT hétérogènes et géographiquement dispersés, ainsi que la sélection de ressources appropriées, sont extrêmement complexes. De plus, les dispositifs IoT ont la capacité d'évoluer et de modifier dynamiquement leur flux de travail. Les propriétés internes et les performances des dispositifs IoT sont donc affectées.
De tels systèmes complexes nécessitent une spécification conceptuelle utilisant les techniques d'ingénierie logicielle appropriées. Cela leur offrira un niveau élevé d'abstraction pour relever tous les défis mentionnés. En effet, le comportement global d'un système Fog est déterminé par un certain nombre de facteurs, notamment l'emplacement des divers dispositifs du système, leur

type, la logique régissant le comportement des divers composants du système, et les diverses politiques et propriétés de haut niveau qui doivent être satisfaites. Les systèmes Fog sont difficiles à concevoir en raison de la complexité de ces dépendances, ainsi que de la gestion des effets secondaires potentiellement négatifs sur l'état général du système.
Les systèmes Fog deviennent de plus en plus complexes, ce qui rend leur conception difficile à appréhender. Les approches de développement traditionnelles, bien que coûteuses, ne parviennent pas toujours à garantir la validité du comportement intelligent d'un système Fog, ainsi que sa robustesse et fiabilité.

1.3 Objectifs et contributions

Il est essentiel de s'appuyer sur un modèle pour décrire la conception, la structure et le comportement des systèmes Fog. La modélisation est un effort essentiel pour déterminer et comprendre les changements structurels et les relations comportementales dans un système Fog afin d'éviter la formation de comportements indésirables tels que l'exécution non coordonnée de processus et une réduction de la qualité du service. Le Génie Logiciel peut répondre à ces exigences de par les méthodes formelles et l'ingénierie dirigée par les modèles MDE ("*Model Driven Engineering*").

Les approches formelles, qui sont définies par leur efficacité, leur fiabilité et leur précision, apportent une solution appropriée aux nombreux obstacles et à l'élimination des ambiguïtés sémantiques dans ce contexte. Cela permet un raisonnement rigoureux sur les spécifications formelles dérivées afin d'établir leur validité en utilisant une variété d'approches de simulation et de vérification formelles. Le MDE intègre différentes méthodes formelles et semi formelles afin de concevoir un système aussi complexe tel que les systèmes Fog en respectant ses différentes caractéristiques. De plus, afin de garantir l'efficacité du comportement intelligent défini, il est primordial de l'évaluer et de le valider, avant de l'utiliser dans des environnements Fog réels. Dans le contexte de l'ingénierie des systèmes Fog, les principaux défis que nous relevons dans ce travail de thèse sont les suivants :

D1- Définir une approche MDE à utiliser afin d'abstraire ce type complexe de systèmes.

D2- Choisir le formalisme adéquat afin de spécifier les aspects structurels et comportementaux des systèmes Fog avec leurs caractéristiques.

D3- Traiter les aspects principaux d'interopérabilité et de portabilité des données dans un système Fog.

D4- Évaluer les performances liés à ces aspects à travers une étude de cas.

L'objectif général de cette thèse est de proposer une approche rigoureuse basée sur les méthodes formelles pour la spécification et l'analyse d'un système Fog avec ses différents points de vues structurels, comportementaux et interactionnels. Nous proposons dans ce travail de thèse une approche incrémentale avec diverses phases de spécification et d'analyse. À l'aide de la technique MDE, nous combinons différentes méthodes formelles et semi formelles afin de définir la structure et le comportement des systèmes Fog. De plus, nous représentons formellement le comportement intelligent et distribué de ces systèmes. Nous validons cette proposition à l'aide des outils et langages adéquats. Nous énumérons nos différentes contributions comme suit :

1. Nous élaborons une étude des travaux de recherche similaires aux nôtres. Nous catégorisons ces travaux selon les théories de base sur lesquels ils se sont appuyés et nous les comparons selon des critères que nous jugeons pertinents.

2. Nous proposons un DSML (*Domain Specific Modelling Language*) un langage pour la modélisation spécifique au domaine du Fog computing (appelé Fog-DSML). L'utilisation d'un DSML permet l'exploitation des modèles tout au long du cycle de développement des systèmes selon diverses dimensions telles que les exigences du système, les fonctionnalités, les données, le traitement, les dépendances, l'architecture et l'infrastructure. Cette approche vise à supporter l'ingénierie des systèmes Fog.
3. Nous proposons une architecture multi-couches pour les systèmes Fog. Cette architecture repose sur de nombreux facteurs entremêlés qui devraient être examinés plus tôt dans la phase de conception. Cette architecture se concentre sur la gestions des nœuds Fog, des éléments Cloud et des éléments IoT, en plus des problèmes de communication entre ces éléments en gérant la communication intra et inter-couches à l'aide d'agents dédiés. En définissant une couche de communication contenant des protocoles internes et externes, nous prenons en considération naturellement la communication entre les différentes couches. Ainsi, la latence temporelle diminue et la répartition des différentes applications en fonction de leur rôle (critique ou non) est possible. Cette architecture constitue la syntaxe concrète du langage Fog-DSML.
4. Nous redéfinissons le formalisme des BiAgents (Agents Bigraphiques), une combinaison des bigraphes [Milner, 2009] et des agents mobiles [Pereira et al., 2012] en donnant une attention particulière au comportement des agents. La nouvelle définition permet aux agents de prendre des décisions après leur interaction et leur analyse. Nous notons ce formalisme BiAgents*.
5. Nous proposons un modèle basé sur le formalisme des BiAgents* pour modéliser à la fois les aspects physiques et virtuels des systèmes Fog. Ce formalisme permet également de représenter son comportement intelligent en utilisant le concept des traces. La partie bigraphique permet de représenter la dispersion géographique des éléments d'un système Fog en plus de leur hétérogénéité. En outre, le comportement dynamique d'un système Fog est défini par une combinaison des règles de réaction bigraphique et de l'évolution des états des agents.
6. Nous validons notre approche en modélisant un cas d'étude, un système d'inspection des bagages dans un terminal d'aéroport. Ce cas d'étude comporte plusieurs scénarios représentant différents comportements qu'un système Fog peut avoir.
7. Nous effectuons une exécution guidée de ce modèle de spécification en utilisant le système autour de Maude stratégie [Eker et al., 2007]. Cette exécution permet de simuler différents scénarios plausibles d'un système Fog. Nous vérifions la faisabilité de notre approche en analysant formellement les comportements spécifiés à travers le model-checker de Maude stratégie. Nous nous intéressons particulièrement à la vérification des propriétés de portabilité des données et leur interopérabilité (syntaxique et sémantique).

1.4 Organisation du manuscrit

Ce document est divisé en cinq chapitres principaux en plus d'une introduction et d'une conclusion générales.
Le Chapitre 2 est consacré à la présentation du paradigme Fog qui constitue le contexte général dans lequel s'inscrivent les travaux présentés dans cette thèse. Il donne une description du Fog computing à travers un historique, ses définitions, ses concepts fondamentaux et ses technologies.
Le Chapitre 3 introduit les principaux concepts et théories formelles, au centre de la thèse, afin de préparer le lecteur à une bonne compréhension des contributions qui y sont présentées.
Le Chapitre 4 expose le principe de notre contribution et recense plusieurs travaux existants liés

à la modélisation formelle des systèmes Fog. Le but est d'identifier les manques et limites des solutions existantes, en vue de dégager les enjeux de cette thèse. De plus, ce chapitre regroupe les différentes syntaxes abstraite et concrète du langage Fog-DSML que nous proposons. Ce chapitre constitue le premier pas vers la spécification formelle d'un système Fog.
Le Chapitre 5 présente la sémantique du langage Fog-DSML et définit l'extension du formalisme BiAgent vers BiAgent* qui sert à la spécification structurelle et comportementale d'un système Fog.
Le Chapitre 6 présente la validation de notre proposition par un cas d'étude (un système d'inspection de bagages dans un terminal d'aéroport). Il décrit l'implémentation, l'exécution et la vérification du comportement intelligent d'un système Fog. Il présente un encodage, dans Maude stratégie, des spécifications BiAgents* introduites ainsi qu'une solution pour la vérification formelle des comportements intelligents à travers une technique de model-checking supportée par la logique LTL.

1.5 Diffusions scientifiques

Les travaux présentés dans ce manuscrit ont fait l'objet des publications citées ci-après.

Revue scientifique avec comité de lecture

— MARIR, Souad, BELALA, Faiza, et HAMEURLAIN, Nabil. A Strategy-Based Formal Approach for Fog Systems Analysis. Future Internet, 2022, vol. 14, no 2, p. 52. [Marir et al., 2022]

Conférences internationales avec comité de lecture

— MARIR, Souad, BELALA, Faiza, et HAMEURLAIN, Nabil. A formal model for interaction specification and analysis in IoT applications. In : International Conference on Model and Data Engineering. Springer, Cham, 2018. p. 371-384. [Marir et al., 2018].

— MARIR, Souad, BELALA, Faiza, et HAMEURLAIN, Nabil. Formal Modeling IoT Systems on the Basis of BiAgents* and Maude. In : 2020 International Conference on Advanced Aspects of Software Engineering (ICAASE). IEEE, 2020. p. 1-7.[Marir et al., 2020].

Première partie

État de l'Art

Chapitre 2

Le Fog Computing : contexte et définitions

Sommaire

2.1 Introduction

Le Fog computing est considéré comme une extension locale du Cloud. Le terme anglais Fog (brouillard en français) se forme au dessus du sol contrairement au Cloud, ou nuage, qui se forme dans le ciel (traduisant l'idée d'une informatique distante). L'expression Fog computing renvoie ainsi à la notion d'une informatique plus proche du monde physique, des terminaux et de l'IoT. Un ingénieur de chez Cisco serait à l'origine du concept [Krim, 2017].
Dans ce chapitre, nous présentons le contexte dans lequel s'inscrit cette thèse. nous présentons dans la Section 2.2 un historique des systèmes d'environnements connectés depuis les prévisions des scientifiques concernant les objets connectés jusqu'à l'élaboration d'une architecture pour le Fog computing. Ensuite, dans la Section 2.3 nous donnons différentes définitions des systèmes Fog selon certaines entreprises et chercheurs. Dans la Section 2.4, nous définissons les piliers du Fog computing. Dans la Section 2.5, nous présentons ses architectures et ses domaines d'application. Enfin, dans la dernière section, nous présentons les avantages d'une des techniques de l'ingénierie des systèmes Fog et nous illustrons un de ces systèmes par un exemple mettant en évidence les différents défis lancés par le paradigme Fog.

2.2 Historique

Nicolas Tesla disait en 1926 : "Le jour où la technologie sans fil sera parfaitement appliquée, la terre entière sera convertie en un immense cerveau, ce qui est en fait le cas. Tous les objets seront des particules d'un tout réel et rythmique et les instruments qui nous permettront de faire ceci seront incroyablement simples en comparaison à notre présent téléphone, un homme sera capable d'avoir un de ces instruments dans sa poche". C'est ainsi qu'il prédit les environnements d'objets connectés [La_Liste_Inconnue, 2022].

En effet, les environnement supportant les objets connectés se déclinent en plusieurs paradigmes tels que l'IoT (Internet of Things), l'IoE (Internet of Eveverything), le CoT (Cloud of Things), le Cloud computing, l'Edge computing, et le Fog Computing. Certaines subtilités différencient ces paradigmes. Dans les années 1990, la société Akamai a lancé son réseau de diffusion de contenu (CDN). L'idée était alors d'introduire des nœuds à des endroits géographiquement plus proches de l'utilisateur final pour la livraison de contenus en cache tels que des images et des vidéos. En 1997, les auteurs de [Noble et al., 1997] ont démontré comment différents types d'applications (navigateurs Web, vidéo et systèmes de reconnaissance de la parole) fonctionnant sur des appareils mobiles à ressources limitées peuvent décharger certaines tâches sur des serveurs puissants (substituts). L'objectif était de diminuer la charge sur les ressources informatiques. Dans un travail ultérieur, ils démontrèrent comment améliorer la durée de vie de la batterie - des appareils mobiles. Aujourd'hui, par exemple, les services de reconnaissance vocale de Google, Apple et Amazon fonctionnent de la même manière. En 2001, les auteurs de [Satyanarayanan, 2001] ont généralisé cette approche en référence à l'informatique pervasive.

Le terme "internet of things" (IoT) a été créé en 1999 par Kevin Ashton [Patel and Patel, 2016], père de la technologie RFID [1], cette année là, le concept est né au MIT (Massachusetts Institute of Technology). À l'issue de la standardisation et de la mécanisation des traitements automatiques de documents et d'information sur support matériel puis numérique, l'IoT est apparue et s'est diffusée rapidement grâce à la mondialisation. À travers le temps, les IoT ont evolué [Patel and Patel, 2016] en passant par les puces RFID et les protocols IP (Internet Protocol) devenant des objets connectés à des serveurs centralisés ou connectés entre eux.

1. Radio Frequency IDentification – Technologie d'identification automatique

En 2001, les applications distribuées évolutives et décentralisées ont utilisé différents réseaux de superposition "*peer-to-peer*" (appelés tables de hachage distribuées). Ces réseaux de superposition auto-organisés permettent un routage efficace et tolérant les pannes, la localisation des objets et l'équilibrage des charges. De plus, ces systèmes permettent également d'exploiter la proximité réseau des connexions physiques sous-jacentes dans l'internet, évitant ainsi les liaisons longue distance entre pairs. Cela non seulement diminue la charge globale du réseau, mais améliore également la latence des applications.

Le Cloud computing est une influence majeure dans l'histoire de l'Edge computing. Il a fait l'objet d'une attention particulière en 2006. L'année où Amazon a fait la promotion de son « Elastic Compute Cloud ». Cela a ouvert de nouvelles possibilités en termes de calcul, de visualisation et de capacité de stockage.

Néanmoins, le Cloud computing en tant que tel n'était pas la solution dans tous les cas d'utilisation. Avec l'avènement des voitures autonomes et de l'IoT (industriel), par exemple, l'accent a été mis de plus en plus sur le traitement local de l'information afin de permettre une prise de décision instantanée.

L'intégration du Cloud computing à l'IoT a permis la création de la notion de Cloud of Things (CoT). Cloud of Things est une plateforme d'application IoT haute performance basée sur le Cloud qui permet de surveiller, gérer et contrôler à distance les périphériques activés par l'IoT. Le "*Cloud of Things*" peut être utilisé pour connecter des appareils et machines afin de les surveiller et de les gérer.

En 2009, dans [Satyanarayanan et al., 2009], les auteurs introduisent le terme Cloudlet. Dans ce travail, l'accent est mis sur la latence. Plus précisément, cet article propose une architecture à deux niveaux. Le premier niveau est connu sous le nom de Cloud (latence élevée), et le second sous le nom de Cloudlet (latence inférieure). Ces dernières sont des composantes décentralisées et largement dispersées de l'infrastructure Internet. Leurs cycles de calcul et leurs ressources de stockage peuvent être exploités par les ordinateurs mobiles à proximité. De plus, un Cloudlet ne stocke qu'un état léger tel que des copies en cache de données.

Entre 2012 et 2013, Cisco a introduit les termes Fog computing et Internet of Everything (IoE). Le Fog computing est une infrastructure matérielle et logicielle distribuée permettant de stocker et de traiter des données provenant d'objets connectés. L'idée derrière cet environnement est de faire appel à des équipements situés à la périphérie du réseau (routeurs, passerelles, commutateurs, appareils mobiles, etc.) pour effectuer le traitement au lieu de centraliser les informations fournies par les capteurs dans le Cloud. L'objectif est d'optimiser les temps de réponse des applications en construisant cette couche intermédiaire au plus près de la génération des données.

L'Internet of Everything (IoE) est un concept qui étend l'accent de l'internet des objets (IoT) sur les communications de machine à machine (M2M) pour décrire un système plus complexe qui englobe également les personnes et les processus. Pour les infrastructures de Cloud dispersées. L'objectif était de promouvoir l'évolutivité de l'IoT, c'est-à-dire de gérer un grand nombre d'appareils IoT et de volumes de mégadonnées pour des applications à faible latence en temps réel.
En 2018, la première proposition d'architecture standard pour le Fog computing basée sur la norme ISO/IEC/IEEE 42010 a été proposée par l'OpenFog Consortium.

Ces différents événements peuvent être illustrés selon la flèche temporelle de la Figure 2.1.

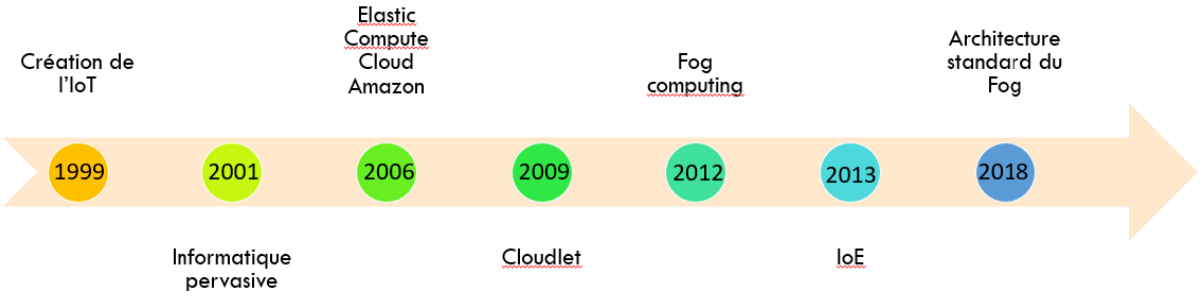


FIGURE 2.1 – Flèche temporelle de l'évolution de l'informatique depuis l'IoT au Fog computing

2.3 Définitions d'un système Fog

Nous trouvons dans la littérature plusieurs efforts de définition des systèmes Fog. En effet, de grandes sociétés telles que IEEE, Cisco et le NIST ont contribué à ces définitions ainsi que plusieurs chercheurs dans le domaine des systèmes complexes. Selon le standard IEEE OPFRA001.020817 [Association et al., 2018] un système Fog peut être défini comme suit :

Définition 1 Un système Fog distribue les fonctions de traitement, de stockage, de contrôle et de mise en réseau plus près des utilisateurs, le long d'un continuum Cloud à objet.

Un système Fog est défini selon Cisco par :

Définition 2 Un système Fog est une infrastructure informatique décentralisée étendant un système Cloud en plaçant stratégiquement des nœuds entre les serveurs Cloud et les objets. Cela rapproche les données, le traitement, le stockage et les applications de l'utilisateur ou du dispositif IoT où les données doivent être traitées, créant ainsi un "brouillard" en dehors du Cloud centralisé et réduisant les temps de transfert de données nécessaires au traitement des données.

Le NIST (National Institue of Standards and Technology) défini un système Fog comme suit :

Définition 3 Un système Fog comporte des ressources horizontales, physiques ou virtuelles qui résident entre les terminaux intelligents et les centres de données ou de Cloud traditionnels. Ce type de systèmes prend en charge des applications isolées verticalement et sensibles à la latence en fournissant une connectivité informatique, de stockage et de réseau omniprésente, évolutive, stratifiée, fédérée et distribuée.

Nous pouvons déduire de ces différentes définitions qu'un système Fog est une extension d'un système IoT classique par ajout de nœuds de stockage et de contrôle entre les dispositifs impliqués et les serveurs Cloud. Le traitement des données ne se fait plus uniquement au niveau du Cloud mais aussi dans les nœuds Fog en périphérie du réseau.
Nous identifions les avantages du Fog computing par rapport au Cloud et Edge computing en rappelant la comparaison publiée dans [Yousefpour et al., 2019]. Le Tableau 2.1 résume cette comparaison.

TABLE 2.1 – Avantages du du Fog Computing

Fog	**Cloud**
Temps de latence réduit	Temps de latence maintenu
Décentralisation	Centralisation
Diminution de la consommation d'énergie	Augmentation de la consommation d'énergie
Matériel prend moins de place	Matériel prend plus de place
Accès au stockage et traitement depuis la bordure du réseau	Accès au stockage et traitement uniquement depuis le Cloud distant
Pas besoin de connexion internet continue	Besoin de connexion internet continue
Fog	**Edge computing**
Le traitement, la mise en réseau, le stockage et le contrôle peuvent se faire n'importe où entre le Cloud et les objets	Le traitement, la mise en réseau, le stockage et le contrôle ne se font que dans le Edge

2.4 Piliers d'un système Fog

Le Fog computing a été conçu sur un ensemble de principes de base que l'on considère comme étant ses piliers [Association et al., 2018]. Ils représentent les attributs clés d'un système Fog assurant la distribution de calcul, de stockage, de contrôle, et des fonctions de mise en réseau plus proches de la source de données (utilisateurs, objets, etc.) le long du continuum Cloud-à-objet. Ces piliers sont énumérés comme suit (voir Figure 2.2) :

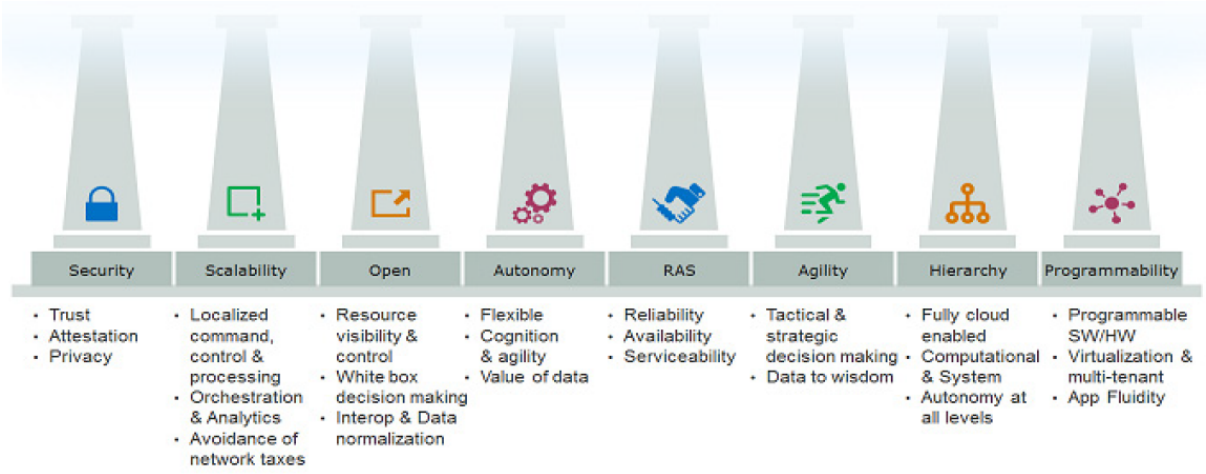


FIGURE 2.2 – Piliers du Fog computing [IEEE, 2018]

— **La sécurité** des objets, des nœuds Fog, des serveurs Cloud et tout le réseau reliant ces éléments. Tous les nœuds Fog doivent utiliser une racine de confiance basée sur le matériel. La racine de confiance matérielle est un composant matériel de confiance qui reçoit le contrôle à la mise sous tension. Elle étend ensuite la chaîne de confiance à d'autres composants matériels, micrologiciels et logiciels. La racine de confiance doit ensuite pouvoir être attestée par des agents logiciels fonctionnant à l'intérieur et à travers l'infrastructure. En raison de leur proximité avec la périphérie, les nœuds des réseaux Fog jouent souvent le rôle de premier nœud de contrôle d'accès et de cryptage.

— **La scalabilité** répond aux besoins dynamiques et techniques derrière le déploiement d'un système Fog. Les propriétés hiérarchiques du Fog et sa localisation dans les bordures logiques du réseau offrent des possibilités d'évolution supplémentaires. Afin de garantir cette

propriété, plusieurs éléments doivent être scalables tels que la performance, la capacité, la fiabilité, la sécurité, le hardware et les logiciels. La scalabilité permet aux nœuds Fog de fournir une assistance de base pour répondre aux besoins de l'entreprise et permet un modèle de paiement au fur et à mesure de la croissance du FaaS (Fog as a service), ce qui est essentiel pour l'économie de son déploiement initial et son succès à long terme.

— **La transparence** comme principe fondamental permet aux nœuds Fog d'exister n'importe où dans un réseau et dans les réseaux accessibles. La transparence est définie par :

  - **La composabilité** favorise la portabilité et la fluidité des applications et des services à l'instanciation. L'importance de la composabilité est également visible dans le pilier de la programmabilité.
  - **L'interopérabilité** permet de découvrir en toute sécurité les capacités de calcul, de réseau et de stockage et favorise la fluidité et la portabilité pendant l'exécution. Le marché a clairement exprimé son désir d'un écosystème de fournisseurs dynamique, avec des attentes raisonnables que les éléments d'un fournisseur puissent être librement substitués aux éléments d'un autre fournisseur.
  - **La communication ouverte** permet des fonctionnalités telles que la mise en commun des ressources à la périphérie du réseau pour collecter la puissance de traitement, la capacité de stockage, la capacité de détection et la connectivité sans fil inutilisées. Par exemple, une application de calcul intensif développée dans une architecture Fog peut tirer parti de centaines de gigaoctets inutilisés sur des ordinateurs portables, des systèmes et des décodeurs situés à proximité dans un foyer chaque soir, ou parmi les passagers d'un système de transport public. La découverte ouverte de ces ressources informatiques proches est essentielle. En effectuant le travail fonctionnel au plus près de la périphérie, les taxes supplémentaires sur le réseau sont évitées lors de la montée de la pile vers le Cloud. Les taxes de réseau sont définies par le coût de la transmission.
  - **L'emplacement non défini** de toute instance de nœud afin de garantir que les nœuds puissent être déployés n'importe où dans la hiérarchie. La transparence de l'emplacement offre une alternative au contrôle de l'opérateur réseau. Cela signifie que tout dispositif IoT, tel qu'une montre intelligente, n'a pas besoin de son propre plan de données appartenant à l'opérateur. Chaque chose ou entité logicielle peut observer ses conditions locales et prendre des décisions sur le réseau à rejoindre. Chaque point d'extrémité d'un réseau Fog peut optimiser son chemin vers les ressources de calcul, de réseau et de stockage dont il a besoin (peu importe que ces ressources se trouvent dans les couches hiérarchiques du Fog ou dans le Cloud).

— **L'autonomie** permet aux nœuds Fog de continuer à fournir les fonctionnalités prévues en cas de défaillance des services externes, l'autonomie est donc soutenue tout au long de la hiérarchie. La prise de décision se fait à tous les niveaux de la hiérarchie d'un déploiement, y compris près du dispositif ou des couches d'ordre supérieur. La prise de décision centralisée dans le Cloud n'est plus la seule option. L'autonomie à la périphérie du réseau signifie que l'intelligence des dispositifs locaux et les données des pairs peuvent être utilisées pour remplir la mission de l'entreprise à l'endroit où cela a le plus de sens.

— **La programmabilité** permet des déploiements hautement adaptatifs, y compris la prise en charge de la programmation au niveau des couches logicielles et matérielles. Cela signifie que la réaffectation d'un nœud ou d'un groupe de nœuds Fog pour tenir compte de la dynamique opérationnelle peut être entièrement automatisée. La réaffectation peut

être effectuée à l'aide des interfaces de programmabilité inhérentes aux nœuds Fog. La programmabilité d'un nœud Fog présente plusieurs avantages tels qu'une infrastructure adaptative, des déploiements efficaces en terme de ressources et une sécurité améliorée.

— **La fiabilité, disponibilité et l'aptitude au service** du hardware, des logiciels et des opérations sur ces derniers. Ces caractéristiques sont présentes dans toutes les architectures de système réussies et sont de grande importance dans l'architecture Fog. Le matériel, les logiciels et les opérations sont les trois principaux domaines de ces piliers. Un déploiement fiable continuera à assurer la fonctionnalité prévue dans des conditions d'exploitation normales et défavorables.

— **L'agilité** concerne les décisions opérationnelles pour le déploiement de l'architecture Fog. Il n'est pas possible pour les humains seuls d'analyser les données générées à l'échelle prévue par l'IoT comme base de décisions commerciales et opérationnelles rapides et judicieuses. Le pilier de l'agilité se concentre sur la transformation de ce volume de données en informations exploitables. L'agilité traite également de la nature hautement dynamique des déploiements du Fog et de la nécessité de réagir rapidement aux changements.

— **La hiérarchisation**. Les ressources informatiques d'un système Fog peuvent être considérées comme une hiérarchie logique basée sur les exigences fonctionnelles d'un système IoT de bout en bout. Selon l'échelle et la nature du scénario abordé, la hiérarchie peut être un réseau de systèmes cloisonnés intelligents et connectés disposés en couches physiques ou logiques, ou bien elle peut se grouper en un seul système physique.

L'OpenFog Consortium ont défini ces piliers sans pour autant qu'il y ait obligation qu'un système Fog repose sur absolument tous ces derniers. Selon le domaine et l'application, les priorités varient. Certains systèmes favorisent des propriétés par rapport à d'autres selon la demande.

2.5 Architectures de référence

Le Fog computing vise des préoccupations transversales telles que le contrôle des performances, la latence et l'efficacité qui sont essentielles au succès des réseaux Fog. Le Cloud et le Fog sont sur un continuum mutuellement bénéfique et interdépendant, certaines fonctions sont naturellement plus avantageuses à réaliser dans les nœuds Fog, tandis que d'autres sont mieux adaptées au Cloud. Le "*backend*" Cloud traditionnel restera une partie importante des systèmes informatiques à mesure que le Fog computing se dessine. La segmentation de ce qui doit être dirigé vers les nœuds Fog et ce qui va au backend Cloud sont spécifiques à l'application. Cette segmentation pourrait être planifiée, mais aussi changer dynamiquement si l'état du réseau change dans des domaines tels que les charges des processeurs, les bandes passantes des liaisons, les capacités de stockage, les cas d'erreur, les menaces de sécurité, les hausses de coûts, etc.

2.5.1 Architecture de "l'OpenFog consortium"

La Figure 2.3 illustre l'architecture de référence pour le Fog computing proposée par l'OpenFog consortium [IEEE, 2018]. Cette architecture comporte plusieurs couches constituant un système Fog. En effet, cette architecture contient plusieurs couches de sécurité, de gestion et

d'application. Ces dernières sont définis avec des perspectives de garantie de propriétés de sécurité, de manageabilité et d'analyse de données entre autres.

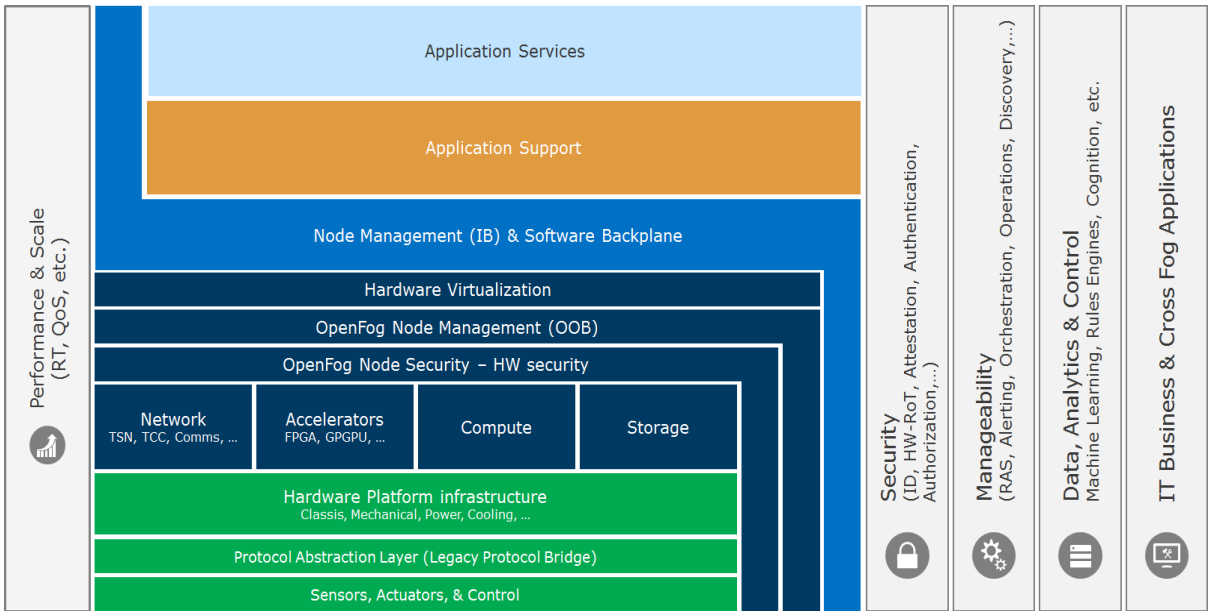


FIGURE 2.3 – Description d'une Architecture Fog [IEEE, 2018]

Cette architecture est définie selon différents points de vues et perspectives. En effet, pour l'élaboration de cette architecture, OpenFog Consortium s'est basé sur le standard ISO/IEC/ IEEE 42010 :2011 afin de définir une architecture standard pour les systèmes Fog. Ainsi, ils ont définis l'architecture selon les point de vues fonctionnel et de déploiement. Cette architecture peut être étudiée sous forme d'un groupe de nœuds (les deux couches inférieures de l'architecture dans la Figure 2.3), de systèmes (les couches médianes de l'architecture incluant la virtualisation des dispositifs physiques) ou de logiciels (les trois couches supérieures) selon les vues présentées. Il existe d'autres efforts de standardisation des systèmes Fog par de grandes sociétés telles que le NIST ou encore Cisco.

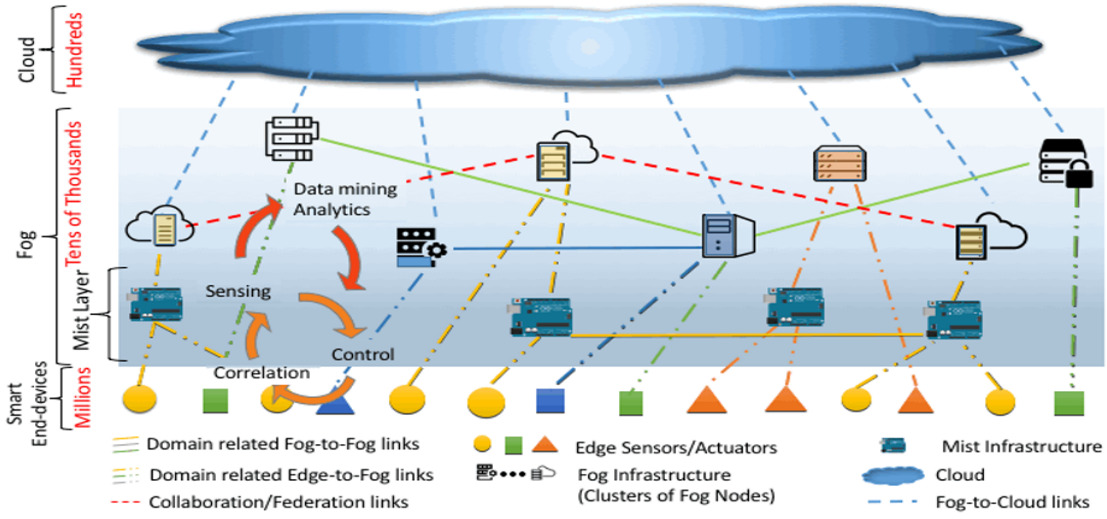


FIGURE 2.4 – Architecture pour le Fog computing proposée par le NIST [McKendrick,]

Les nœuds Fog dans l'architecture proposée par le NIST [McKendrick,] sont sensibles au

contexte et supportent un système commun de gestion des données et de communication. Ils peuvent être organisés en clusters, soit verticalement (pour supporter l'isolation), soit horizontalement (pour supporter la fédération), par rapport à la distance de latence des nœuds Fog ou par rapport aux équipements terminaux intelligents. Le Fog computing minimise le temps de réponse des demandes aux applications prises en charge et fournit aux équipements terminaux des ressources informatiques locales et, si nécessaire, une connectivité réseau aux services centralisés. Le NIST a donc proposé un autre type de nœud Fog, le Mist.

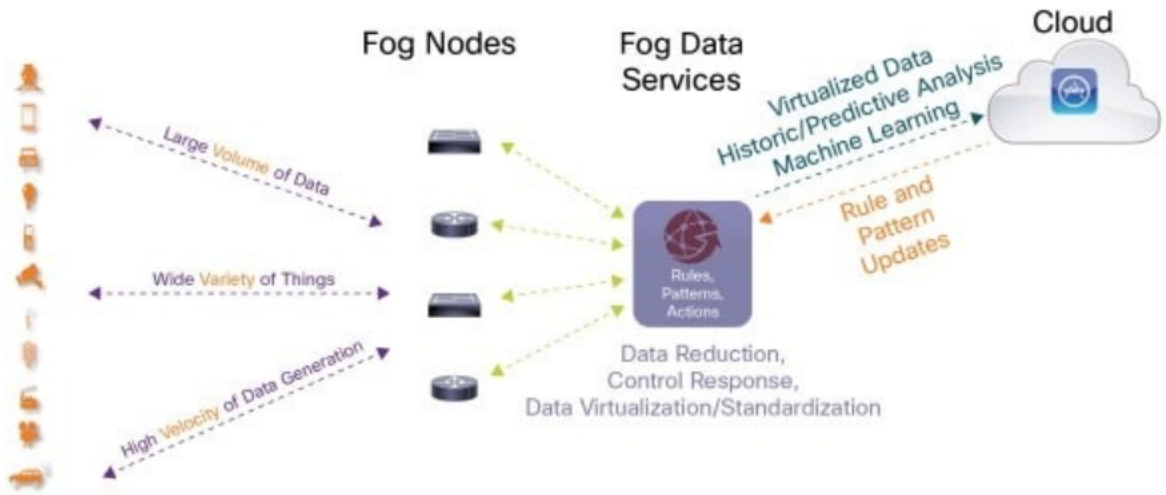


FIGURE 2.5 – Architecture Fog proposée par Cisco [Iorga et al., 2018]

L'architecture proposée par Cisco [Iorga et al., 2018] décrit le rôle de l'IoT dans le développement du Fog computing. La couche Fog est chargée d'assembler les données collectées par les capteurs et les appareils terminaux, de traiter ces données localement, puis de répondre par des commandes de contrôle (requêtes) aux actionneurs. D'autre part, la couche Fog est capable de filtrer, d'agréger, puis de rediriger ces données, éventuellement, vers un centre de données distant.

En plus de ces propositions d'architectures standards, la communauté de chercheurs a manifesté son intérêt vers le Fog computing en proposant des architectures adaptées aux différents domaines d'application de ce nouveau paradigme. Dans ce qui suit, nous présentons quelques exemples de ces architectures.

2.5.2 Architectures par domaine d'application

Contrairement au paradigmes tels que le Cloud, le Fog computing ne possède pas encore d'architecture standard proprement définie. Dans cette section, nous donnons un exemple d'architecture par domaine d'application.

Transport intelligent : Les auteurs de [Asadi et al., 2021] supposent que les véhicules peuvent communiquer avec des serveurs d'application disponibles via des dispositifs portatifs du conducteur tels qu'un smartphone (machine client). Les machines clientes peuvent utiliser la communication de données via un réseau mobile enregistré ou des réseaux WiFi disponibles. Les serveurs d'application peuvent être déployés dans le Cloud ou les nœuds Fog. Les nœuds Fog sont placés à proximité du réseau pour réduire le trafic du cœur du réseau et réduire le temps de réponse de l'application. Ce système Fog est illustré par la Figure 2.6.

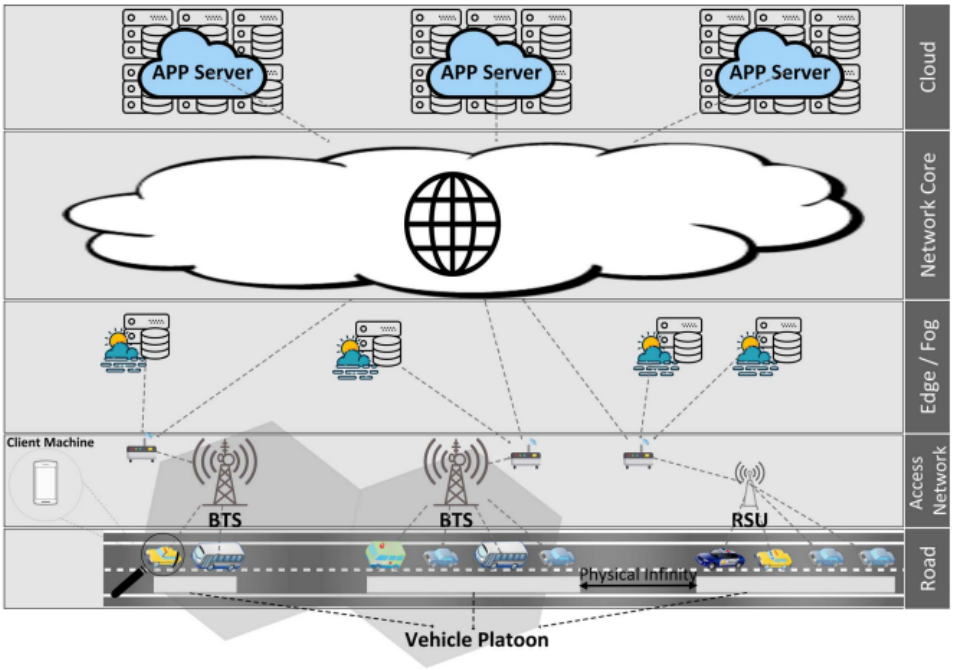


FIGURE 2.6 – Architecture conceptuelle d'un système de transport intelligent [Asadi et al., 2021]

Agriculture : Dans [Lee et al., 2020], les auteurs proposent une stratégie qui assigne une partie des couches d'analyse Deep Learning aux nœuds Fog dans un environnement d'agriculture intelligente basé sur le Fog computing. L'architecture dans la Figure 2.7, proposée par [Lee et al., 2020] montre qu'une application recueille des données multimédias à partir de dispositifs, et demande l'analyse des informations requises d'un serveur Cloud avec une puissance de calcul suffisante. Un algorithme pour application sur le Cloud, extrait des informations significatives des données reçues. Ensuite, les informations retournées du Cloud sont utilisées pour exécuter des fonctions définies de l'application. Les appareils transfèrent des données vers un Cloud ou une passerelle selon l'infrastructure réseau. Si la connexion au Cloud n'est pas possible, les appareils transfèrent des données vers une passerelle voisine qui peut les transférer vers le Cloud.

Bâtiments intelligents : La Figure 2.8 présente une vue d'ensemble de haut niveau du DECENTER proposé par [Kochovski and Stankovski, 2021]. L'architecture possède trois couches principales : Infrastructure, Plateforme et Application. La couche infrastructure se compose de différents types de matériel déployé dans le continuum Cloud-to-Edge, couvrant ainsi le pipeline complet de données. Les ressources de calcul dans le niveau Fog, qui se situent entre le niveau Cloud et le niveau Edge offrent de meilleures performances de réseau que les ressources du niveau Cloud ou Edge. Le niveau Cloud est réservé aux tâches nécessitant une puissance de calcul très élevée, de grandes capacités de stockage, mais des performances réseau plus faibles. Les tâches qui nécessitent les caractéristiques de ce niveau peuvent être exécutées sur des services Cloud privés ou publics. La couche plateforme de l'architecture est conçue pour soutenir des scénarios où les fournisseurs d'infrastructures peuvent interagir entre eux et fédérer leurs ressources afin de fournir des services aux utilisateurs. La couche plateforme est placée entre la couche Infrastructure et la couche Application. Elle contient des composants logiques qui permettent la découverte et l'orchestration des ressources dans le continuum Cloud-à-Edge. La couche d'application offre des applications et des services d'application préétablis. Les services d'application peuvent être considérés comme des microservices qui peuvent être implémentés dans différentes applications.

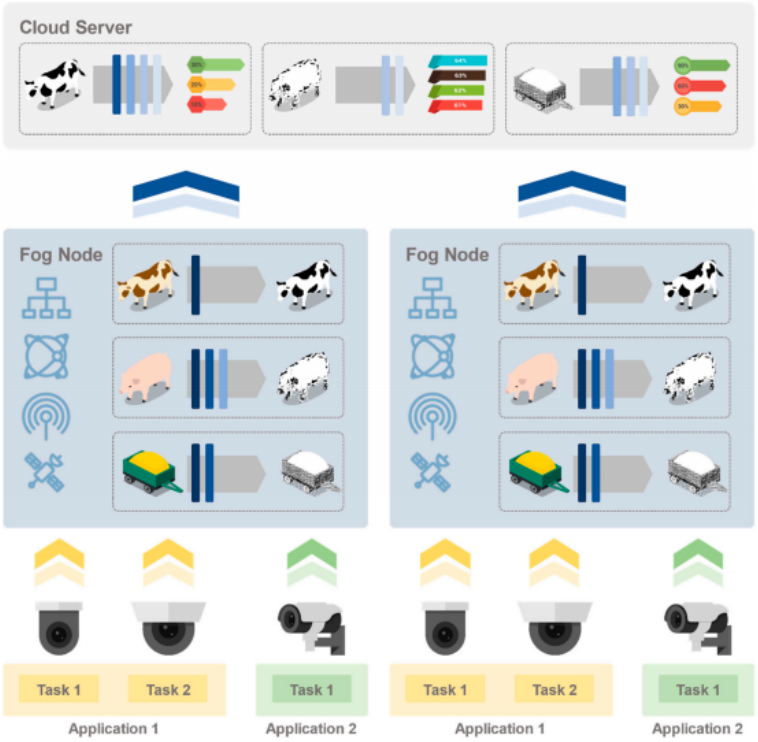


FIGURE 2.7 – Exemple de système intelligent d'agriculture basé sur le Fog computing [Lee et al., 2020]

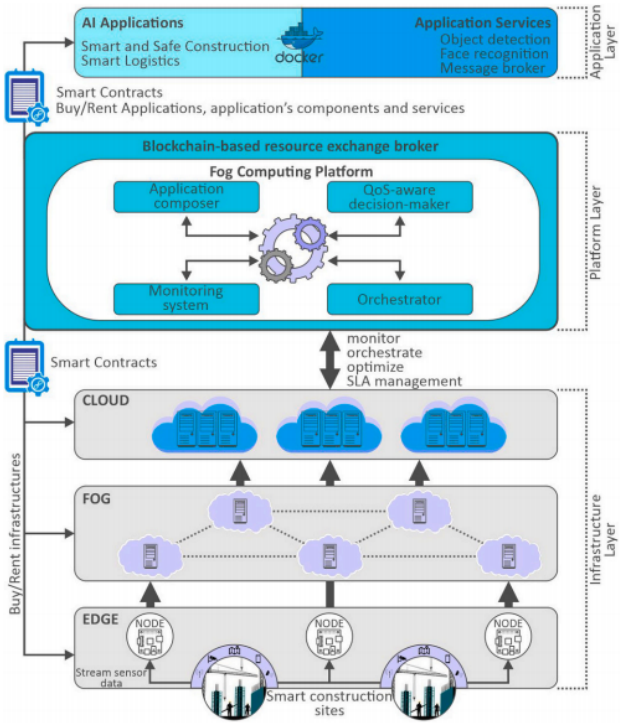


FIGURE 2.8 – Architecture décentralisée d'un système de construction intelligente [Kochovski and Stankovski, 2021]

Soins de santé : Les auteurs de [Li et al., 2022] proposent un système de santé basé sur le Fog comme le montre la Figure 2.9. Dans cette proposition, le Fog computing agit comme une plate-forme intermédiaire entre le Cloud et les utilisateurs. Les utilisateurs téléchargent leurs données sur des nœuds Fog à travers des appareils intelligents. Ces derniers peuvent traiter et stocker les données, dans les nœuds Fog qui possèdent la capacité de calcul, de stockage et de connexion. De plus, ces nœuds peuvent transmettre des données au Cloud et fournir un service de recherche de données pour les médecins authentifiés, ce qui aide les médecins à déterminer le traitement correspondant pour les patients.

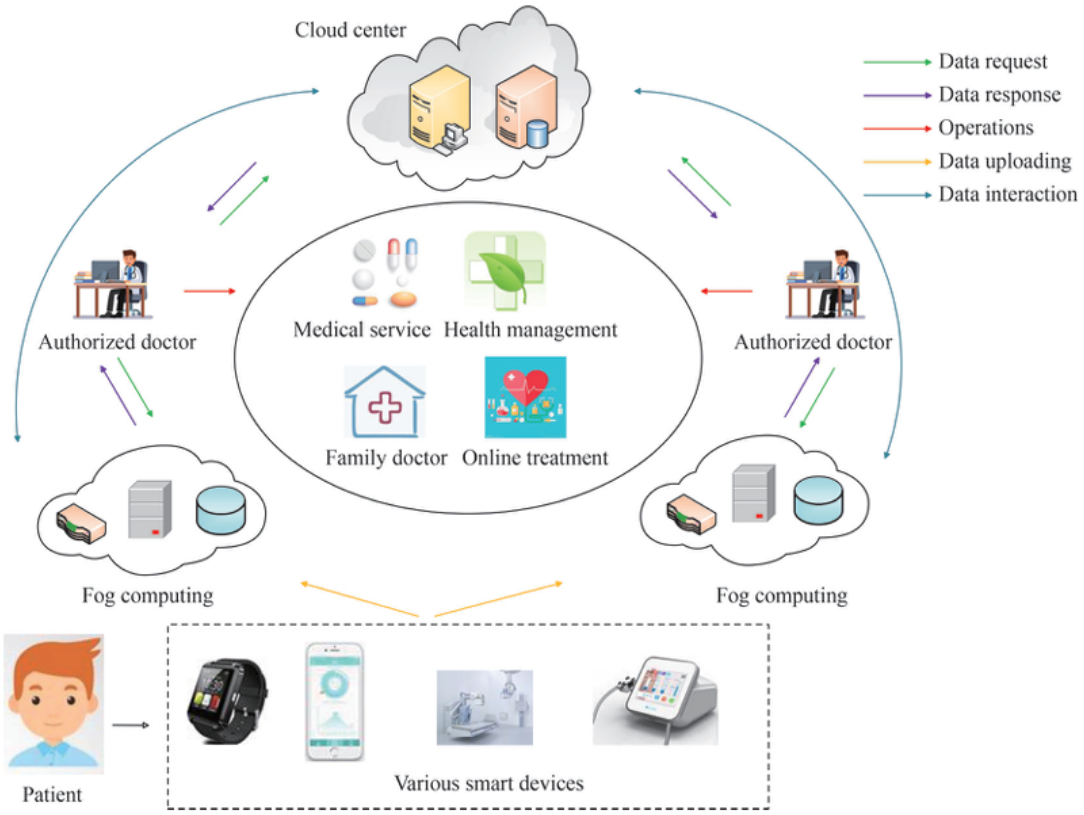


FIGURE 2.9 – Système de santé basé sur le Fog [Li et al., 2022]

Services financiers : Dans [Hernández-Nieves et al., 2020] une nouvelle solution de Fog Computing est proposée, développée dans le domaine de la fintech [2]. Il intègre des systèmes prédictifs dans le processus de prestation de services personnalisés à la clientèle pour la recommandation des produits d'une entité bancaire. L'architecture proposée comprend des nœuds Fog où les données sont traitées par des agents intelligents légers permettant la mise en œuvre de systèmes de recommandation contextuels ainsi que la configuration d'un raisonnement basé-cas dans la couche Cloud. Le système de recommandation est la pierre angulaire de l'architecture comportant des produits bancaires, tels que les hypothèques, les prêts, les plans de retraite, etc.

Nous remarquons que pour chaque domaine, il y a la notion de nœuds Fog responsables de traitement ou de stockage ou des deux. Selon le domaine, ces nœuds sont soient plus proches des dispositifs IoT, soient des serveurs Cloud en fonction de leurs rôles. Dans tous les cas, le but de la définition de ces architecture est d'optimiser le fonctionnement des systèmes selon le besoin des utilisateurs.

2. Contraction de "Finance" et de "Technologie", le terme Fintech désigne les acteurs technologiques qui bouleversent les secteurs bancaire et financier.

2.6 Ingénierie d'un système Fog

L'ingénierie de système [Faisandier, 2011] est l'ensemble des activités qui permet de concevoir un système optimal pour répondre à un besoin ou une opportunité. Elle utilise la vision système qui englobe les multiples paramètres, l'aspect pluridisciplinaire, tout le cycle de vie, les contextes d'utilisation. Elle consiste à :

— Établir la compatibilité fonctionnelle et physique du système avec les besoins et les contraintes ;

— Équilibrer l'économie globale de la solution sur toutes les étapes de la vie du système (vue de l'acquéreur) ;

— Rechercher l'équilibre entre contraintes, performances, coûts, délais et risques (vue du concepteur).

L'ingénierie est basée sur plusieurs fondations selon les auteurs de [Borque and Fairley, 2014] dont la plus importante étant la modélisation, la simulation et le prototypage. En particulier, les modèles et les méthodes de l'ingénierie logicielle des systèmes (voir Figure 2.10) visent plusieurs objectifs. Chacun est réalisé selon un type spécifique de modèle (comportemental, informationnel ou structurel). Parmi les méthodes d'ingénierie des logiciels, les méthodes formelles sont les plus empruntées pour procéder aux différentes analyses des propriétés du modèle (complétude, consistance, exactitude, etc). La nécessité de recouvrir aux méthodes formelles s'impose plus lorsque les systèmes informatiques sont complexes et leur ingénierie est coûteuse. Évidemment, les systèmes Fog et leur observation constituent un champ d'exploration et de recherche qui mérite beaucoup d'attention.

2.6.1 Apports de la technique MDE

En Génie Logiciel, le MDE (Model Driven Engineering) est l'application des modèles au développement du logiciel. En effet, avec l'accroissement de la compléxité des systèmes informatiques, nous nous sommes appuyés sur de plus en plus sur la modélisation pour maîtriser cette complexité, tant pour produire le logiciel (conception) que pour le valider (test). L'intérêt de l'MDE a été amplifié à la fin du XX^e siècle par L'OMG (Object Management Group) dans une démarche de génération d'une application informatique à partir de modèles. Plusieurs avantages découlent de l'utilisation de cette démarche, nous pouvons citer :

1. Moins de bugs. En effet, dans la technique MDE, toutes les parties prenantes d'un logiciel travaillent ensemble au sein d'un modèle unique, en limitant les erreurs au minimum. Pour des types de logiciels plus complexes, cette méthodologie améliore la transparence, aboutit à un mode de travail plus structuré et facilite la supervision du processus. Tous les bugues peuvent être enlevés dès le début du processus de bout en bout au lieu de l'achèvement, évitant les pertes de temps inutiles et les dépenses hors de contrôle.
2. Une unification des documents de travail. Cette technique élimine le besoin de feuilles de calcul Excel séparées, de documents Word, de diagrammes et de PDF contenant toutes sortes d'informations disjointes sur le logiciel. Les utilisateurs peuvent modifier le modèle directement à tout moment, ce qui permet de le convertir automatiquement en code, réduisant ainsi considérablement le temps nécessaire au déploiement des modifications. Il permet également de remplacer plus facilement et plus efficacement les logiciels désuets (anciens).
3. Elle crée des prototypes améliorés. Elle rend le prototypage plus accessible et économique, puisque la méthode permet de créer des prototypes virtuels. Les entreprises peuvent utiliser le modèle pour faire des calculs dans le logiciel avant qu'il ne soit intégré dans une machine. En utilisant le prototype virtuel pour tester le logiciel, on obtient un prototype

physique plus solide et plus performant. C'est une solution parfaite, en particulier, pour les entreprises utilisant des équipements et des dispositifs complexes avec des prototypes très coûteux, comme les entreprises opérant dans les industries de haute technologie, médicale et de défense.

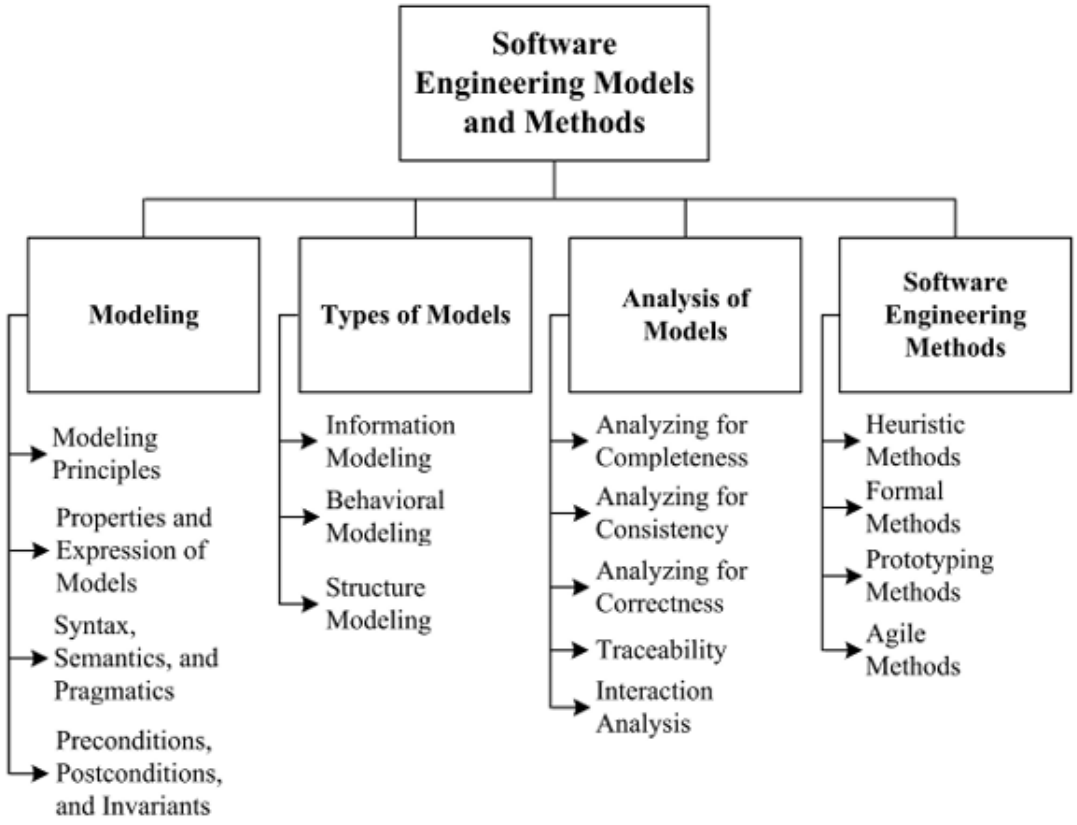


FIGURE 2.10 – Modèles et méthodes du génie logiciel [Borque and Fairley, 2014]

2.6.2 Exemple de motivation

Afin de profiter des avantages de la technique MDE pour aborder le développement des systèmes Fog et pallier aux différents gaps qui existent entre les technologies innovantes utilisés dans ces systèmes et les résultats théoriques quant à leur modélisation, nous considérons dans cette section, un exemple d'un système Fog de transport (voitures intelligentes) [Association et al., 2018]. Nous identifions à travers quelque scénarios de son comportement, les enjeux liés à la modélisation de son architecture et son comportement.

Les voitures autonomes intelligentes génèrent de multiples téraoctets de données chaque jour à partir des combinaisons de détection et de portée de lumière, de systèmes de positionnement global (GPS), de caméras, etc. L'OpenFog consortium atteste que lorsque la voiture intelligente est couplée à une infrastructure intelligente, un modèle Cloud ne fonctionnera pas pour le transport autonome, et qu'une approche Fog est nécessaire.

Dans cet exemple, le véhicule est un nœud Fog mobile qui communique avec d'autres nœuds Fog. Ce dernier doit également être capable d'effectuer toutes les opérations requises à bord du véhicule de manière autonome lorsqu'il ne peut se connecter à d'autres nœuds Fog ou au Cloud.

Les nœuds Fog embarqués fournissent des services tels que l'info-divertissement, les systèmes avancés d'assistance à la conduite (ADAS), la conduite autonome, la prévention des collisions, la navigation, etc. Plusieurs technologies de mise en réseau différentes, dont les communications à courte portée (DSRC), les technologies cellulaires (par exemple 3G, LTE, 5G, etc.) et d'autres technologies de mise en réseau relient en toute sécurité les véhicules entre eux et l'infrastructure.

Le réseau Fog de transport se compose d'une hiérarchie à trois niveaux de nœuds Fog (voir Figure 2.11).

— Le premier niveau de la hiérarchie est les nœuds Fog d'infrastructure, ou nœuds Fog routier. À ce niveau, les capteurs Fog routier recueillent des données provenant d'autres dispositifs tels que les caméras routières. Les nœuds Fog effectuent une analyse locale pour l'action locale, comme alerter le véhicule sur les mauvaises conditions routières, déclencher une réponse autonome pour ralentir, et effectuer certaines fonctions autonomes, même si les connexions aux couches supérieures ne sont pas disponibles.

— Le deuxième niveau concerne les nœuds du trafic routier du voisinage. Les données du premier niveau d'interactions sont agrégées et envoyées vers le ces derniers.

— Le troisième niveau représente les nœuds Fog du trafic régional. Les nœuds du deuxième et troisième niveau procèdent à une analyse et une distribution plus poussées que ceux du premier. Certaines données peuvent également être distribuées est-ouest à d'autres nœuds d'infrastructure pour leur utilisation.

Typiquement, chaque couche Fog dans la hiérarchie fournira des capacités supplémentaires de traitement, de stockage et de réseau en service de l'application verticale à leur niveau de la hiérarchie. Par exemple, les couches de niveau supérieur fournissent un traitement supplémentaire pour fournir une analyse de données ou de grandes capacités de stockage.
Les nœuds Fog de contrôle de la circulation peuvent recevoir d'autres sources, comme les systèmes intelligents de feux de circulation, les gestionnaires municipaux et les systèmes basés sur le Cloud. Les flux de données entre le système de contrôle du trafic, les nœuds Fog d'infrastructure et les véhicules dans toutes les directions, assurant tous les niveaux de la hiérarchie ont les capacités de données et de contrôle dont ils ont besoin.

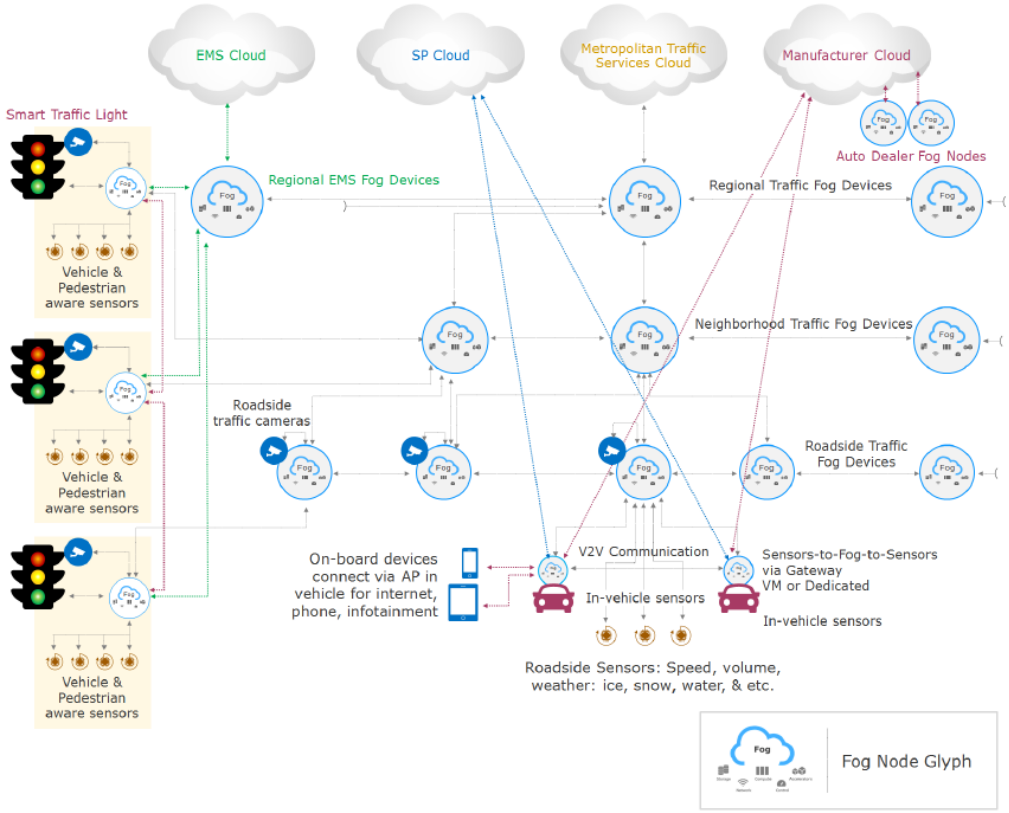


FIGURE 2.11 – OpenFog Transportation : Voiture intelligente et système de contrôle du trafic [Association et al., 2018]

À partir de cet exemple nous déduisons q'un système Fog est un système complexe sachant que la structure et le comportement d'un système complexe ne sont pas déductibles, ses composants peuvent changer et augmenter sa complexité en termes d'arrangement de ses éléments selon [Norman and Kuras, 2006].
Cet exemple montre différents défis lancés par l'ingénierie et la conception de ces systèmes Fog. Ces défis peuvent être classifiés comme étant des défis de conception, d'analyse ou encore de développement. Nous catégorisons ces défis comme suit :

— Hétérogénéité des différents objets impliqués en bordure de réseau, les feux de signalisation, les véhicules, les nœuds Fog à chaque niveau, les protocoles de communication et les serveurs Cloud. Les nœuds Fog ou dispositifs terminaux sont créés par de nombreux fabricants et existent dans une gamme de sortes qui doivent être placées en fonction de leurs plateformes. Il est nécessaire de pouvoir représenter ces éléments malgré leurs différences.

— Distribution géographique des éléments qui constituent le système. La représentation de la localité ainsi que la connectivité entre les différents éléments est primordiale. En effet, le Fog, contrairement au Cloud centralisé, offre des services et des applications distribués qui peuvent être placés n'importe où et les objets impliqués dans un système Fog sont distribués dans un large spectre qu'il faut identifier. Il est donc nécessaire de définir les connexions entre les différents éléments.

— Dynamicité, c'est à dire la gestion du changement de l'état du système dans le temps. Sachant que le système passe par plusieurs états d'analyse et de gestion des données. Nous devons définir les différents états par lesquelles passe le système afin de prévoir son comportement.

— Intelligence dans les prises de décision du système Fog. En effet, plusieurs nœuds Fog de l'architecture permettent d'effectuer une analyse de plus en plus approfondie selon leur niveau hiérarchique.
— Interopérabilité syntaxique et sémantique des réseaux et des plateformes. Les données échangées entre les éléments du système doivent être compréhensibles par les différents éléments du systèmes ainsi qu'entre les plateformes partageant ces données à travers des protocoles de communication bien définis. Les réseaux auxquels ces systèmes sont reliés doivent aussi pouvoir communiquer des données cohérentes afin de permettre le bon déroulement de l'exécution des différents systèmes Fog.
— Sensibilité au contexte, signifiant la réactivité du système selon son environnement. Nous remarquons ici les véhicules qui se comportent selon le fonctionnement des feux de circulation.

Le Fog computing est un paradigme vaste et complexe nécessitant une syntaxe proche de ce domaine autour de laquelle plusieurs acteurs peuvent coopérer pour concevoir, analyser, gérer les performances et tester un logiciel développé dans ce contexte.

2.7 Conclusion

Dans ce qui a précédé, nous avons présenté le contexte dans lequel s'inscrit le travail de notre thèse en présentant un historique depuis la création de la notion d'internet des objets au Fog computing en passant par les écosystèmes où évoluent les paradigmes reliés à ce dernier tel que le Cloud. Par la suite, nous avons présenté différentes définitions des systèmes Fog, leurs piliers ainsi que quelque efforts de standardisation des architectures Fog. Nous avons aussi présenté des architectures dédiées au Fog computing selon leur domaine d'application. En présentant l'apport de techniques de l'ingénierie des systèmes complexes telle que l'MDE, ainsi qu'un exemple illustratif des systèmes Fog, nous avons déduit différents défis lancés par ces systèmes quant à leur conception.

Chapitre 3

Fondements mathématiques

Sommaire

3.1 Introduction

Dans ce chapitre, nous présentons les différents concepts de base dont le lecteur aura besoin afin de bien comprendre les contributions de cette thèse. Nous nous intéressons dans la Section 3.2 aux langages de modélisation spécifiques au domaine (DSML). Nous présentons ce qu'est un DSML avec ses différentes syntaxes et sémantique. Dans la Section 3.3 nous introduisons les systèmes réactifs bigraphique (BRS : *"Bigraphic Reactive Systems"*). Nous présentons les différentes définitions ainsi que les opérations possibles sur les Bigraphes. La Section 3.4 définit l'extension BiAgents des BRS avec ses différentes structures. Enfin, la dernière Section 3.5 s'intéresse au langage de spécification Maude et son extension Maude stratégie. Nous y présentons leurs différentes caractéristiques ainsi que leur fonctionnement.

3.2 Langage de Modélisation Spécifique au Domaine (DSML)

Dans le domaine de l'ingénierie dirigée par les modèles (MDE), les langages spécifiques au domaine constituent une solution ayant un niveau d'abstraction plus élevé qu'un langage dédié de programmation, pour les utilisateurs experts du domaine afin de manipuler ses connaissances et ses concepts. Les langages de modélisation spécifiques au domaine (DSML) formalisent la structure, le comportement et les exigences des applications dans des domaines particuliers. Ces langages ont tendance à supporter des abstractions de plus haut niveau que les langages de modélisation généraux, et sont plus proches du domaine du problème que du domaine de l'implémentation.

3.2.1 Définition

Un DSML suit les abstractions et la sémantique du domaine, permettant aux concepteurs de se percevoir comme travaillant directement avec les concepts du domaine. De plus, les règles du domaine peuvent être incluses dans le langage sous forme de contraintes, interdisant la spécification de modèles incorrects.
Un DSML est défini par un ensemble de modèles qui sont conformes avec la syntaxe abstraite du langage, représentés par plusieurs syntaxes concrètes et satisfaisant une sémantique donnée [Da Silva, 2015]. Une définition formelle d'un langage de modélisation est donnée par [Combemale, 2008] (Figure 3.1). Selon [Combemale, 2008], un langage de modélisation (Lm) est défini comme suit :

Définition 4

$$(Lm) = \{AS, CS*, Mac*, SD, Mas\}$$

où

- AS est la syntaxe abstraite,
- $CS*$ est la (les) syntaxe(s) concrète(s),
- $Mac*$ est l'ensemble des mappings de la syntaxe abstraite vers la (les) syntaxe(s) concrète(s),
- SD est le domaine sémantique,
- Mas est le mapping de la syntaxe abstraite vers le domaine sémantique.

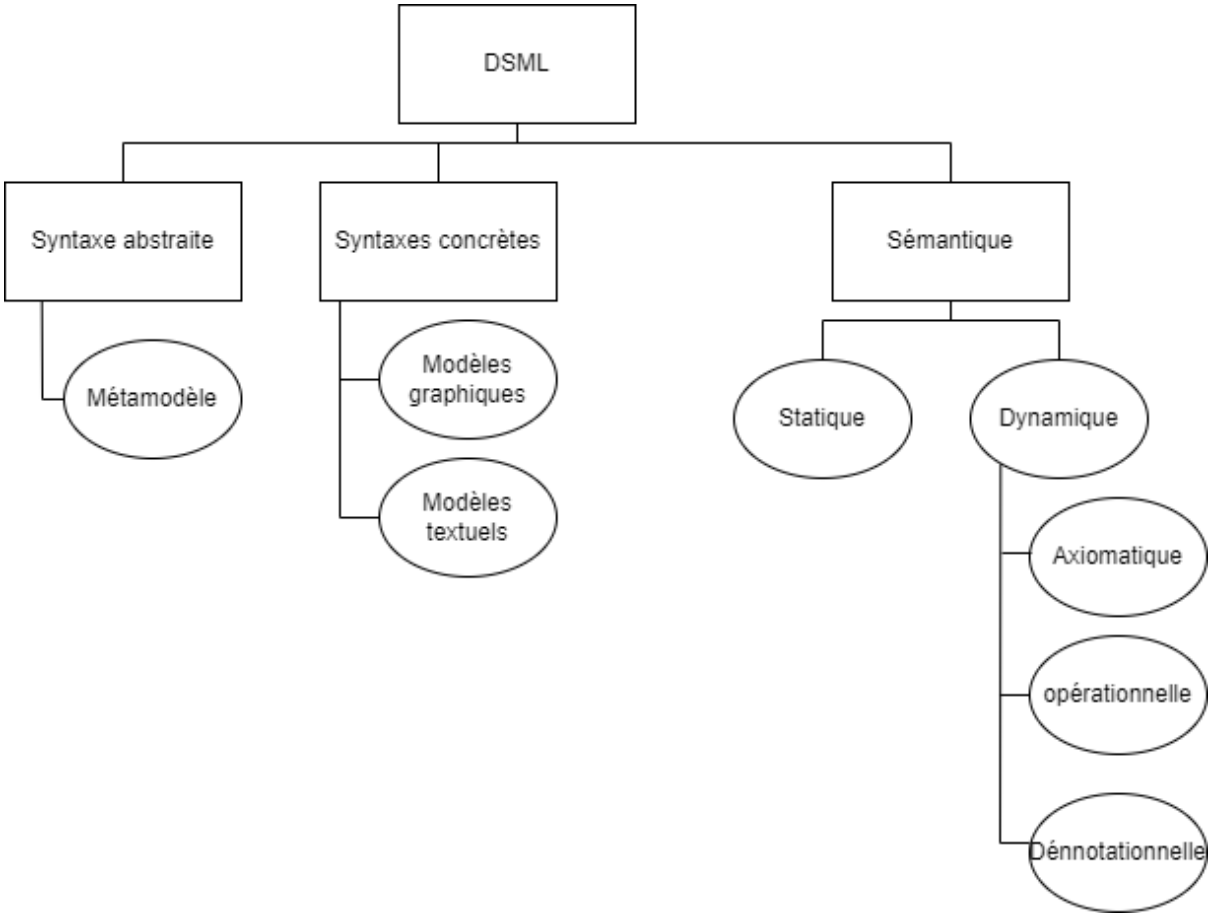


FIGURE 3.1 – Composants d'un langage de modélisation [Smaali, 2017]

Syntaxe abstraite *AS*

La syntaxe abstraite décrit la structure des idées, des relations et des règles grammaticales de DSML. Les relations et les règles grammaticales indiquent comment ces composants peuvent être joints pour générer des modèles légitimes [Greenfield and Short, 2003]. Les concepts reflètent les types d'éléments gérés par DSML, tandis que les relations et les règles grammaticales expliquent comment ces éléments peuvent être combinés pour former des modèles valides. L'applicabilité des concepts spécifiés dépend fortement du niveau d'abstraction visé. L'extraction des caractéristiques essentielles d'un système par rapport à un certain type d'utilisateur et à un domaine spécifique est appelée abstraction d'un système. L'abstraction cache les éléments non essentiels à l'utilisateur tout en lui permettant de s'appuyer sur l'expertise métier pour relever les défis récurrents du domaine.
La syntaxe abstraite est placée au cœur de la description d'un langage de modélisation. Elle est généralement décrite en premier et sert de base pour définir la syntaxe concrète. La syntaxe abstraite (*AS*) d'un langage de modélisation exprime, de manière structurelle, l'ensemble de ses concepts et leurs relations. Elle est décrite à l'aide des langages de méta-modélisation tels que le standard MOF de l'OMG [MOF, 2003]. Ces derniers offrent les concepts et les relations élémentaires qui permettent de décrire un méta-modèle représentant la syntaxe abstraite d'un langage de modélisation à travers des classes contenues par des packages et reliées par différentes relations (associations, compositions ou spécialisations). Chaque classe représente un concept du langage de modélisation dédié, c'est à dire un concept du domaine pour lequel le langage est conçu.

Trouver les abstractions correctes, selon [Tolvanen, 2006], est la composante la plus cruciale du processus de conception d'un DSML. Le degré d'abstraction doit être soigneusement établi afin de ne conserver que les notions qui sont pertinentes pour un DSML. Travailler au niveau d'abstraction approprié est essentiel au succès du DSML car il a un impact direct sur son expressivité. Un DSML avec un niveau d'abstraction trop complexe ne sera pas en mesure de donner des informations adéquates dans ses modèles. D'un autre côté, un niveau d'abstraction trop détaillé peut donner lieu à un DSML difficile à comprendre [Kahlaoui, 2011]. En général, le niveau d'abstraction d'un système est défini par l'objectif pour lequel il est utilisé [Kramer, 2007]. Un modèle est plus facile à abstraire s'il est clair.

Syntaxe concrète $CS*$

La syntaxe concrète fournit aux utilisateurs les notations nécessaires pour exprimer un modèle. Elle peut être soit textuelle, soit graphique (aussi appelée syntaxe visuelle). La syntaxe concrète textuelle est surtout utilisée dans les DSL de programmation, tandis que la syntaxe concrète graphique est surtout utilisée dans les DSML.
Il est possible de proposer plusieurs syntaxes concrètes pour un même langage afin de satisfaire les préférences d'un plus grand nombre d'ingénieurs en logiciel. La syntaxe concrète permet de représenter les concepts de la syntaxe abstraite et créer des modèles qui lui sont conformes. La définition d'une syntaxe concrète consiste à définir un des mappings de $Mac* : AS \leftrightarrow CS$, et permet ainsi d'annoter chaque construction du langage de modélisation définie dans la syntaxe abstraite par une (ou plusieurs) décoration(s) pouvant être manipulée(s) par l'utilisateur du langage.
La syntaxe graphique concrète spécifie les arrangements géométriques des composants graphiques qui seront utilisés pour générer des modèles DSL. Elle comprend une interface utilisateur graphique qui permet de manipuler les idées DSL (spécifier, ajouter, supprimer, connecter, etc.). Un symbole visuel est attribué à chaque élément représentable du langage. La capacité d'une syntaxe visuelle à communiquer des informations de manière intuitive et compréhensible constitue son principal avantage [Clark et al., 2004].

Sémantique du domaine SD

Pour être utile dans le domaine informatique, tout langage (qu'il soit textuel ou visuel ou utilisé pour la programmation, les exigences, la spécification ou la conception) doit s'accompagner de règles rigides qui énoncent clairement les expressions syntaxiques autorisées et donnent une description claire de leur signification.
La spécification explicite et formelle de la sémantique des modèles est particulièrement intéressante, car l'absence de sémantique explicite présente une possibilité d'inadéquation sémantique entre les modèles de conception et les langages de modélisation des outils d'analyse. Bien que ce problème existe dans pratiquement tous les domaines où les DSML sont utilisés, il est plus fréquent dans les domaines où le comportement doit être explicitement représenté, car l'absence de sémantique comportementale explicite entrave fortement le développement d'outils d'analyse et de simulation formels, reléguant les modèles à leur rôle actuel de simples illustrations.
La sémantique d'un langage définit de manière précise et non ambiguë la signification des constructions de ce langage. Elle permet ainsi de donner un sens précis aux programmes construits à partir de celui-ci. On dit qu'une sémantique est formelle lorsqu'elle est exprimée dans un formalisme mathématique. Une sémantique est alors caractérisée par :

— Un domaine sémantique dont les concepts sont compréhensibles et bien définis.

— Un mapping entre les éléments de la syntaxe abstraite et les éléments du domaine sémantique.

On distingue deux types de sémantique : (1) La sémantique statique qui correspond à des propriétés indépendantes de l'exécution ou valables pour toutes les exécutions. Celle-ci est en général vérifiée statiquement lors de la compilation des programmes (et exprime des règles, comme le typage, qui ne peuvent pas être exprimées par la syntaxe) et (2) La sémantique dynamique (ou comportementale) qui permet de décrire le comportement des programmes à l'exécution. Elle peut être opérationnelle, dénotationnelle ou axiomatique.

— **Sémantique axiomatique** : Elle propose une vision déclarative en décrivant l'évolution des caractéristiques d'un élément lors du déroulement d'un programme. Elle peut être exprimée soit à l'aide du langage de méta-modélisation lui-même (ex. les multiplicités) soit en la complétant de contraintes exprimées à l'aide d'un langage comme OCL [Warmer and Kleppe, 2003] (invariant, pré- ou post-condition).

— **Sémantique dénotationnelle** : Elle s'exprime selon une représentation abstraite différente de celle définie dans la syntaxe abstraite du langage considéré. Ce type de sémantique correspond à une sémantique par traduction qui définit un langage par sa transformation vers un autre langage formellement défini et permet ainsi d'utiliser les outils de simulation, de vérification et d'exécution fournis par l'espace technique cible.

— **Sémantique opérationnelle** : Elle permet de décrire le comportement dynamique des constructions d'un langage. Elle vise à exprimer la sémantique comportementale du système modélisé, à l'aide d'un langage d'action, afin de permettre l'exécution des modèles qui lui sont conformes. Elle est exprimée sur des instances des concepts définis dans la syntaxe abstraite.

3.2.2 Domaines d'application

La définition d'un DSML peut se faire en deux parties principales : structure et comportement [Rivera, 2010]. Dans l'MDE, il existe déjà un accord commun sur la manière de spécifier la structure d'un DSML : au moyen d'un méta-modèle. Les concepts structurels définis dans le méta-modèle peuvent être dotés d'une syntaxe concrète en les faisant correspondre à leurs symboles (graphiques ou textuels) correspondants. Les mêmes concepts peuvent également être dotés d'une sémantique en les faisant correspondre à un domaine sémantique.
Il existe plusieurs propositions de DSML tel que :

— DySAL [Smaali, 2017] est un langage spécifique à la modélisation des architectures logicielles dynamiques et a été proposé afin d'offrir un cadre conceptuel et sémantique pour la conception et l'administration d'architectures logicielles dynamiques, tout en surmontant la complexité de leur analyse et de leur vérification, et en soutenant les concepteurs dans le processus de développement

— WF-CML (*Workflow Communication Modeling Language*) [Wu et al., 2011] est un langage spécifique au domaine de la communication du workflow et permet la réalisation rapide d'applications de communication collaborative centrées sur l'utilisateur.

— D-CreEA [Silva and Andrade, 2021] est un DSML pour la création d'analogues éducatifs de cartes. D-CreEA est une approche pour faciliter le processus de création de jeux basés sur les idées fondamentales de la conception de jeux de cartes par le professeur ou le développeur de jeux, qui vise à mettre en œuvre une méthodologie d'apprentissage ludique, avec un contenu éducatif personnalisé.

Les mappings sémantiques peuvent être définis vers plus d'un domaine sémantique, fournissant au DSML des représentations sémantiques alternatives. Chaque domaine sémantique est plus approprié pour représenter et raisonner sur certaines propriétés, et pour mener certains types d'analyse. Par conséquent, ces mappings sémantiques sont très utiles non seulement pour fournir une sémantique précise aux DSML, mais aussi pour pouvoir les simuler et les analyser en

utilisant le cadre logique et sémantique disponible dans le domaine cible.

3.3 Systèmes Réactifs Bigraphiques (BRS)

Les systèmes réactifs bigraphiques (BRS) sont un formalisme conçu pour la modélisation de l'évolution temporelle et spatiale des systèmes. La théorie des bigraphes a été introduite par [Milner, 2009] afin de fournir un modèle graphique intuitif capable de représenter la localité et la connectivité des systèmes distribués. Un système bigraphique réactif est constitué d'un ensemble de bigraphes représentant les états du système et un ensemble de règles de réaction décrivant son évolution. La théorie des bigraphes a été développée avec deux objectifs principaux : (1) être en mesure d'intégrer dans le même formalisme les aspects importants des systèmes ubiquitaires ; et (2) fournir une unification des théories existantes en développant une théorie générale, dans laquelle les différents calculs existants pour la concurrence et la mobilité, tels que le calcul des systèmes communicants [Milner, 1980], le π-calcul [Milner et al., 1992], le calcul ambiant [Milner, 1993] et les réseaux de Petri, peuvent être représentés avec une théorie comportementale uniforme. Cette dernière est obtenue en représentant la dynamique des bigraphes par une définition abstraite de règles de réaction à partir de laquelle un système de transition peut être dérivé pour décrire le comportement des systèmes modélisés. Dans cette section, nous définissons les bigraphes d'une manière informelle avant d'aborder leur définition formelle et les différentes notions qui y sont liées.

Les systèmes réactifs bigraphiques (BRS) sont un formalisme permettant de modéliser l'évolution temporelle et spatiale des systèmes. L'auteur de [Milner, 2009] a proposé la théorie des bigraphes pour donner un modèle graphique simple permettant de représenter la localité et la connexité des systèmes distribués. Un système bigraphique réactif est constitué d'une collection de bigraphes qui reflètent les états du système et d'un ensemble de règles de réaction qui décrivent son évolution. La théorie des BRS a été développé avec deux objectifs principaux : être capable d'intégrer des aspects importants des systèmes ubiquitaires dans un formalisme unique, et fournir une unification des théories existantes en développant une théorie générale dans laquelle les différents calculs existants pour la concurrence et la mobilité, tels que le calcul des systèmes communicants [Milner, 1980], le *pi*-calcul [Milner et al., 1992], le calcul ambiant [Milner, 1993] et les réseaux de Petri, pourraient être représentés avec une théorie comportementale uniforme. Cette dernière est générée en abstrayant la dynamique des bigraphes en un ensemble de règles de réaction, à partir desquelles un système de transition peut être construit pour caractériser le comportement des systèmes simulés. Nous définissons les bigraphes de manière informelle dans cette section avant de passer à leur définition formelle et aux autres notions qui leur sont associées.

Les définitions apportées ici correspondent à celles initialement posées pour les bigraphes classiques. Nous reprenons également, en termes de présentation, l'essentiel des synthèses de ces définitions déjà proposées par [Khebbeb, 2019], [Benzadri, 2016], [Sahli, 2017] et [Cherfia, 2016].

Anatomie des bigraphes

La Figure 3.2 est une représentation d'un bigraphe. Les composants ou entités (physiques ou logiques) qui composent le système sont représentés graphiquement par des nœuds qui peuvent avoir des formes géométriques variées (ovale, carré, rectangle, etc.). Les nœuds d'un bigraphe ont un certain type, appelé contrôle, qui est identifié par un identifiant alphabétique (A, B, C, etc.). La signature d'un bigraphe est constituée de l'ensemble de ses contrôles. L'imbrication hiérarchique entre les différents nœuds du système exprime la répartition spatiale des

nœuds. Si un nœud ne peut pas contenir d'autres nœuds, il est considéré comme atomique. Les connexions non binaires sous forme d'hyper-arcs (liens reliant deux ou plusieurs choses) expriment les interactions entre les nœuds. Ces connexions peuvent représenter des canaux de communication de nœud à nœud ainsi que d'autres interactions abstraites. Un nœud peut avoir zéro ou plusieurs ports, qui sont représentés sur sa membrane par des puces rondes qui indiquent les connexions possibles. Le nombre de ports est le même sur les nœuds ayant les mêmes contrôles. Les rectangles en pointillés sont des régions (également appelées racines) qui décrivent différents éléments du système. Les carrés gris, appelés sites, représentent des parties abstraites du système que le modèle ne traite pas. Les nombres naturels sont utilisés pour numéroter les régions et les sites (en commençant par 0). Un bigraphe peut inclure diverses formes de liens de communication en plus des hyper-arcs, comme les noms internes et externes. Ils indiquent les connexions possibles à d'autres bigraphes représentant des contextes externes.

Un bigraphe dispose d'une interface indiquant ses possibilités d'interactions avec son environnement extérieur. Par exemple, le bigraphe dans la Figure 3.2 possède trois sites, deux régions et deux ensembles de noms externes $\{y_0, y_1, y_2\}$ et noms internes $\{x_0, x_1\}$. La paire $< 3, \{x_0, x_1\} >$ désigne l'interface interne de ce bigraphe, alors que $< 2, \{y_0, y_1, y_2\} >$ représente son interface externe.
Enfin, la localité d'un bigraphe est déterminée par la distribution géographique des nœuds, tandis que sa connectivité est déterminée par les liens. Le graphe de places et le graphe des liens sont définis par ces deux représentations. L'intersection de ces deux graphes est un ensemble partagé de nœuds qui correspondent aux entités du bigraphe.

Le graphe des places est modélisé comme une forêt, et il représente la distribution géographique de divers éléments tout en ignorant leurs relations (voir la Figure 3.3). Le graphe des liens est un hypergraphe qui décrit le réseau de connexion de plusieurs nœuds tout en ignorant leur emplacement géographique (voir la Figure 3.4). Un hyper-arc dans le graphe des liens établit une connexion entre les ports de ces éléments et peut également relier leurs noms internes et externes, tandis qu'un arc dans le graphe des places illustre la relation d'imbrication entre les parties de l'application. Dans le graphe des places, chaque arbre représente une région (dont il est la racine) qui peut contenir des nœuds et des sites. Ces derniers correspondent aux feuilles de l'arbre.

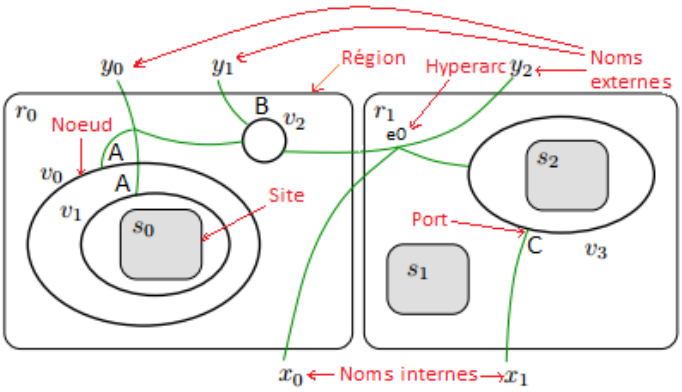


FIGURE 3.2 – Anatomie des bigraphes

3.3.1 Définitions Formelles

Définition 5 (signature.) Une signature de base prend la forme $(\mathcal{K}, ar)$. Elle possède un ensemble $\mathcal{K}$ dont les éléments sont des types de nœuds appelés contrôles, et une fonction $ar : \mathcal{K} \to N$ attribuant une arité, un nombre naturel, à chaque contrôle. La signature est notée $\mathcal{K}$ lorsque l'arité est comprise.

Une signature du bigraphe précédemment présenté dans la Figure 3.2 est donnée par : $\mathcal{K} = A : 1, B : 3, C : 2$.

Définition 6 (graphe de places) . Le graphe de place est défini formellement par :

$$B^P = (V_B, ctrl_B, prnt_B) : m \to n$$

— V_B est un ensemble fini de nœuds.

— $ctrl_B$ est la fonction de contrôle.

— $prnt_B : m \uplus V_B \to V_B \uplus n$ est une fonction de parenté.

— La notation $m \to n$ désigne l'interface interne (m) et l'interface externe (n) du graphe de place B^P.

La Figure 3.3 illustre le graphe de places du bigraphe de la Figure 3.2. Il prend la structure d'une forêt où les racines sont les régions r_0 et r_1 et où les feuilles sont les nœuds ou les sites du bigraphe. Il représente la distribution spatiale de ces entités en ignorant leurs connexions.

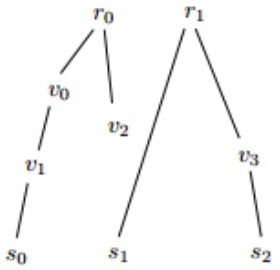


FIGURE 3.3 – Graphe de places

Définition 7 (graphe de liens.) Le graphe de liens est défini formellement par :

$$B^L = (V_B, E_B, ctrl_B, link_B) : X \to Y$$

— V_B est un ensemble fini de nœuds.

— E_B est un ensemble fini d'hyper-arcs.

— $ctrl_B$ est la fonction de contrôle.

— $link_B : X \uplus P \to E_B \uplus Y$ est une fonction qui relie les noms internes X ou les ports P avec les noms externes Y ou les hyper-arcs E.

La Figure 3.4 illustre le graphe de liens du bigraphe de la Figure 3.2. Il prend la forme d'un hypergraphe montrant le réseau de connectivité des différents nœuds en ignorant leur localité. Par exemple : $link_B(p0) = y0$.

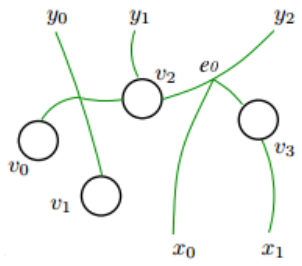


FIGURE 3.4 – Graphe de liens

Définition 8 (bigraphe.) Formellement, un bigraphe B prend la forme suivante :

$$B = (V_B, E_B, ctrl_B, B^P, B^L) : I_B \to J_B$$

— V_B est un ensemble fini de nœuds.

— E_B est un ensemble fini d'hyper-arcs.

— $ctrl_B : V_B \to \mathcal{K}$ est une fonction qui associe à chaque nœud $v_i \in V_B$ un contrôle $c \in \mathcal{K}$ indiquant le nombre de ses ports. La signature $\mathcal{K}$ est un ensemble fini de contrôles.

— $B^P = (V_B, ctrl_B, prnt_B) : m \to n$ est le graphe de places associé à B. m et n représentent le nombre de sites et le nombre de régions. $prnt_B : m \uplus V_B \to V_B \uplus n$ est une fonction de parenté qui associe à chaque nœud ou site son parent hiérarchique.

— $B^L = (V_B, E_B, ctrl_B, link_B) : X \to Y$ est le graphe de liens de B, où $link_B : X \uplus P \to E_B \uplus Y$ 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 B.

— $I =< m, X >$ et $J =< n, Y >$ représentent respectivement les interfaces internes et externes du bigraphe B.

3.3.2 Opérations sur les bigraphes

Les opérations de composition et de produit tensoriel sont utilisées pour générer des bigraphes complexes à partir de bigraphes basiques [Milner, 2008]. Nous donnons des définitions formelles de ces deux opérations sur les bigraphes ainsi que des illustrations de leur fonctionnement.

Composition

Le processus de composition consiste à insérer un bigraphe F dans le contexte d'un autre bigraphe G. Si et seulement si l'interface extérieure de F correspond à l'interface intérieure de G, la composition des deux bigraphes G et F est valide. En hébergeant les régions de F dans les sites de G et en fusionnant les noms externes de F avec les liens de G qui ont des noms internes correspondants, la notation $GcircF$ illustre la composition de ces bigraphes. La définition formelle de l'opération de composition de deux bigraphes est donnée ci-dessous. Pour ce faire, nous devons d'abord définir la composition aux niveaux du graphe de places et du graphe de liens avant de fournir la composition au niveau du bigraphe.

Définition 9 (composition de deux graphes de places.) Si $F^P : k \to m$ et $G^P : m \to n$ sont deux graphes de places avec des supports disjoints, leur composition (voir Figure 3.5) :

$$G^P \circ F^P = (V, ctrl, prnt) : k \to n$$

où

— $V = V_F \uplus V_G$ est l'ensemble des nœuds

— $ctrl = ctrl_F \uplus ctrl_G$ est la fonction de contrôle

— $prnt$ est sa fonction de parenté définie par : Si w est un site ou un nœud dans $G \circ F$ tel que $w \in k \uplus V$ alors :

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

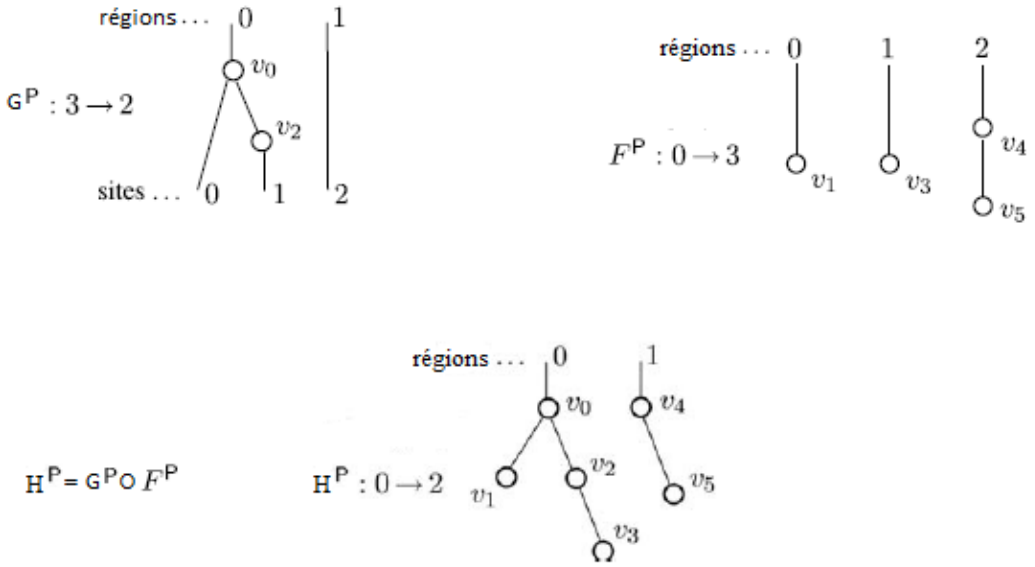


FIGURE 3.5 – Composition de deux graphes de places : $H^P = G^P \circ F^P$

Définition 10 (composition de deux graphes de liens.) Si $F^L : X \to Y$ et $G^L : Y \to Z$ sont deux graphes de liens avec des supports disjoints, leur composition (voir Figure 3.6) :

$$G^L \circ F^L = (V, E, ctrl, link) : X \to Z$$

où

— $V = V_F \uplus V_G$ est l'ensemble de nœuds.

— $E = E_F \uplus E_G$ est un ensemble d'hyper-arcs

— $ctrl = ctrl_F \uplus ctrl_G$ est une fonction de contrôle.

— Sa fonction $link$ est définie par : Si q est un point de $G \circ F$ et $q \in X \uplus P_F \uplus P_G$ alors :

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

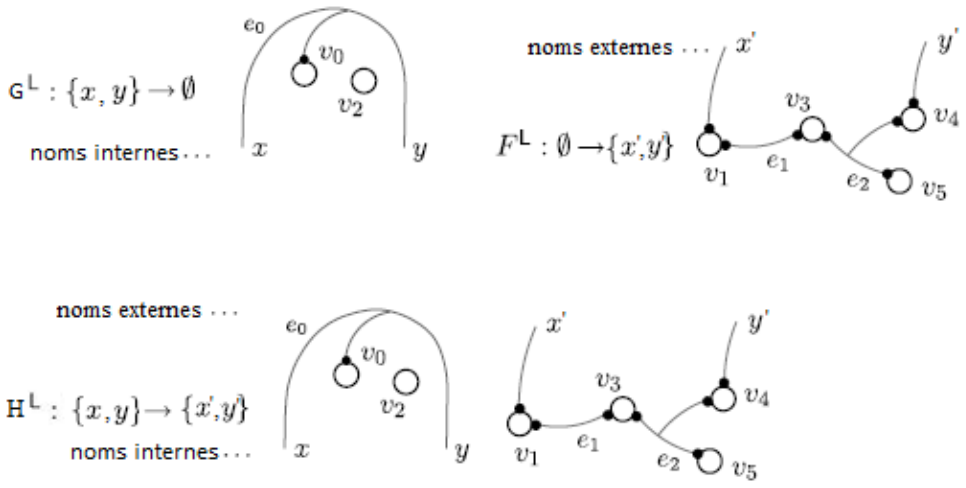


FIGURE 3.6 – Composition de deux graphes de liens : $H^L = G^L \circ F^L$

Définition 11 (composition de deux bigraphes.) Si $F :< k, X >\to< m, Y >$ et $G :< m, Y >\to< n, Z >$ sont deux bigraphes avec des supports disjoints, leur composition est donnée par :

$$G \circ F \stackrel{\text{def}}{=} (V_G \uplus V_F, E_G \uplus E_F, ctrl_G \uplus ctrl_F, G^P \circ F^P, G^L \circ F^L) :< k, X >\to< n, Z >$$

L'exemple suivant (Figure 3.7) montre le bigraphe $H = G \circ F : \epsilon \to< 2, \emptyset >$ résultant de la composition de deux bigraphes $G :< 3, x, y >\to< 2, \emptyset >$ et $F : \epsilon \to< 3, x, y >$ dont les graphes de places et les graphes de liens sont composés dans les Figures 3.5 et 3.6 respectivement.

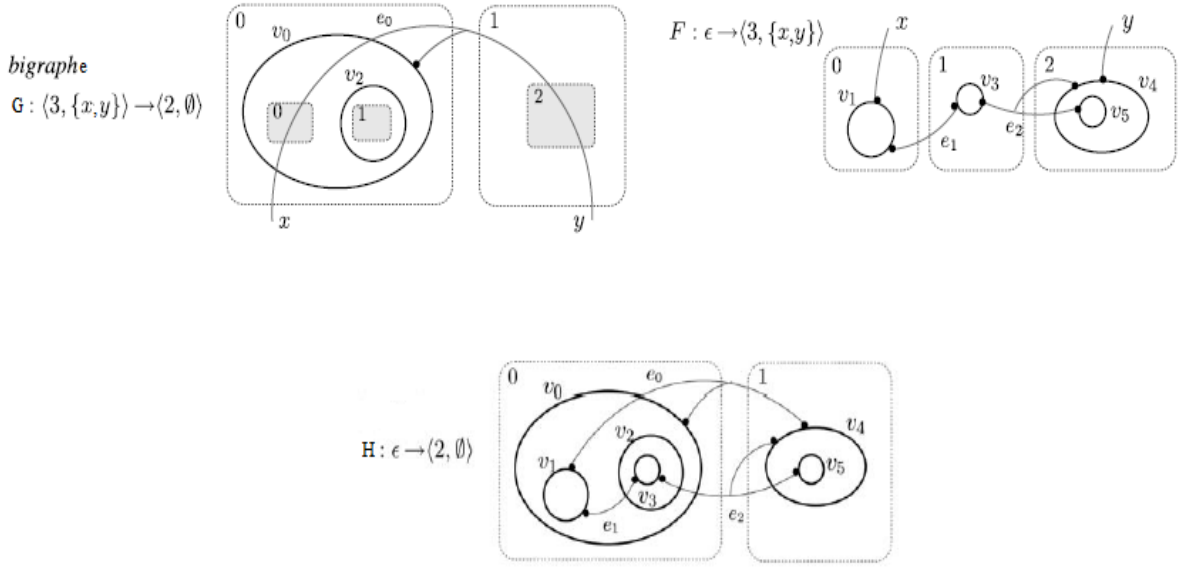


FIGURE 3.7 – Composition de deux bigraphes : $H = G \circ F$

Produit Tensoriel

Le produit tensoriel est une autre opération qui peut être effectuée sur les bigraphes. Elle consiste à relier des liens ouverts communs pour juxtaposer des régions de bigraphes. La composition horizontale est un autre nom pour cette procédure. Nous définissons d'abord le produit tensoriel au niveau des graphes de places et des graphes de liens avant de passer aux bigraphes.

Définition 12 (Produit tensoriel des graphes de places.) Si $G^P = (V_G, ctrl_G, prnt_G) : m_0 \to n_0$ et $F^P = (V_F, ctrl_F, prnt_F) : m_1 \to n_1$ sont deux graphes de places disjoints, leur produit tensoriel $G \otimes F : m_o + m_1 \to n_0 + n_1$ est donné par :

$$G^P \otimes F^P \stackrel{\text{def}}{=} (V_G \uplus V_F, ctrl_G \uplus ctrl_F, prnt_G \uplus prnt'_F)$$

Où $prnt'_F(m_0 + i) = n_0 + j$ lorsque $prnt_F(i) = j$.

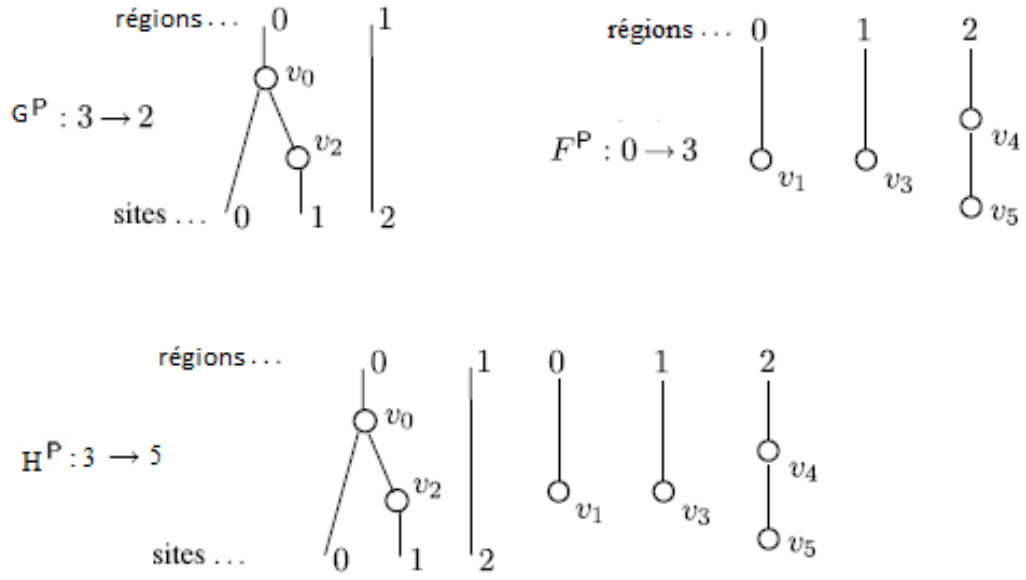


FIGURE 3.8 – Produit tensoriel de deux graphes de places : $H^P = G^P \otimes F^P$

Définition 13 (Produit tensoriel des graphes de liens.) Si $G^L = (V_G, E_G, ctrl_G, prnt_G) : X_G \to Y_G$ et $F^L = (V_F, E_F, ctrl_F, prnt_F) : X_F \to Y_F$ sont deux graphes de places disjoints, leur produit tensoriel $G^L \otimes F^L : X_G \uplus X_F \to Y_G \uplus Y_F$ est donné par :

$$G^L \otimes F^L \stackrel{\text{def}}{=} (V_G \uplus V_F, E_G \uplus E_F, ctrl_G \uplus ctrl_F, link_G \uplus link_F)$$

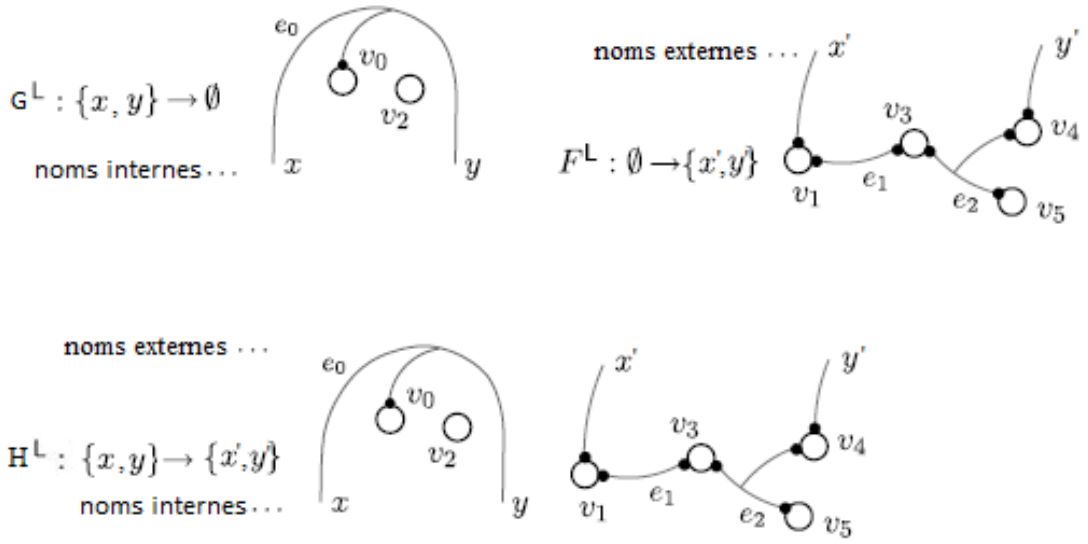


FIGURE 3.9 – Produit tensoriel de deux graphes de liens : $H^L = G^L \otimes F^L$

Définition 14 (Produit tensoriel des bigraphes.) Si $G :< m_G, X_G >\rightarrow< n_G, Y_G >$ et $F :< m_F, X_F >\rightarrow< n_F, Y_F >$ sont deux bigraphes avec des supports disjoints, leur produit tensoriel est donné par :

$$G \otimes F \stackrel{\text{def}}{=} (G^P \otimes F^P, G^L \otimes F^L) :< m_G + m_F, X_G \uplus X_F >\rightarrow< n_G + n_F, Y_G \uplus Y_F >$$

L'exemple suivant (Figure 3.10) montre le bigraphe $A = G \otimes F :< 3, x >\rightarrow< 3, y >$ résultant du produit tensoriel de deux bigraphes $G :< 2, y >\rightarrow< 1, y >$ et $F :< 1, x >\rightarrow< 2, y >$ dont les produits tensoriels des graphes de places et des graphes de liens sont donnés dans les Figures 3.8 et 3.10 respectivement.

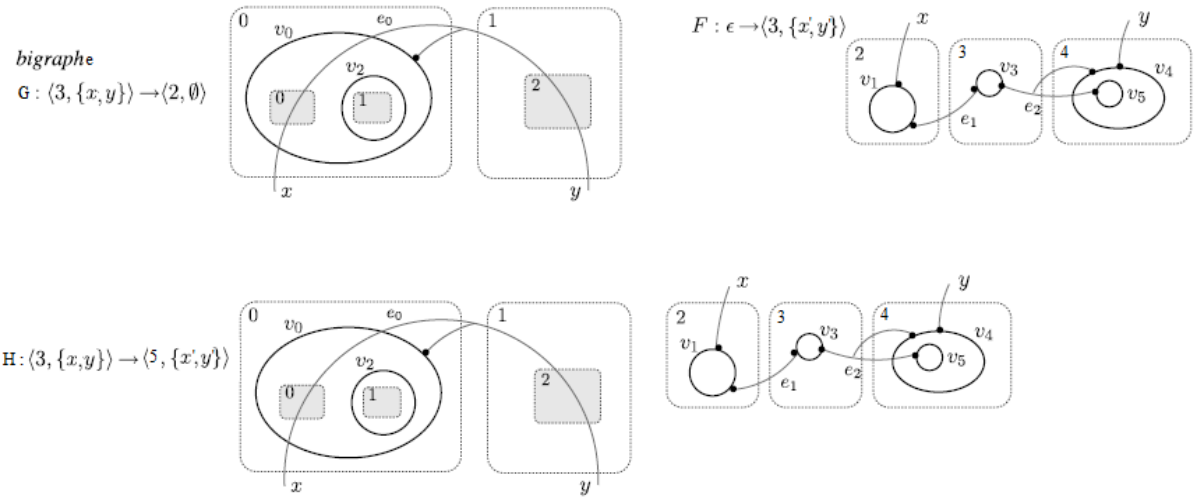


FIGURE 3.10 – Produit tensoriel de deux bigraphes : $H = G \otimes F$

3.3.3 Forme algébrique des bigraphes

Pour exprimer les bigraphes de manière algébrique, ils disposent d'un vocabulaire de concepts algébriques décrivant des opérations autres que la composition et le produit tensoriel. Ce langage comprend un certain nombre d'opérations qui permettent de construire des bigraphes. Les définitions formelles des opérations clés fournies par ce langage, qui seront significatives pour la suite du travail, sont données ici : le produit parallèle (noté *parallel*), la fusion (notée *mid*), et l'imbrication (notée .).

Définition 15 (produit parallèle) .

Produit parallèle des graphes de places : Si $P_i = (V_i, ctrl_i, prnt_i) : m_i \parallel n_i (i = 0, 1)$ sont deux graphes de places disjoints, leur produit parallèle $P_0 \parallel P_1 : m_o + m_1 \to n_0 + n_1$ est donné par :

$$P_0 \parallel P_1 \stackrel{\text{def}}{=} P_0 \otimes P_1$$

Produit parallèle des graphes de liens : Si $L_i = (V_i, E_i, ctrl_i, prnt_i) : X_i \to Y_i (i = 0, 1)$ sont deux graphes de places disjoints et $link_0 \cup link_1$ est une fonction de transformation sur les liens, leur produit parallèle $L_0 \parallel L_1 : X_o \cup X_1 \to Y_0 \cup Y_1$ est donné par :

$$L_0 \parallel L_1 \stackrel{\text{def}}{=} (V_0 \uplus V_1, E_0 \uplus E_1, ctrl_0 \uplus ctrl_1, link_0 \cup link_1)$$

Produit parallèle des bigraphes : Si $B_i :< m_i, X_i > \to < n_i, Y_i >, (i = 0, 1)$ sont deux bigraphes avec des supports disjoints, leur produit parallèle est donné par :

$$B_0 \parallel B_1 \stackrel{\text{def}}{=} ({B_0}^P \parallel {B_1}^P, {B_0}^L \parallel {B_1}^L) :< m_0 + m_1, X_0 \cup X_1 > \to < n_0 + n_1, Y_0 \cup Y_1 >$$

Définition 16 (fusion des bigraphes.) Étant donné deux bigraphes $B_i :< m_i, X_i > \to < n_i, Y_i >$ avec $(i = 0, 1)$, supposons que $B_0 \parallel B_1$ est défini. La fusion de ces deux bigraphes est donnée par :

$$B_0 \mid B_1 \stackrel{\text{def}}{=} (merge_{n_0+n_1} \otimes id_{Y_0 \cup Y_1}) \circ (B_0 \parallel B_1)$$

Avec $B_0 \mid B_1 :< m_0 + m_1, X_0 \cup X_1 > \to < 1, Y_0 \cup Y_1 >$.

Définition 17 (imbrication des bigraphes.) Étant donné deux bigraphes $F : I \to < m, X > etG : m \to < n, Y >$, leur imbrication est définie par :

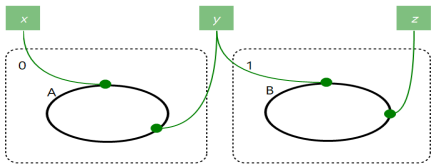
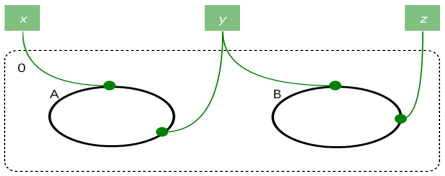
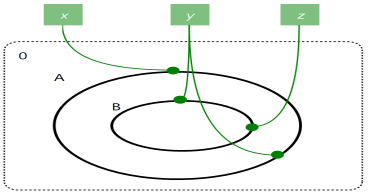
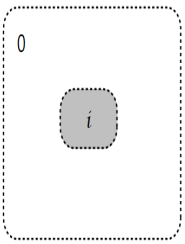
$$G.F \stackrel{\text{def}}{=} (G \parallel id_x) \circ F : I \to < n, X \cup Y >$$

Le Tableau 3.1 [Milner, 2009] contient les opérations de base définies par le langage algébrique des bigraphes.

3.3.4 Logique de typage

Il est souvent important de limiter les possibilités de générer l'ensemble des bigraphes acceptables modélisant un système donné afin d'offrir une description précise de ce système. Ceci est accompli en établissant des restrictions sur la catégorisation des contrôles de ces bigraphes [Milner, 2008]. À cette fin, nous expliquons les concepts de type et de sorting, ainsi que la manière de déclarer correctement de telles contraintes. Nous avons besoin de quelque notations de base : les lettres minuscules sont utilisées pour représenter les différents types : a, b et c. Pour les types globaux, nous écrivons $\widehat{ab}$, ce qui signifie qu'un nœud peut être du type a ou b. Un bigraphe Σ-typé est un bigraphe qui répond à un typage Σ.

TABLE 3.1 – Principaux termes du langage algébrique des bigraphes

Terme	Forme algébrique	Forme graphique
Produit parallèle	$A_x y \mid\mid B_y z$	
Fusion	$A_x y \mid B_y z$	
Imbrication	$A_x y.B_y z$	
Identité(bigraphe élémentaire)	id_i	
Site numéroté i	d_i	

Définition 18 (typage des places.) Un typage de places est donné par un triplé

$$\Sigma_P = (\Theta_P, \mathcal{K}, \Phi_P)$$

où

— Θ_P est un ensemble non-vide de sortes

— $\mathcal{K}$ est une signature Σ_P-typée qui associe une sorte à chaque contrôle et est un ensemble non-vide Φ_P de règles de formation pour Σ_P.

— Φ_P est une propriété d'un ensemble de bigraphes Σ_P-typés qui est satisfaite par les identités, les symétries, et préservée par la composition, le produit tensoriel et le produit parallèle.

Exemple

Afin d'illustrer le concept de typage (sorting), nous considérons l'exemple d'un bigraphe S modélisant un switch (Figure 3.11). Un switch est un dispositif qui achemine les paquets entre différents réseaux.
Il peut savoir où les paquets entrants doivent aller en notant l'adresse source sur ces paquets. Á chaque fois qu'un switch voit un paquet avec une adresse source a dans un réseau n, il enregistre de façon interne l'adresse a avec le réseau n. Ensuite, les paquets venant de a sont transmis dans n. Dans la mise en réseau PC/LAN, les switch implémentés ainsi sont appelés "learning bridges". Nous pouvons modéliser un learning bridge en utilisant les systèmes réactifs bigraphiques.
Dans cet exemple [Debois and Damgaard, 2005], nous définissons un learning bridge relié à deux réseaux et nous lui associons ensuite les règles de réactions appropriées pour décrire son comportement. Nous utilisons :

— Des contrôles N (networks) avec une arité 1, M (machines) d'arité 2 et P (paquets) d'arité 2. Le seul port du réseau N, le relie à la machine qui est enregistrée dans ce réseau.

— Les deux ports du paquet P le lient à sa source et à la machine cible respectivement. Les deux ports de la machine M relient, d'un coté, la machine à sa localisation enregistrée si elle existe, et de l'autre, tous les paquets référençant la machine.

— Notre switch va connecter deux réseaux (représentés dans la Figure ci-dessous par N_1 et N_2) avec, initialement, une machine dans chacun des réseaux. On suppose que la deuxième machine $M2$ a une adresse connue du switch (port n_2 relié au nœud N_2) .

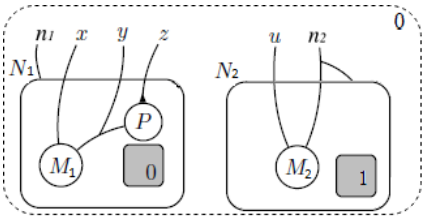


FIGURE 3.11 – Exemple illustratif d'un switch

Si la machine M1 envoie un paquet P à la machine M2, le paquet est d'abord relié au port de M1 (voir Figure 3.11. Ensuite, le switch lie la machine M1 à son réseau pour lui attribuer sa localisation (port n_1). Donc, en envoyant un paquet de M1 à M2, il sera envoyé à l'adresse connue de M2 et l'adresse de M1 sera mise à jour. Les sites sont utilisés comme une abstraction pour le reste des machines et paquets d'un même réseau.
Le typage des nœuds impliqué dans cet exemple est défini par $\Sigma_P = (\Theta_P, \mathcal{K}, \Phi_P)$ où :

— L'ensemble de sortes $\Theta_P = \{n, m, p\}$
— La signature $\mathcal{K} = \{(N_1, n), (N_2, n), (M_1, m), (M_2, m), (P, p)\}$
— Les règles de formations $\Phi_P = \{\Phi_0, \Phi_1\}$ où
  — Φ_0 : Tout nœud de sorte p doit être relié à un nœud de sorte m
  — Φ_1 : Tout nœud de sorte m doit avoir un noeud de sorte n comme parent

3.3.5 Dynamique des bigraphes

La sémantique dynamique des bigraphes explique comment les bigraphes peuvent se reconfigurer en termes de places et de liens. Les règles de réaction, similaires aux règles de réécriture de graphes, sont utilisées pour exprimer cette dynamique. [Milner, 2009].
Une paire <redex, reactum> décrit une règle de réaction $R \to R'$. Le redex est une pré-condition de réaction qui spécifie quelle section du bigraphe sera modifiée (ou réécrite). Le reactum est une post-condition qui définit l'élément de remplacement après que la réaction ait terminé son chargement (réécriture). Les reconfigurations peuvent être divisées en deux catégories : (1) la reconfiguration de place par l'ajout, la suppression ou le déplacement de nœuds (2) la reconfiguration de lien (former ou supprimer un lien entre deux nœuds ou un nœud et un nom externe/interne).

Définition 19 (règle de réaction.) Une règle de réaction est notée : $R = (R : m \to J, R' : m \to J)$ et elle est généralement écrite sous la forme $R \to R'$, où $R : m \to J$ est le redex et $R' : m \to J$ est le reactum.

Définition 20 (système réactif bigraphique.) Un système réactif bigraphique (BRS) comprend une paire (B, R), où B est un ensemble de bigraphes et R un ensemble de règles de réaction définies sur B.

Exemple

Revenons à l'exemple du switch. Nous donnons maintenant les règles de réactions appliquées pour modéliser son comportement (Figures 3.12, 3.13, 3.14 et 3.15).

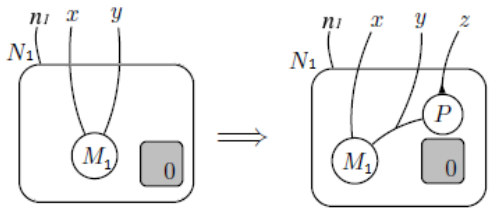


FIGURE 3.12 – **R1** : Pour qu'une machine envoie un paquet à une autre, il doit être créé.

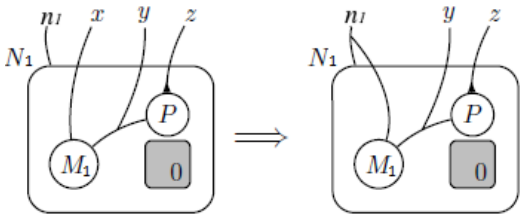


FIGURE 3.13 – **R2** : Si une machine envoie un paquet, le switch associera l'adresse source trouvée dans le paquet avec son réseau.

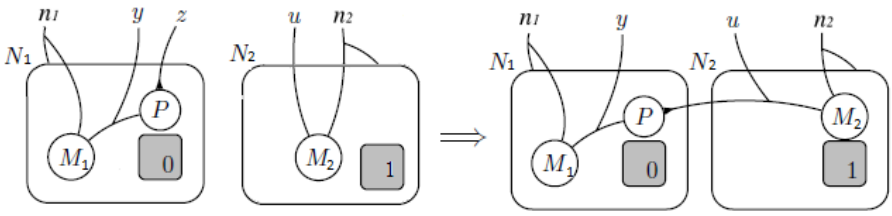


FIGURE 3.14 – **R3** : Le switch associe le paquet à la machine cible.

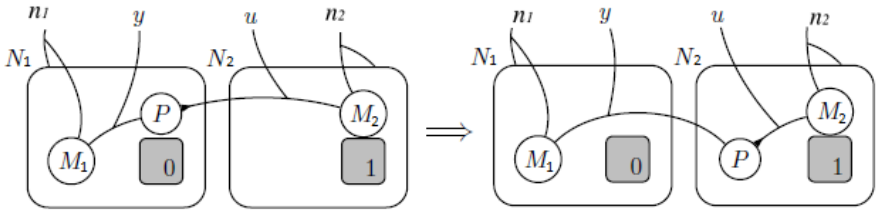


FIGURE 3.15 – **R4** : Le switch peut déplacer un paquet vers le réseau associé à l'adresse cible des paquets.

3.4 BiAgents

Le formalisme des bigraphes présente des points forts vu qu'il modélise un système à travers ses structures, les interactions entre ces dernières et aussi à travers l'étude de sa dynamique. Cependant, afin de répondre au mieux aux besoins des chercheurs, il existe plusieurs extensions des bigraphes. En effet, les auteurs de [Krivine et al., 2008] ont proposé les bigraphes stochastiques en ajoutant aux différentes règles de réaction des probabilités. Aussi, les "binding" bigraphes [Damgaard and Birkedal, 2006], les bigraphes dirigés [Grohmann and Miculan, 2007], les bigraphes avec partage [Sevegnani and Calder, 2015] et enfin les Biagents [Pereira et al., 2012] forment d'autres extensions des bigraphes qui ont été utilisées pour modéliser différents types de systèmes.

Dans cette section, nous présentons les biagents : une extension des bigraphes avec des agents abstraits [Pereira et al., 2012]. Les agents apportent un coté réfléchi et intelligent au concept des bigraphes. Dans un système modélisé par les biagents, sa structure est modélisée par la partie bigraphe et sa gestion mentale par la partie agent. Cette gestion consiste en l'observation et le contrôle des entités physiques.

3.4.1 Structures physique et virtuelle

Les auteurs de [Pereira et al., 2012] définissent un biagent comme suit : $\mathcal{A} \bullet \mathcal{B}$.
$\mathcal{B}$ représente la structure physique d'un biagent et $\mathcal{A}$ sa structure virtuelle telle que présentée par la suite.

Définition 21 La structure physique d'un biagent est définie comme étant un tuple

$$\mathcal{B} = (\mathbb{B}, \mathcal{R}, \mathbb{U}, B_0, F)$$

— $\mathbb{B}$ est l'espace des bigraphes.

— $\mathcal{R}$ est l'ensemble des règles de réaction.

— $\mathbb{U}$ est l'espace de contrôle tel que $\mathbb{U} \subseteq \mathcal{R} \times V_{\mathbb{B}}$, $V_{\mathbb{B}}$ étant l'ensemble des nœuds du bigraphe.

— B_0 est le bigraphe initial.

— Avant de définir F, $dec3$ est une fonction qui donne une décomposition valide d'un bigraphe en trois bigraphes tels que $dec3(B) = (B', B'', B''')$ de façon à ce que $B = B' \circ B'' \circ B'''$. V_R est l'ensemble des nœuds du redex et $V_{R'}$ est l'ensemble des nœuds du reactum. F est la fonction de transition qui, à partir du bigraphe courant et d'une action de contrôle, donne un nouveau bigraphe : $F : \mathbb{B} \times \mathbb{U} \to \mathbb{B}$. Elle est définie comme suit en considérant C comme étant le contexte de R' et d comme étant les paramètres de R' :

$$F(B, (R \to R', h)) = C \circ R' \circ d$$

si $\exists$ $dec3$ tel que $dec3(B) = (C, R, d)$ et $h \in V_R$ et $h \in V_{R'}$.
Si non, F est indéfinie.

Définition 22 La structure virtuelle d'un biagent est définie par un ensemble $\mathcal{A}$ composé de plusieurs agents de type a. Chaque agent a est défini par le tuple

$$a = (\mathcal{O}, \mathcal{U}, host_0, obs, ctr, mgrt)$$

— $\mathcal{O}$ est l'espace d'observation de l'agent tel que $\mathcal{O} \subseteq \mathbb{B}$.

— $\mathcal{U}$ est l'espace de contrôle tel que $\mathcal{U} = \mathcal{R}_a \times V_{\mathbb{B}}$ ($\mathcal{R}_a$ représente un ensemble de règles de réactions.

— $host_0 \in V_{\mathbb{B}}$ est le nœud qui abrite l'agent initialement.

— obs est la fonction d'observation, $obs : \mathbb{B} \times V_{\mathbb{B}} \to \mathcal{O}$

— ctr est la fonction de contrôle avec laquelle nous pouvons effectuer une action, $ctr : \mathcal{O} \to \mathcal{U}$.

— $mgrt$ est la fonction de migration qui, étant donnés le nœud courant et une observation, elle détermine le nœud suivant, $mgrt : V_{\mathbb{B}} \times \mathcal{O} \to V_{\mathbb{B}}$.

3.4.2 Trace formelle d'un Biagent

Dans un système réactif bigraphique, une trace [Perrone et al., 2011] est une séquence de bigraphes $\langle a1, a2, ... \rangle$ tels que pour chaque a_i et a_{i+1} dans la séquence, il y a une règle de réaction $a_i \to a_{i+1}$. S'il existe deux traces s et t et le dernier élément de s est le redex d'une règle de réaction dont le reactum est le premier élément de t, la trace composite existe et commence par tous les éléments de s suivis de tous les éléments de t, dans ce cas t est une extension de s. On note $Tr(A)$ l'ensemble de toutes les traces pour un BRS A donné.

Dans un Biagent, la trace globale est définie par l'application des règles de réaction et actions correspondantes aux agents, sur les couples (B,H) où B est un bigraphe et H les nœuds hébergeant les agents Chaque agent peut avoir sa propre trace appelée *Projection*.

Définition 23 (Projection) On représente l'ensemble des agents à travers une projection t^a d'une trace t définie comme suit :

$$t^a = (p_0^a, H_0^a) \prec (p_1^a, H_1^a) \prec ...$$

où chaque p_i^a est une projection du bigraphe B_i qui retient seulement le nœud dans lequel se trouve l'hôte H_i^a.

Exemple

Dans ce qui suit, nous présentons l'exemple que nous avons utilisé dans la Section 3.3 (l'exemple du switch) afin de montrer l'apport de l'agent dans la modélisation de cet exemple. Nous remarquons la nécessité d'utiliser des agents pour effectuer les opérations définies par des règles de réaction (créer un paquet, lier une machine à son réseau, lier un paquet à sa machine cible, déplacer un paquet d'un réseau à un autre). Ceci s'illustre par le besoin d'entités autonomes capables de s'adapter aux changements et d'agir selon la situation. Grâce au formalisme de biagent, nous pouvons modéliser l'aspect virtuel tel qu'il ne pouvait l'être par la simple utilisation des bigraphes.

Nous définissons la structure physique de ce système comme suit :

— $\mathbb{B}$ étant l'ensemble des bigraphes représentant les états par lesquels passe le système notés B_0, B_1, B_2, B_3, B_4 dans la Figure 3.17

— $\mathcal{R}$ l'ensemble des règles de réaction présentées dans la Section 3.3 que nous notons : R_1, R_2, R_3, R_4.

— $\mathbb{U} \subseteq \mathcal{R} \times V_{\mathbb{B}}$ qui est l'ensemble de couples de règles de réaction et de nœuds sachant que $V_{\mathbb{B}} = \{N_1, N_2, M_1, M_2, P\}$. Cette structure servira à déterminer à quel nœud précis appliquer la règle de réaction.

— B_0 est présenté dans la Figure 3.16

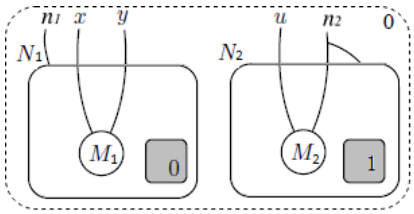


FIGURE 3.16 – État initial du système switch

— $F(B_0, (R_1 \to R'_1, v_1)) = 0 \circ R'_1 \circ 0$ et $v_1 \in \{N_1, M_1\}$

— $F(B_1, (R_2 \to R'_2, v_2)) = 0 \circ R'_2 \circ 0$ et $v_2 \in \{N_1, M_1, P\}$

— $F(B_2, (R_3 \to R'_3, v_2)) = 0 \circ R'_3 \circ 0$

— $F(B_2, (R_3 \to R'_3, v_3)) = 0 \circ R'_3 \circ 1$ et $v_3 \in \{N_2, M_2\}$

— $F(B_3, (R_4 \to R'_4, v_2)) = 0 \circ R'_4 \circ 0$

— $F(B_3, (R_4 \to R'_4, v_3)) = 0 \circ R'_4 \circ 1$

Cette fonction sert à déterminer le bigraphe obtenu après l'application d'une règle de réaction sur un nœud précis dans un bigraphe. Pour les autres (B, u), F est indéfinie et $u \notin \mathbb{U}$.
Dans cet exemple, nous définissons 3 agents : N^{Ag}, M^{Ag} et P^{Ag}

— N^{Ag} représenté par une étoile à 4 pointes dans la Figure 3.17 représente le processus responsable de la liaison de chaque machine au réseau correspondant.

— M^{Ag} représenté par une étoile à 8 pointes dans la Figure 3.17 est le programme responsable de la création d'un paquet, de sa liaison à sa machine source et de sa liaison à sa machine cible.

— P^{Ag} représenté par une étoile à 5 pointes dans la Figure 3.17 est le protocole de transmission.

Formellement, les trois agents N^{Ag}, M^{Ag} et P^{Ag} sont définis comme suit :

TABLE 3.2 – Définition des agents N^{Ag}, M^{Ag} et P^{Ag}

N^{Ag}	$\mathcal{O}^{N^{Ag}} = \{B_1\}$
	$\mathcal{U}^{N^{Ag}} = \{(\text{lier réseau}, N1)\}$
	$host_0^{N^{Ag}} = N_1$
	$obs^{N^{Ag}}(B_1, N_1)$
	$ctr^{N^{Ag}}(B_1) = (\text{lier réseau}, N_1)$
	$mgrt^{N^{Ag}}$ l'agent N^{Ag}ne migre pas
M^{Ag}	$\mathcal{O}^{M^{Ag}} = \{B_0, B_2, B_3\}$
	$\mathcal{U}^{M^{Ag}} = \{(\text{créer}, M1), (\text{lier paquet}, M2)\}$
	$host_0^{M^{Ag}} = M_1$
	$obs^{M^{Ag}}(B_0, M_1)$, $obs^{M^{Ag}}(B_2, M_1)$, $obs^{M^{Ag}}(B_3, M_2)$
	$ctr^{M^{Ag}}(B_0) = (\text{créer}, M_1)$, $ctr^{M^{Ag}}(B_3) = (\text{lier paquet}, M_2)$
	$mgrt^{M^{Ag}}(M_2, B_2)$
P^{Ag}	$\mathcal{O}^{P^{Ag}} = \{B_4\}$
	$\mathcal{U}^{P^{Ag}} = \{(\text{déplacer}, P)\}$
	$host_0^{P^{Ag}} = P$
	$obs^{P^{Ag}}(B_4, P)$
	$ctr^{P^{Ag}}(B_4) = (\text{déplacer}, P)$
	$mgrt^{P^{Ag}}$ l'agent P^{Ag} ne migre pas.

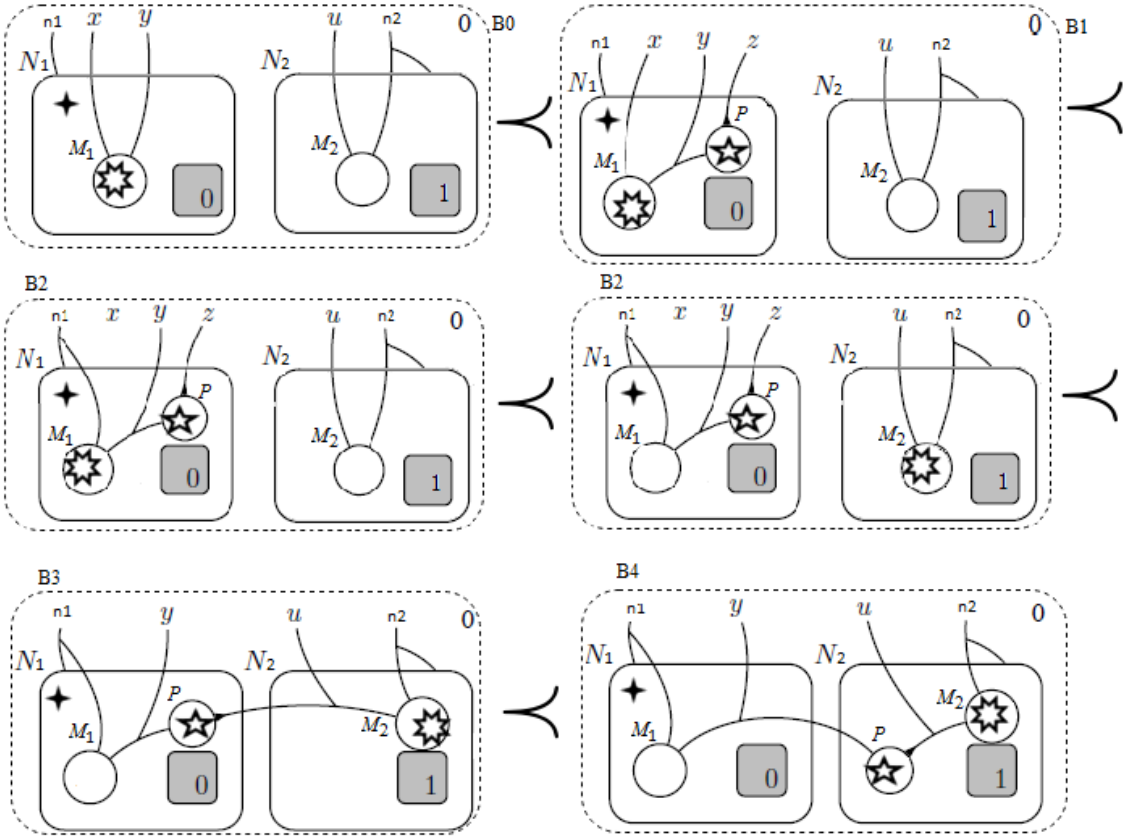


FIGURE 3.17 – Trace du biagent $\{ N^{Ag}, M^{Ag}, P^{Ag}\}\bullet\mathcal{B}$

Pour mieux comprendre le déroulement des événements, nous renommons les règles R_1, R_2, R_3 et R_4 respectivement par créer, lier réseau, lier paquet et se déplacer.

Nous définissons une trace globale du déroulement des événements :

$$t = (B_0, H) \overset{\text{créer}}{\prec} (B_1, H) \overset{\text{lier réseau}}{\prec} (B_2, H) \overset{\text{lier paquet}}{\prec} (B_3, H) \overset{\text{déplacer}}{\prec} (B_4, H)$$

avec $H(N^{Ag}) = \{N_1, N_2\}$ et $H(M^{Ag}) = \{M_1, M_2\}$ et $H(P^{Ag}) = \{P\}$.

Cette trace se traduit, pour chaque agent, comme suit :

TABLE 3.3 – Trace pour chacun des agents.

Agent	Trace t projetée
$t^{N^{Ag}}$	$(0, N_1) \overset{\text{créer}}{\prec} (0, N_1) \overset{\text{lier réseau}}{\prec} (0, N_1) \overset{\text{mgrt}}{\prec} (0, N_1) \overset{\text{lier paquet}}{\prec} (0, N_1) \overset{\text{déplacer}}{\prec} (0, N_1)$
$t^{M^{Ag}}$	$(N_1, M_1) \overset{\text{créer}}{\prec} (N_1, M_1) \overset{\text{lier réseau}}{\prec} (N_1, M_1) \overset{\text{mgrt}}{\prec} (N_2, M_2) \overset{\text{lier paquet}}{\prec} (N_2, M_2) \overset{\text{déplacer}}{\prec} (N_2, M_2)$
$t^{P^{Ag}}$	$(\varepsilon, \varepsilon) \overset{\text{créer}}{\prec} (N_1, P) \overset{\text{lier réseau}}{\prec} (N_1, P) \overset{\text{mgrt}}{\prec} (N_1, P) \overset{\text{lier paquet}}{\prec} (N_1, P) \overset{\text{déplacer}}{\prec} (N_2, P)$

ε étant l'élément vide.

La Figure 3.17 représente la trace du biagent $\{ N^{Ag}, M^{Ag}, P^{Ag}\}\bullet\mathcal{B}$. Cette trace signifie que :

— À l'état initial, le système est modélisé par B_0.
— L'agent M^{Ag}va, en premier lieu, créer un paquet. On obtient B_1 ;
— Ensuite N^{Ag} va relier la machine M_1 au réseau N_1. (B_2) ;
— Après cela, l'agent M^{Ag} va migrer vers M_2 (B_2) pour pouvoir relier ensuite le paquet P à la machine cible M_2. (B_3)
— L'agent P^{Ag} pourra donc permettre au paquet de se déplacer du réseau N_1 au réseau N_2 (B_4).

N^{Ag} ne migre pas dans cet exemple. Mais si nous avions eu plusieurs réseaux dont les machines ne sont pas reliées, l'agent N^{Ag} aurait migré vers ces réseaux pour effectuer la liaison. Les biagents permettent d'associer au système deux vues distinctes : structurelle (les bigraphes) et virtuelle (les agents). Les agents spécifient le programme du switch, tandis que les bigraphes forment la représentation interne des réseaux gérés par le switch. À travers cet exemple illustratif, nous pouvons noter que l'utilisation des biagents a amélioré la prise en charge de la dynamique des bigraphes. En effet, les règles de réaction utilisées séparément définissent l'évolution du système switch d'un état à un autre de manière indéterministe (choix de la règle à appliquer). En enrichissant les règles avec la notion d'observation et de contrôle, cela simplifie considérablement le nombre d'états à considérer durant l'exécution du système.

Le fait que l'agent doit observer pour déduire la prochaine action à effectuer réduit l'indéterminisme dans le choix des règles applicables dans une situation donnée. L'observation obligatoire avant chaque action permet à l'agent de réfléchir avant d'agir et de s'assurer d'effectuer des actions sensées dans leur contexte.

Par exemple, la règle "se déplacer" (d'un point de vue bigraphe) ne peut s'effectuer que si la machine source a un emplacement connu (reliée à un réseau). Après avoir introduit les agents, si la machine n'a pas un emplacement connu, l'agent N^{Ag} se déplace vers le réseau contenant la machine et relie cette dernière à son réseau. Si nous sommes donc au niveau de B_3, et M_1 n'est pas relié à N_1, niveau bigraphe, le paquet ne peut pas se déplacer. Niveau biagent, l'agent N^{Ag} a pour mission de lier la machine émettrice d'un paquet au réseau dans lequel elle se trouve pour résoudre ce problème.

3.5 Langage Maude

Maude [Clavel et al., 2007] [Clavel et al., 2016] [Clavel et al., 2002] [Clavel et al., 1996] est un langage de haut niveau basé sur les concepts mathématiques de la réécriture et de la logique équationnelle. Maude est un langage de spécification formelle pour les systèmes informatiques, en particulier les systèmes concurrents et distribués. Les équations et les types de données intégrés nativement expliquent les caractéristiques structurelles et statiques d'un système dans Maude (entiers relatifs, nombres à virgule flottante, etc.). Les règles de réécriture sont utilisées pour définir les éléments comportementaux et dynamiques. Maude génère des spécifications exécutables qui peuvent être simulées et vérifiées formellement. Dans Maude, la vérification formelle est accomplie à l'aide d'un interpréteur et d'une variété de techniques et d'outils de vérification et d'analyse formelles, comme le model-checker LTL et l'approche de vérification par invariants. Dans cette section, nous introduisons Maude, ses caractéristiques et son fonctionnement.
Le langage Maude est caractérisé par [Clavel et al., 2016] :

La simplicité. La spécification de base de Maude est simple et directe. Les expressions du langage Maude (équations et règles de réécriture) ont une interprétation simple : réécrire le côté gauche de l'expression dans son côté droit.

La performance. En une seule seconde, le système Maude peut effectuer un nombre important de réécritures. Cela permet une simulation et une analyse assez efficaces des nombreux chemins d'exécution alternatifs dans une spécification.
L'expressivité. Par le biais de modules fonctionnels et de systèmes, le langage Maude permet une modélisation compréhensible de systèmes déterministes et non déterministes. Les équations sont utilisées pour réaliser le calcul déterministe dans les modules fonctionnels. Le calcul non déterministe, quant à lui, est représenté dans les modules système par des règles de réécriture.

3.5.1 Syntaxe et notations

Les modules sont les éléments de base de la spécification ou de la programmation dans le langage Maude. Les modules fonctionnels, qui mettent en œuvre des théories équationnelles, et les modules de système, qui mettent en œuvre des théories de réécriture qui décrivent le comportement dynamique d'un système, sont les deux formes de base des modules. Les notations de base et les caractéristiques syntaxiques des divers modules Maude sont présentées dans cette section.

Modules fonctionnels

Les types de données et les opérations nécessaires aux équations sont définis dans ces modules (conditionnels et inconditionnels), les sortes, les sous-sortes et les opérations. Les mots-clés suivants sont utilisés dans Maude pour définir les attributs des opérations : `ctor` (constructeur), `assoc` (associative), `comm` (commutative), `id` (élément neutre) et `prec` (degré de précédence ou de priorité). 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 .
```

Dans un module fonctionnel, les déclarations et les expressions peuvent être :
Des importations d'autres modules. Les primitives de protection, d'inclusion et d'extension peuvent être utilisées pour importer des modules fonctionnels prédéfinis (`protecting`, `including`

ou `extending`). Déclarations de sortes et de sous-sortes. Le terme `sort` est utilisé pour spécifier des sortes (types de données), et le terme `subsort` est utilisé pour définir des sous-sortes. Si nous devons définir plusieurs sortes ou sous-sortes, nous utilisons le mot-clé `sorts` ou `subsorts`

Des déclarations d'opérations. Le mot clé `op` est utilisé pour spécifier les opérations qui agissent sur les sortes et les sous-sortes.

Des déclarations d'équations. Il existe deux types d'équations : conditionnelles et inconditionnelles. Dans ce qui suit, la syntaxe d'une équation inconditionnelle :

```
eq <Terme-1>=<Terme-2>[<Attributs>] .
```

Dans l'équation Term-1 = Term-2, les deux mots Term-1 et Term-2 doivent être du même type. Toutes les variables de Term-2 (partie droite de l'équation) doivent apparaître dans Term-1 pour que l'équation soit exécutable (partie gauche). Dans ce qui suit, un exemple d'équation conditionnelle :

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

Il existe trois façons différentes d'écrire les conditions d'une équation conditionnelle :

— Équations ordinaires de la forme t = t',

— Équations de correspondance (matching) t : = t'

— [Bool], équivalent à **t = true**.

Un module fonctionnel est donc défini par $(\Sigma, E \cup A)$ où Σ est la signature du module qui spécifie les sortes, les sous-sortes, les types et les opérateurs. E est la collection d'équations (éventuellement conditionnelles), et A est l'ensemble des attributs équationnels déclarés pour certains opérateurs tels que *assoc* pour associatif, *comm* pour commutatif, etc.

Modules systèmes

Maude possède un module système qui implémente la théorie de la réécriture. Il y a des catégories, des types, des opérateurs, des déclarations d'équation et des règles de réécriture dans une théorie de réécriture, qui peut être conditionnelle ou non. Par conséquent, chaque théorie de réécriture a une théorie équationnelle correspondante. Les règles de réécriture représentent des transitions concurrentes locales à l'intérieur d'un module de système. Si la composante gauche d'une règle correspond à un élément de l'état du système et que la condition de la règle est remplie, la règle peut être exécutée. La transition de la règle peut être utilisée, et le fragment d'état détecté sera transformé en l'instance équivalente du côté droit.

Dans ces modules, la réécriture est basée sur la simplification équationnelle : pour obtenir la forme réduite et finale, dite canonique, d'une expression, les règles de réécriture sont appliquées de manière répétée jusqu'à ce que plus aucune réduction ne soit possible. Par conséquent, chaque classe d'équivalence a une seule représentation canonique, qui peut être déterminée par une simplification équationnelle alimentée par des règles de réécriture.
Un module système Maude est déclaré selon la syntaxe :

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

Le terme `mod` est utilisé pour définir un module de système dans Maude, suivi du nom du module. Les importations d'autres modules fonctionnels ou de système, les déclarations de types

ou de sous-types, les déclarations d'opérations, les énoncés d'équations et les déclarations de règles de réécriture conditionnelles ou inconditionnelles sont tous des exemples de déclarations et d'expressions dans un module fonctionnel. La syntaxe suivante est utilisée pour spécifier des règles de réécriture conditionnelles et inconditionnelles dans un module système :

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

Les mots <Term-1> et <Term-2> sont tous deux de même sorte contenant des variables du même type. L'étiquette de la règle de réécriture est <Label> ; elle peut être supprimée. Les expressions de réécriture qui testent la possibilité de réécrire des termes algébriques peuvent se trouver dans les conditions d'une règle de réécriture conditionnelle <Condition-k>.

Un module système implémente donc une théorie de réécriture sous la forme d'un triplet $(\Sigma, E \cup A, R)$, où $(\Sigma, E \cup A)$ est la partie théorie équationnelle d'appartenance du module, et R est une collection de règles de réécriture éventuellement conditionnelles.

Exemple

Nous illustrons un problème simple en utilisant Maude. Dans ce problème [Martí-Oliet et al., 2009], un berger doit transporter jusqu'à l'autre côté d'une rivière un loup, une chèvre et un chou. Le berger n'a qu'un seul bateau avec deux sièges pour lui et un autre voyageur. Le problème dans cet exemple est que, en l'absence du berger `s`, le loup w mangerait la chèvre `g`, et la chèvre mangerait le chou `c`. L'opération `Initial` se réfère à la situation initiale, où l'on suppose que tous les éléments se trouvent sur la rive gauche de la rivière grâce à une équation. Les règles de réécriture représentent la façon dont le loup et la chèvre mangent et les différentes façons de traverser la rivière.

Dans cet exemple, $(\Sigma_{RC}, E_{RC} \cup A_{RC}, R_{RC})$ est défini comme suit : Σ_{RC} = (des sortes telles que `Side` , des opérations telles que `left : -> Side` , et des variables telles que `S : Side`). E_{RC} = des equations telles que `eq change ( left ) = right`. A_{RC} = des attributs équationnels tels que `assoc` et `comm`. R_{RC} = des règles de réécritures telles que

```
rl [ shepherd - alone ] : s ( S ) = > s ( change ( S ) ) .
```

```
   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
```

3.5.2 Langage de stratégie

Le passage de la logique équationnelle à la logique de réécriture permet d'utiliser des règles de réécriture pour spécifier le comportement dynamique de nombreuses sortes de systèmes. Cependant, comme ces règles sont déclaratives et non déterministes, contrôler leur exécution et s'assurer que le processus de réécriture ne prend pas de mauvaises directions est extrêmement difficile. Le langage de stratégie [Eker et al., 2007], [Martí-Oliet et al., 2009] qui est lié au langage Maude, fournit un cadre pour concevoir et exprimer diverses stratégies et plans d'exécution qui régulent la façon dont les termes sont réécrits dans Maude et fixent des limites à l'exécution des règles. L'architecture du langage est construite sur une distinction claire entre les règles de réécriture définies dans les modules du système et les expressions de stratégie exprimées indépendamment dans les modules de stratégie. En raison de cette séparation, plusieurs modules de stratégie peuvent être construits pour réguler les réécritures d'un module système particulier de nombreuses façons. La syntaxe de déclaration des modules de stratégie est la suivante :

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

Le terme `smod` est utilisé pour déclarer un module de stratégie, qui est ensuite suivi du nom du module. Les variables, les importations à partir d'autres modules et les déclarations de stratégie sont autant de déclarations et d'expressions dans un module de stratégie. Ce qui suit est une définition d'une stratégie exprimée dans un module de stratégie :

```
strat s : @ type-de-terme . sd s := Expression .
```

Le terme `strat` est utilisé pour déclarer l'identifiant d'une stratégie. Après le symbole `@`, on définit le type d'expression sur lequel cette technique peut être utilisée. Le terme `sd` est utilisé pour commencer la définition d'une stratégie, qui est suivie de l'identifiant de la stratégie et de l'expression de la stratégie. Les définitions de stratégies conditionnelles sont possibles, auquel cas le mot-clé `csd` est utilisé pour spécifier la stratégie. Dans Maude, une stratégie est définie comme une opération qui, lorsqu'elle est appliquée à un terme donné, crée une collection de termes (résultat). Les stratégies de base peuvent être utilisées pour définir des stratégies. Elles comprennent l'application d'une règle de réécriture (identifiée par le nom de la règle) sur un mot donné, permettant aux variables de la règle d'être instanciées par des substituts avant l'application de la règle. De nombreux opérateurs et expressions régulières de combinaisons, tels que l'union (`|`), la concaténation (`;`), l'itération (`*` et `+`), et d'autres sortes d'opérateurs, sont utilisés pour combiner les stratégies fondamentales afin de construire des stratégies plus compliquées. L'opérateur d'union (`|`) est à la fois associatif et commutatif, mais l'opérateur de concaténation (`;`) est seulement associatif. Le Tableau 3.4 [Eker et al., 2007] résume les nombreux opérateurs utilisés pour générer des stratégies exprimant le langage de stratégie Maude.

Exemple

Pour résoudre le problème du berger [Martí-Oliet et al., 2009], les deux premières équations de RIVER-CROSSING sont utilisées pour forcer le système Maude à les appliquer avant toute règle de réécriture. Outre le fait que cette solution introduit un problème de cohérence qu'il fallait résoudre, elle change également la sémantique du problème. Une autre solution consiste

TABLE 3.4 – Langage de stratégies de Maude [Sahli, 2017]

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

à utiliser trois stratégies définies dans un module de stratégie.

La stratégie *eating* contrôle la façon de manger pour la chèvre et le loup (contrôle l'exécution des règles $wolf - eats$ et $goat - eats$). La stratégie *oneCross* n'applique une des autres règles qu'une seule fois. Enfin, la stratégie $allCE$ retourne tous les états accessibles. Sur la base de la première définition $(\Sigma_{RC}, E_{RC} \cup A_{RC}, R_{RC})$, R_{RC} est remplacé par $S(R, SM)$ où $R = R_{RC}$ et SM = des sortes de stratégies telles que `strat eating : @ Group` et des définitions de stratégies telles que

```
sd eating := ( wolf - eats | goat - eats ) !. 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
```

3.5.3 Analyse et vérification formelle

Un module de système spécifie une théorie de réécriture pour exprimer un modèle mathématique qui peut être exécuté. Par conséquent, Maude utilise une variété de techniques de vérification formelle pour étudier le comportement des systèmes représentés. Par conséquent, des modules de vérification formelle peuvent être définis dans Maude pour décrire certains attributs relatifs au comportement d'un système. Cela permet de vérifier si certaines qualités sont garanties au moment de l'exécution et, si elles ne le sont pas, de proposer des contre-exemples démontrant leur violation.
La vérification d'invariants, les prouveurs de théorèmes, le model-checker LTL, l'analyse de terminaison et l'analyse de cohérence ne sont que quelques-uns des outils et techniques disponibles dans le système Maude pour l'étude formelle du comportement des systèmes. En raison de son importance pour le travail fourni dans cette thèse, nous allons examiner en profondeur le model-checker LTL.
Le model-checker Maude utilise la logique temporelle linéaire (LTL). Pour la définition et la vérification de nombreuses sortes de propriétés, cette approche de vérification formelle est riche, simple et fréquemment utilisée. Les attributs de sécurité (rien de mauvais ne se produit jamais),

de vivacité (quelque chose de bien finit par se produire) et d'équité, par exemple (quelque chose de bon est répété à l'infini). Dans Maude, nous pouvons utiliser un module fonctionnel pour spécifier les caractéristiques et les formules LTL requises. Les formules de la logique temporelle linéaire propositionnelle LTL(AP), par exemple, sont définies nativement dans Maude comme suit :

— True : $T \in LTL(AP)$.

— Propositions élémentaires : si $p \in AP$, alors $p \in LTL(AP)$.

— Opérateur Next : si $\varphi \in LTL(AP)$ alors $\bigcirc\varphi \in LTL(AP)$.

— Opérateur Until : si $\varphi, \psi \in LTL(AP)$, alors $\varphi \cup \psi \in LTL(AP)$.

— Opérateurs logiques : si $\varphi, \psi \in LTL(AP)$, alors les formules : $\neg\varphi$, et $\varphi \vee \psi$ appartiennent 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 $= \neg T$.

— Conjunction : $\varphi \wedge \psi = \neg((\neg\varphi) \vee (\neg\psi))$

— Implication : $\varphi \rightarrow \psi = \neg(\neg\varphi) \vee \psi$.

Quelques opérateurs temporels supplémentaires :

— Eventuellement (Eventually) : $<> \varphi = T \cup \varphi$.

— Toujours (Henceforth) : $[]\varphi = \neg <> \neg\varphi$.

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

Un concepteur peut définir ses propres propositions dans AP en utilisant des équations conditionnelles de la manière suivante :

ceq <terme> \mid= <φ_i> = true if <condition> == true .

Si la condition est vraie, le terme est un fragment du système indiquant un état à étiqueter avec la proposition φ (définie comme une équation du module fonctionnel est satisfaite). Les axiomes (propositions présumées vraies) peuvent également être définis à l'aide d'équations inconditionnelles. Les formules LTL propositionnelles sont ensuite définies à l'aide d'équations comme suit :

eq <formule> = [φ_1 | OP1] [φ_2] OP2 φ_3 ... OPm φ_n.

Où `formule` est une propriété de LTL(AP) à définir, `OP1,2...m` sont des opérateurs de la logique LTL et $\varphi_1, \varphi_2, ...\varphi_n$ sont des propositions élémentaires de AP.

Le model checker de Maude. Les paramètres suivants sont utilisés pour exécuter le model-checker LTL Maude :

(1) L'état E du système comme état initial pour la vérification ;

(2) Une formule F en LTL(AP) qui doit être examinée. Si aucune violation de F n'est identifiée tout au long de l'exécution du système (par le déclenchement des règles de réécriture) à partir de l'état E, la primitive "True" sera affichée. Le model-checker produira un contre-exemple démontrant le chemin d'exécution qui a conduit à la violation si F est violé pendant

l'exécution. Il est intéressant de noter que le fragment (`terme`) décrivant E doit appartenir à l'union des sortes de termes utilisés pour décrire les propositions atomiques. Les auteurs dans [rub, 2019] ont proposé l'extension du model-checker de Maude afin de prendre en charge la notion de stratégies. Un troisième paramètre est ajouté à l'exécution

(3) La stratégie à prendre en compte pour l'exécution. L'affichage du résultat du model checking se fait de la même manière que pour Maude classique.

3.6 Conclusion

Nous avons présenté dans ce chapitre les fondements formels utilisés dans notre étude. Tout d'abord, nous avons établi une définition générale du DSML, y compris ses caractéristiques syntaxiques et sémantiques. Nous avons ensuite présenté les systèmes réactifs bigraphiques. L'anatomie des bigraphes, ainsi que leur forme graphique et leurs définitions formelles, ont été discutées. Nous avons également démontré comment effectuer les opérations les plus courantes sur les bigraphes. Ensuite, nous avons abordé leur forme algébrique ainsi que la théorie de typage des bigraphes. La nature dynamique des systèmes réactifs bigraphiques a également été abordée. Nous avons ensuite présenté l'extension Biagents des bigraphes avec ses parties structurelle et virtuelle. Nous avons définis le concept de trace formelle d'un Biagent à travers des exemples. Nous avons également présenté plusieurs concepts associés au langage Maude ainsi que son langage de stratégie en plus des outils d'analyse formelle associés.

Deuxième partie

Contributions

Chapitre 4

Fog-DSML : Un Langage de Modélisation Spécifique aux Systèmes Fog

Sommaire

4.1 Introduction

Le Fog computing offre une faible latence, un fonctionnement fiable et élimine le besoin de connexions persistantes au Cloud ce qui fait que ce paradigme entre en adéquation avec la résolution de nombreux nouveaux scénarios d'aujourd'hui attirant ainsi l'attention de la communauté scientifique. Plusieurs travaux de recherche s'intéressent donc à la maîtrise de ces systèmes et à la réduction de leur complexité de développement à travers plusieurs techniques dont la modélisation formelle. Dans la Section 4.2 nous recensons plusieurs travaux liés à la modélisation formelle des systèmes Fog avec lesquels nous positionnons notre travail de recherche. Ensuite, nous introduisons notre approche dans la Section 4.3, un langage de modélisation spécifique aux systèmes Fog basé sur une architecture multi-couches d'une part et le formalisme des BiAgents* d'autre part. Nous présentons la syntaxe abstraite du langage Fog-DSML sous forme de diagrammes UML permettant de définir les différentes entités d'un système Fog. Nous définissons aussi la syntaxe concrète de ce langage sous forme d'architecture multi-couches. L'architecture que nous proposons est notée CLA*4*Fog pour (*Cross Layer Architecture for Fog computing*). En nous basant sur la norme ISO/IEC/IEEE 42010 :2011, nous présentons les différents points de vues et vues de l'architecture.

4.2 Classification des approches de conception des systèmes Fog

Dans ce qui suit, nous nous intéressons aux approches de conception des systèmes Fog avec lesquelles nous pouvons positionner nos travaux de recherche. Les approches considérées dans notre étude de l'existant visent la conception d'un système Fog e le décrivant de manière assez abstraite afin de réduire sa complexité. Certaines de ces approches sont de base formelles (STE et AP) et d'autres semi-formelles (MDE et SMA) telles qu'illustrées dans la Figure 4.1.

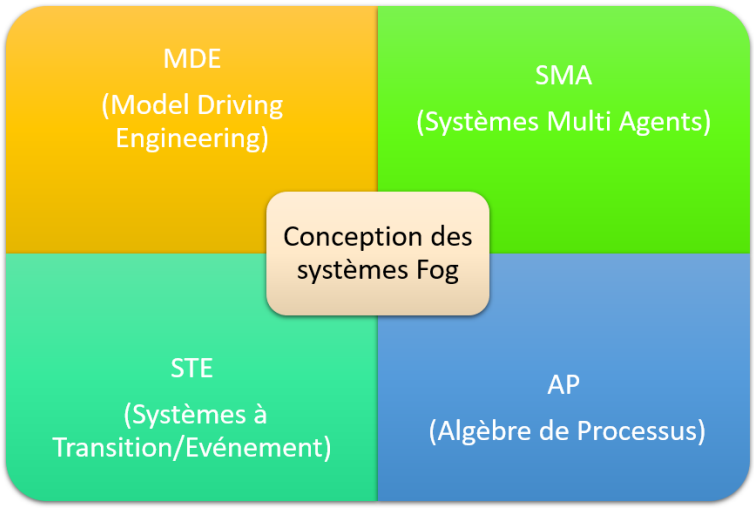


FIGURE 4.1 – Classes des travaux connexes

Les travaux identifiés dans cette classification de base sur un certain nombre de concepts fondamentaux tels que la technique MDE, Les SMA, les STE et les algèbres de processus (voir Figure 4.1). Chacun de ces concepts oriente et motive l'approche adoptée pour spécifier et développer un système Fog. Nous rapprochons ces travaux à travers un ensemble de critères, jugés pertinents pour positionner notre contribution qui consiste en la proposition d'un DSML, et qui sont évoqués comme suit :

— Les concepts de base sur lesquels les différents travaux se basent,

— La prise en charge de la structure complexe des systèmes Fog en termes de localité et de connectivité des éléments impliqués,

— La spécification de l'évolution des systèmes Fog, tout en considérant l'aspect intelligent de ces systèmes,

— La vérification et de la validation des approches introduites mettant l'accent sur la technique de vérification formelle utilisée ainsi que l'outil de simulation.

— La généricité des efforts de modélisation des systèmes Fog ou la spécificité selon un cas d'étude précis.

Nous regroupons dans le Tableau 4.1 l'ensemble des travaux étudiés en les analysant selon les critère susmentionnés (le symbole ✓ signifie que les travaux en question répondent au critère mentionné et le symbole × veut dire que ces travaux ne prennent pas en charge ce critère).

4.2.1 Approches basées techniques MDE

Un langage de modélisation et de déploiement de pipelines analytiques basés sur des données est proposé par les auteurs de [Díaz-de Arcaya et al., 2020] en plus des environnements Edge et Fog qui utilisent des conditions basées sur la plage des données historiques afin de prédire les défaillances du matériel tout en surveillant en permanence le système déployé. Ce langage vise à intégrer toutes les particularités et contraintes des modèles de déploiement en fournissant des schémas pleinement conformes pour la mise en œuvre de charges de travail analytique de données. L'adoption de PADL (un langage de définition et de déploiement de pipeline analytique) permet la mise en œuvre de ces pipelines de manière reproductible et résiliente. En outre, PADL est en mesure d'utiliser pleinement les avantages des couches Edge computing et Fog computing. Le langage a été validée avec un pipeline d'analyse déployé sur un environnement de calcul Edge pour résoudre un cas d'utilisation de l'industrie 4.0. Les résultats ouvrent la voie à l'adoption généralisée de ce langage lors du déploiement.

En utilisant les Langages Spécifiques au Domaine, les auteurs de [Barriga et al., 2021] introduisent SimulateIoT, une approche de développement pilotée par un modèle visant à définir, générer du code et déployer la simulation de systèmes IoT au moyen d'un métamodèle de domaine, d'une syntaxe graphique concrète et d'un modèle de transformation de texte. L'environnement de simulation IoT généré à partir de chaque modèle comprend les capteurs, les actionneurs, les nœuds Fog, les nœuds Cloud et les caractéristiques analytiques, qui sont déployés comme microservices et conteneurs Docker et où les éléments sont connectés en utilisant le protocole de communication publication-abonnement. De plus, deux études de cas, axées sur la construction intelligente et l'agriculture, sont présentées pour montrer l'expressivité de la simulation.

TABLE 4.1 – Approches de conception des systèmes Fog.

Travaux de recherche	Concept		Description			Simulation /Analyse	Généricité
			Physique	Virtuelle	Comportementale		
[Díaz-de Arcaya et al., 2020]	MDE	ADL	✓	×	×	×	✓
[Barriga et al., 2021]		DSL	✓	×	✓	SimulateIoT	✓
[Petrovic and Tosic, 2020]		Ontologie	✓	×	✓	Environnement de simulation proposé/ Analyse sémantique	✓
[Mutlag et al., 2021]	SMA	Agents dédiés	×	✓	✓	iFogSim	Système de santé
[Abbas et al., 2018]		Agents Jade	×	✓	✓	JADE	Ville intelligente
[Gharbi et al., 2021]		Blockchain	×	✓	✓	Outil de simulation non mentionné	✓
[Murtaza et al., 2020]	STE	Automates temporisés	×	✓	✓	iFogSim/ UPPAL model-checker	✓
[Zahra et al., 2017]		HLPN	✓	×	×	Z3 for SMT	✓
[Sahli et al., 2019]		BRS	✓	×	✓	×	✓
[Benzadri et al., 2021]		CA-BRS	✓	✓	✓	Maude	Raffinerie de pétrole et gaz
[Khebbeb et al., 2020]		Maude	✓	×	✓	Maude	✓
[Chen et al., 2020]	AP	PEPA	✓	×	×	Simulation stochastique	✓
[Guo et al., 2020]		ROR	×	×	✓	ROR	✓
[Roig et al., 2021a]		ACP	✓	×	✓	Code Promela/ model-checker Spin	✓
[Roig et al., 2021b]		ACP	×	×	✓	Vérification algébrique	✓
Notre approche	MDE STE	DSML Biagents*	✓	✓	✓	Maude stratégie/ Model-checker	✓

Dans leur étude, les auteurs de [Petrovic and Tosic, 2020] définissent une composante structurelle, appelée SMADA-fog, qui génère automatiquement du code pour la gestion de l'infrastructure Fog. Les auteurs utilisent un méta-modèle qui définit la structure et les contraintes d'une famille de modèles sur deux aspects différents du déploiement dans une architecture Fog computing. Cette modélisation est au cœur du SMADA-fog car elle permet à la fois l'annotation du modèle de déploiement dans l'ontologie, ainsi que l'utilisation de stratégies d'adaptation par la génération de code pour s'adapter aux changements dans l'environnement. SMADA-fog est particulièrement intéressant par rapport à son utilisation des structures sémantiques pour la génération automatisée de code. Cette opération est similaire aux principes de déploiement de service en raison du code personnalisable qui peut être déployé. Ces travaux proposent une approche très approfondie de la mise en œuvre démontrant les avantages des structures sémantiques pour la représentation complexe des données. Cependant, SMADA-fog est limité dans la définition de sa structure sémantique et son modèle d'optimisation puisque nous ne savons pas s'il est interopérable avec d'autres ontologies. De plus, cette structure repose uniquement sur l'optimisation linéaire et les capacités de raisonnement des ontologies pour effectuer son déploiement de code.

4.2.2 Approches basées SMA

En utilisant les SMA afin de modéliser virtuellement le comportement d'un système critique de gestion des tâches de santé, les auteurs de [Mutlag et al., 2021] ont utilisé quatre types d'agents, agents personnels, agents maîtres des personnels, agents nœuds Fog et agents maîtres des nœuds Fog, afin de diriger des tâches critiques en hiérarchisant et en programmant ces derniers. Ils ont simulé le comportement d'un système de santé critique en utilisant iFogSim avec des ensembles de données ECG réels.

Dans [Abbas et al., 2018], les auteurs proposent un framework de ville durable intelligente qui intègre le Cloud computing, le Fog et des agents JADE dans une même architecture. Des serveurs Fog distribués sont utilisés pour héberger des services critiques pour la localisation et la coordination des agents qui agissent comme des courtiers, des entremetteurs et des facilitateurs. Les dispositifs Edge sont modélisés comme des agents autonomes qui interagissent entre eux et avec les unités Fog. Les serveurs Fog agissent comme des entités intermédiaires pour les activités avec les agents des Edge inférieurs et avec les services et ressources dans le Cloud de niveau supérieur. Les résultats de la simulation (à l'aide de DF [1] et de NOSHAPE [2]) montrent que la solution proposée offre des performances élevées en termes de temps de réponse des services, si un modèle d'autorégulation est utilisé. Le système autorégulé, en combinaison avec le Fog computing, a été introduit pour faire progresser les applications des villes intelligentes en tant que modèle technologique afin de développer l'évolutivité et la complexité des futures villes intelligentes. Cependant, ce framework est considéré comme étant peu sécurisé et peu fiable.

Une architecture integrée sécurisée Fog Cloud-IoT basée sur des SMA et la technologie Blockchain est proposée par les auteurs de [Gharbi et al., 2021] pour protéger la confidentialité des données et fournir un accès sûr à faible latence à de vastes volumes de données, ainsi que pour augmenter l'efficacité de la prise de décision. Le système virtuel a démontré son efficacité dans la prise de décision, l'exécution distribuée et la réponse en cas d'intrusion sans intervention de l'utilisateur.Ils ont évalué l'efficacité de l'architecture qu'ils proposent et l'ont comparée aux

1. JADE implémente un agent de facilitation d'annuaire DF (*Directory Facilitator*) spécifié par FIPA. Le DF est souvent comparé à l'annuaire téléphonique des "Pages jaunes". Les agents qui souhaitent faire de la publicité pour leurs services s'inscrivent auprès du DF.
2. Un modèle organisationnel des systèmes Multi Agents

modèles existants. Le résultat de l'étude montre que la réduction du temps de réponse améliore l'efficacité.

4.2.3 Approches basées STE (Systèmes à Transition/Événement)

Dans cette classe d'approches, nous nous limitons aux travaux de littérature utilisant les formalismes suivants : les réseaux de Petri, les automates, les bigraphes et la logique de réécriture, afin de spécifier et analyser formellement les systèmes Fog.

En se basant sur les automates temporisés, les auteurs de [Murtaza et al., 2020] ont proposé une approche adaptative pour améliorer le temps de traitement des tâches dans les systèmes Fog-Cloud en utilisant l'apprentissage et le raisonnement basés respectivement sur des règles et des cas. Ils utilisent des algorithmes de planification afin de modéliser virtuellement un système Fog et de représenter son comportement à l'aide d'automates temporels Uppaal. Ils améliorent les paramètres d'exécution de la tâche en utilisant certaines tâches dispatchées aléatoirement vers le Cloud et le Fog. À l'aide de leurs résultats, ils distribuent chaque tâche restante vers le nœud Fog le plus proche ou vers le Cloud lorsque ce dernier n'est pas libre. Ils simulent cette exécution en utilisant iFogSim [3] et vérifient leur proposition en utilisant le model-checker Uppaal.

Les auteurs de [Zahra et al., 2017] ont considéré différents protocoles de sécurité dans le réseau Fog et IoT pour la sécurité des données. ils ont utilisé les réseaux de Petri à Haut niveau (High-Level Petri Net : HLPN) pour modéliser physiquement leur système et Z3 le solutionneur de contraintes pour la solution automatisée SMT (Satisfaisabilité Modulo Théories) et la vérification formelle. Ils ont choisi le protocole de sécurité Shibboleth après son analyse dans l'environnement Cloud-IoT. Compte tenu du bien-fondé des propriétés de ce protocole, les auteurs ont ajouté de nouvelles couches de Shibboleth parmi les clients et les nœuds Fog. Leur but étant de sécuriser l'authentification, l'autorisation, la vie privée des utilisateurs et l'accès aux données auprès des fournisseurs de services.

Le formalisme des (BRS) a aidé les auteurs de [Sahli et al., 2019] à modéliser un système Fog auto-adaptatif. Ils modélisent la structure et le comportement d'un système Fog avec des règles de réaction pour décrire la distribution physique des éléments de ce type de systèmes ainsi que pour modéliser leur évolution.

L'extension des BRS par des agents (CA-BRS : Control Agents Bigraphical systems) a permi aux auteurs de [Benzadri et al., 2021] de proposer une architecture pour le systèmes Fog d'une raffinerie de pétrole et de gaz. Ils ont utilisé les bigraphes pour modéliser la vue physique de ce système et les agents de contrôle pour la modélisation virtuelle. Le comportement de ce système est spécifié par les règles de réaction bigraphiques et le comportement des agents de contrôle. Le modèle résultant est simulé à l'aide d'un outil basé sur Maude pour CA-BRS.

En utilisant Maude et sa logique de réécriture, les auteurs de [Khebbeb et al., 2020] modélisent un système Fog en se concentrant sur l'aspect d'orchestration des ressources. Ils proposent un ensemble de prédicats et d'opérations atomiques adaptables. Les prédicats ont été appliqués pour déterminer les situations d'approvisionnement des ressources des deux niveaux. Les opérations ont été utilisées pour déterminer quels mécanismes d'adaptation devraient être utilisés. Ils ont également spécifié un ensemble de propriétés temporelles qui devraient être respectées afin d'analyser les comportements de l'orchestration et d'assurer leur qualité.

3. Le simulateur qui permet de modéliser et de simuler le Fog computing afin d'évaluer les politiques de gestion et d'ordonnancement des ressources en périphérie et dans le Cloud

4.2.4 Approches basées Algèbre de processus

Le langage formel PEPA (algèbre du processus d'évaluation des performances) a servi de base pour les auteurs de [Chen et al., 2020] afin de modéliser un système hybride-Cloud-Fog (HCF). Avec l'aide du modèle de composition PEPA, ils ont spécifié la composition physique des systèmes Fog. Ils ont conçu la gestion virtuelle de ce type de systèmes en utilisant le schéma d'ordonnancement PEPA. PEPA contribue à la modélisation de systèmes à grande échelle en raison de ses caractéristiques de composabilité et d'abstraction. Ils ont évalué la performance de leur cadre par simulation stochastique et la méthode d'analyse numérique fluid flow.

Dans [Guo et al., 2020], les auteurs ont proposé un schéma d'authentification basé sur une extension du modèle ROR (Real Or Random) pour les systèmes Fog. Les auteurs proposent un modèle dit "léger" (*lightweight*) et sûr. Ils ont vérifié la sécurité du modèle proposé à travers des analyses de sécurité formelles et informelles. Le modèle s'est avéré sûr et léger. Cependant, cette approche ne montre pas bien le comportement du système à travers une simulation.

Les auteurs de [Roig et al., 2021a] proposent d'entreprendre deux types de modèles différents, comme un modèle algébrique, avec une algèbre de processus appelée ACP (Algebra of Communicating Processes) et un modèle de codage avec un langage de modélisation appelé Promela. Les deux formalismes ont été utilisées pour construire des modèles considérant une infrastructure Edge avec une sauvegarde Cloud, qui a été étendue avec l'ajout de nœuds Fog supplémentaires en appliquant les techniques de vérification appropriées. Plus précisément, une conception générique d'Edge computing a été spécifiée de manière algébrique avec ACP, suivie de sa vérification algébrique correspondante, tandis qu'elle a également été spécifiée au moyen du code Promela, qui a été vérifié au moyen du model-checker Spin.

En utilisant aussi le formalisme ACP, les auteurs de [Roig et al., 2021b] proposent une modélisation algébrique formelle d'un scénario générique Fog avec un support Cloud pour des dispositifs IoT en mouvement. Une topologie générique a été introduite, intégrant une gamme de relais sans fil, un orchestrateur et un ensemble d'hôtes avec une topologie de commutation les connectant tous ensemble, considérant le Cloud comme système de secours.
L'étude s'est concentrée sur la modélisation du mouvement des machines virtuelles déclenchées lorsqu'un dispositif IoT en mouvement se connecte ou se déconnecte d'un relais sans fil dans l'écosystème Fog, en tenant compte d'un total de six scénarios différents, qui couvrent les cas les plus courants. Toutes ces actions ont été modélisées par des moyens algébriques (ACP) et ont été vérifiées par cette algèbre.

Les différents travaux cités ne permettent pas de décrire un état complexe d'un système Fog de par ses dimensions physique, virtuelle, comportementale avec une vérification formelle. En effet, les travaux basés sur la technique MDE sont génériques mais, en majorité, ne modélisent pas l'aspect virtuel des systèmes Fog. Les approches basée SMA prennent en considération l'aspect virtuel des systèmes Fog et modélisent leurs comportements mais sont rarement génériques et se concentrent sur une application précise. De plus, ils ne permettent pas l'analyse formelle de leur propriétés. Les systèmes à transition/événement ainsi que les modèles basés algèbres de processus spécifient rigoureusement la structure physique des systèmes Fog mais négligent souvent l'aspect virtuel, omettant ainsi la caractéristique d'intelligence dans la gestion du comportement des systèmes Fog.

Notre approche consiste à proposer un modèle formel générique exécutable et vérifiable, spécifiant les structures physique et virtuelle d'un système Fog et son comportement intelli-

gent tel que nous le démontrons dans la section suivante. Elle hérite des avantages des deux formalismes BRS et Maude pour proposer le langage Fog-DSML.

4.3 Principe de l'approche proposée

Dans le cadre de cette thèse, nous proposons une approche incrémentale de modélisation formelle des systèmes Fog et leur analyse. Nous formalisons d'abord leurs structures et leurs comportements selon plusieurs dimensions. Cette formalisation aboutit à la définition d'un langage de modélisation spécifique aux systèmes Fog que nous notons Fog-DSML. La Figure 4.2 illustre les différentes étapes de notre approche.

Le langage Fog-DSML est défini par le tuple : $\{AS_{Fog}, CS_{Fog}, Mac_{Fog}, SD_{Fog}, Mas_{Fog}\}$ où

1. Nous définissons la syntaxe abstraite (AS_{Fog}) du langage Fog-DSML sous forme de diagrammes de classes regroupant les différentes entités d'un système Fog et les relations entre ces dernières.

2. Nous définissons une architecture multi-couches pour le Fog computing nommée CLA*4*Fog (Cross-Layer Architecture for Fog computing) [Marir et al., 2022]. Nous nous basons sur la norme ISO/IEC/IEEE 42010 [ISO/IEC/IEEE, 2011] afin de définir une architecture dédiée aux systèmes Fog en général. Nous procédons au mapping Mac_{Fog} de la syntaxe abstraite du langage Fog-DSML vers cette architecture constituant sa syntaxe concrète CS_{Fog}.

3. Nous optons ensuite pour un formalisme réunissant le concept des Bigraphes et des Agents tel que proposé par [Pereira et al., 2012]. Afin de modéliser tous les aspects d'un systèmes Fog, nous étendons la structure virtuelle de ce formalisme afin de permettre aux agents présents dans ce modèle d'être "intelligents" et "décideurs". Nous définissons cette intelligence comme étant la capacité des agents à analyser leur environnement (la structure physique du système)[Marir et al., 2017] en plus de l'observer, et surtout d'interagir [Marir et al., 2020] les uns avec les autres avant la prise d'une quelconque décision. Ces décisions sont définies selon les stratégies que l'agent en question peut définir [Marir et al., 2022]. Nous effectuons donc le mapping Mas_{Fog} de la syntaxe abstraite vers la sémantique SD_{Fog} de Fog-DSML basée sur les BiAgents*.

4. Nous implémentons les différents concepts que nous avons défini afin de procéder à la simulation d'un modèle d'un système Fog. Nous effectuons la transformation du modèle BiAgents* vers un code exécutable. Afin de valider notre proposition. Nous proposons de modéliser un cas d'étude dans le domaine du Fog computing : Un système d'inspection des bagages dans un terminal d'aéroport [Marir et al., 2022] (LIS). Nous définissons plusieurs scénarios que nous modélisons à l'aide du langage Fog-DSML. Nous vérifions le comportement de ce dernier à l'aide de l'outil Maude stratégie par model checking, en vérifiant les différentes propriétés d'interopérabilité et de portabilité des données [Marir et al., 2022].

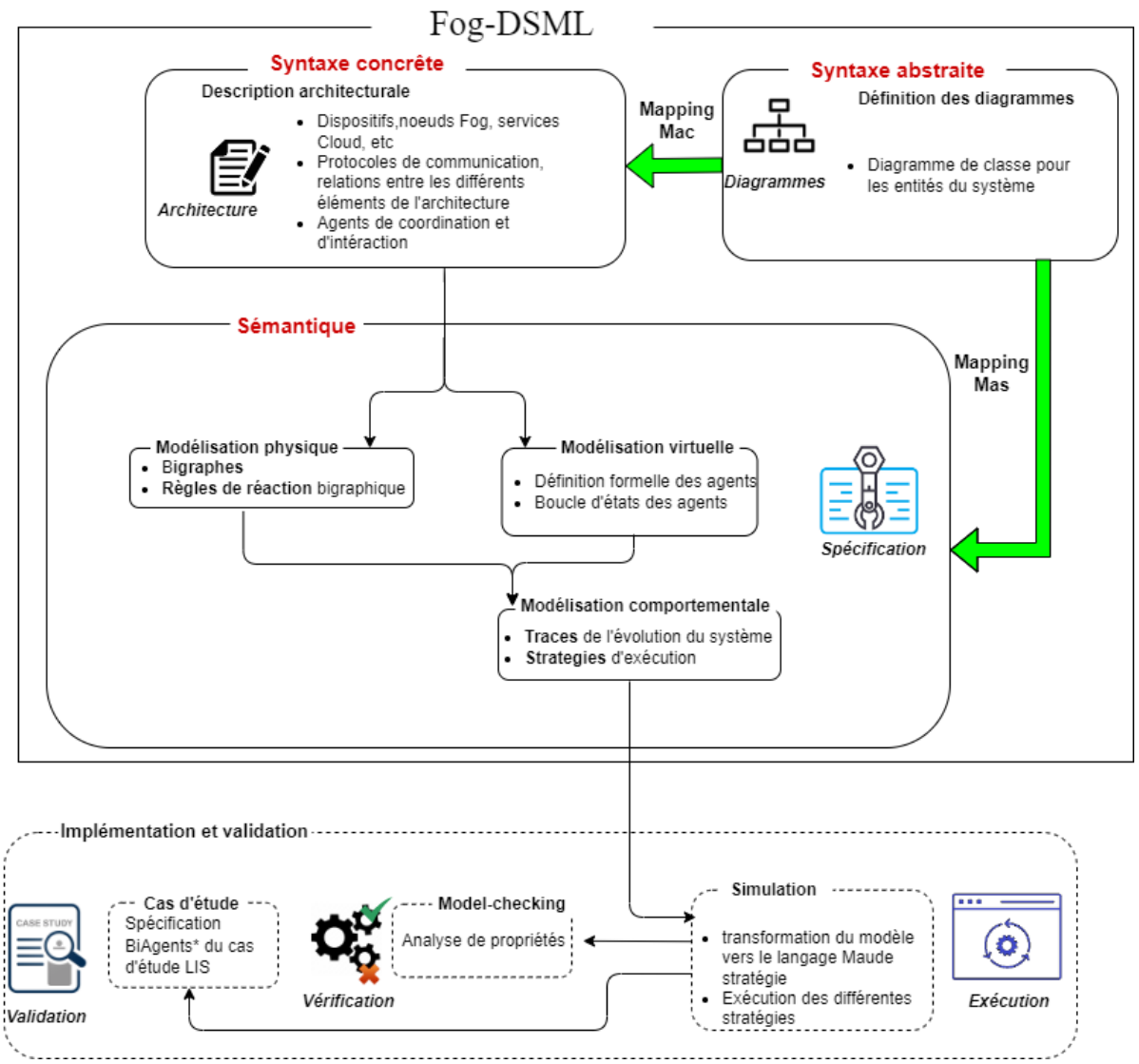


FIGURE 4.2 – Principe de notre approche

4.4 Syntaxes de Fog-DSML

4.4.1 Syntaxe abstraite

La Figure 4.3 représente la vision graphique de la syntaxe abstraite AS_{Fog} (le diagramme de classe) mettant en avant les éléments constituant un système Fog. L'aspect graphique est proposé afin de pallier progressivement à la complexité des systèmes Fog. Ce diagramme possède treize méta classes. En nous basant sur les la définition d'un système Fog ainsi que la norme internationale ISO/IEC/IEEE 42010 :2011 [ISO/IEC/IEEE, 2011] présentée dans la Figure 4.4 visant à déterminer les architectures des systèmes complexes, nous avons défini les classes impliquées dans la syntaxe abstraite de notre langage.

— La classe "*Système_ Fog*" modélise n'importe quel système Fog

— La classe "*Composants*" modélise les catégories d'éléments contenus dans un système Fog. Cette classe est composée de quatre méta classes "*Dispositifs*", "*Communication*", "*Cloud*" et "*Fog*".

— "*Dispositifs*" est la classe représentant les différents objets IoT impliqués dans un système Fog.

— La classe "*Communication*" modélise la communication entre les différents composants d'un système Fog représentant ainsi l'aspect interactionnel d'un système Fog.

— "*Cloud*" est la classe qui modélise les serveurs Cloud.

— "*Fog*" est la classe modélisant les différents nœuds installés dans la périphérie du réseau.

— La classe "*Partie_ prenante*" modélise les différents acteurs (les concepteurs, les développeurs, les architectes,etc).

— Les besoins de chaque acteur sont modélisés par la classe "*Besoin*". Cette classe est illustrée à travers les différents points de vues des parties prenantes.

— Un systèmes Fog peut avoir plusieurs aspects, la classe "*Aspects_ système*" modélise donc ces derniers. Il peut y avoir trois aspects d'un système Fog : "*physique*", "*virtuel*" et "*interactionnel* illustrant chacun une catégorie de composants d'un système Fog. La classe "*physique*" illustre les éléments physiques de tous les composants du système, la classe "*virtuel*" illustre les "*Agents*" intelligents faisant partie des éléments Fog du système et la classe "*interactionnel*" illustre la communication entre les différents composants du système. Nous remarquons ici que la communication est l'une des caractéristique à laquelle nous accordons une grande importance. Cette communication peut se faire entre des composants de même type ou bien hétérogènes.

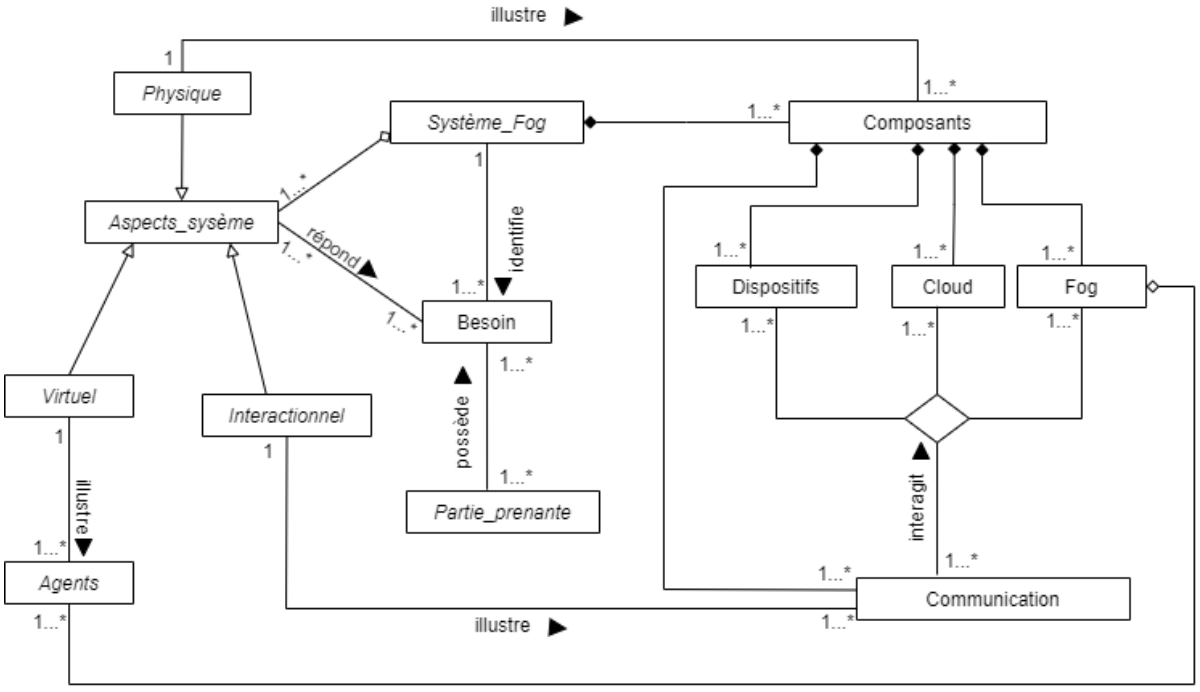


FIGURE 4.3 – Syntaxe abstraite AS_{Fog} de Fog-DSML : vue graphique

Ce diagramme se focalise sur l'identification des éléments conceptuels relatifs aux architectures des systèmes Fog, ce qui permet de simplifier le passage vers l'élaboration d'une architecture dédiée aux systèmes Fog.

4.4.2 Syntaxe concrète

Afin d'avoir un vocabulaire général et d'aider à soutenir la collaboration technique entre organisations, nous représentons la syntaxe concrète du langage Fog-DSML par une architecture multi-couches basée sur les différentes caractéristiques d'un système en respectant la norme internationale ISO/IEC/IEEE 42010 :2011 [ISO/IEC/IEEE, 2011] présentée dans la Figure 4.4. Nous présentons quelques-uns des termes du vocabulaire commun, utilisés dans la définition de l'architecture proposée :

— Point de vue : Un point de vue est un moyen de raisonner sur un système à partir d'une perspective spéciale.

— Vue : Une vue est une représentation des aspects visuels de l'architecture.

— Description architecturale : Afin d'illustrer les différents points de vue et leurs vues, nous décrivons l'architecture de manière graphique.

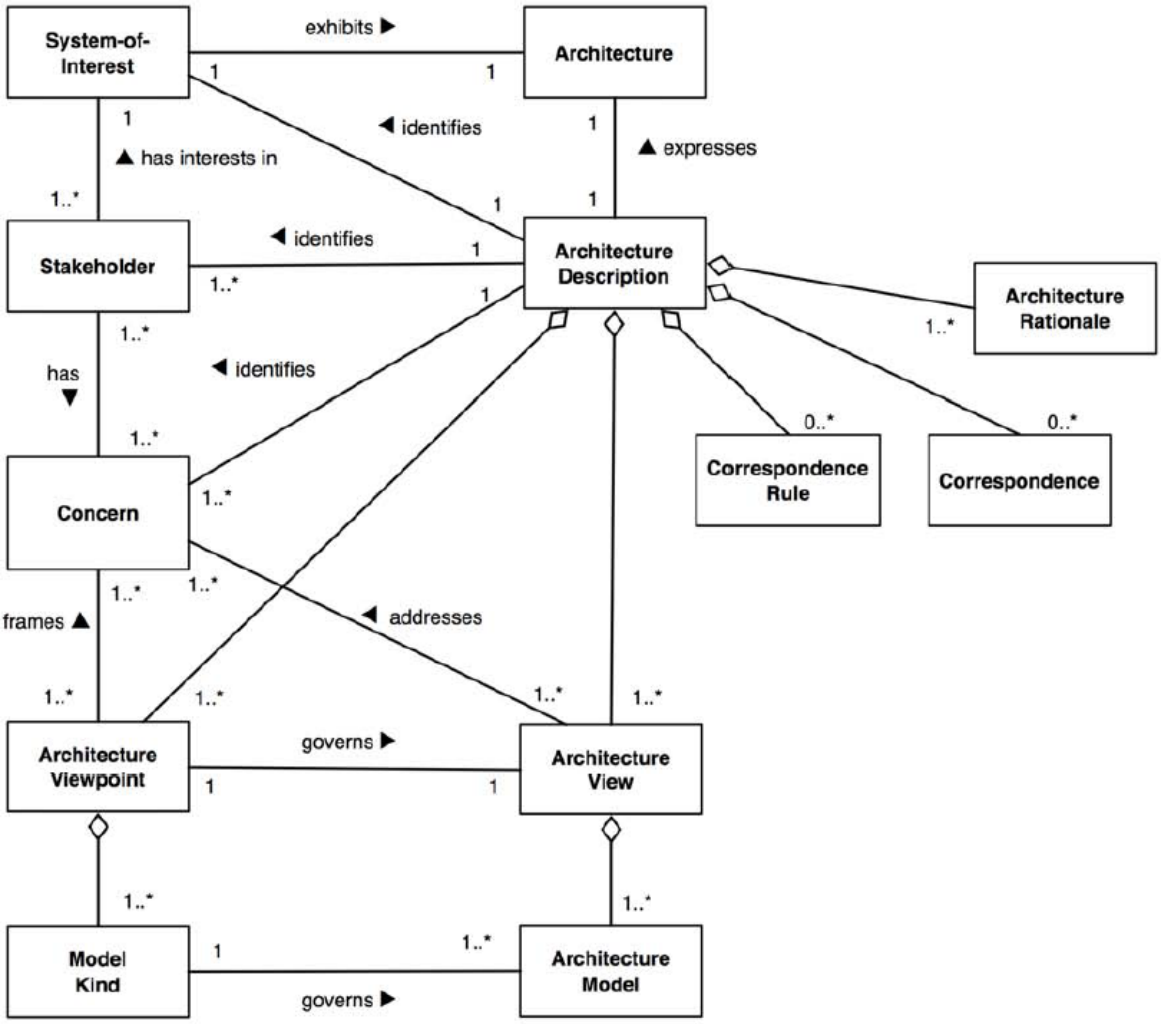


FIGURE 4.4 – Éléments de modèle conceptuel d'une description d'architecture selon la norme ISO/IEC/IEEE 42010 [ISO/IEC/IEEE, 2011]

Dans ce qui suit, nous détaillons les différents aspects de l'architecture CLA*4*Fog présentée dans la Figure 4.5, dont les éléments sont déduits de la norme à la Figure 4.4. Nous nous concentrons sur la présentation :

1. Des points de vue (*viewpoint*) de l'architecture d'un système Foh ;
2. Les vues de cette architecture ;
3. La description graphique de l'architecture elle-même. Les modèles qui découlent feront l'objet des chapitres suivants.

Points de vue de CLA*4*Fog ("*viewpoint*")

Nous présentons ici la façon dont les différentes parties de l'architecture pourraient être reliées, partiellement ou complètement, pour améliorer l'intelligence globale et les performances du système. Cette étape consiste à décrire les besoins de chaque intervenant dans l'écosystème du Fog. Cela inclut le développeur de l'application et le concepteur du système. La description de l'architecture CLA*4*Fog est une synthèse de ces nombreuses préoccupations des intervenants, qui sont désignées par les vues. Chaque intervenant peut avoir un point de vue différents et qui peut être :

— **Point de vue fonctionnel**
Le point de vue fonctionnel de l'architecture indique comment nous appliquons les éléments architecturaux et les vues, pour répondre à la faisabilité de diverses constructions dans une architecture Fog et l'adéquation de l'architecture proposée au comportement d'un système Fog selon une étude de cas donnée. Chaque scénario de cette étude de cas se concentrera sur un aspect différent du Fog computing.

— **Point de vue conceptuel**
Le point de vue conceptuel présente comment chaque partie de l'architecture résume une installation concrète d'un système Fog constituant une première étape avant la modélisation de ce type de systèmes. Ce point de vue répond au souci de vérification et de validation d'un modèle construit à partir de l'architecture CLA*4*Fog.

Vues de CLA*4*Fog

Nous catégorisons les différents éléments de l'architecture CLA*4*Fog en trois vues :

— **La vue physique** est représentée dans les quatre couches définies dans la description de l'architecture, et comprend les capteurs, les actionneurs, les protocoles de communication interne et externe, les systèmes Cloud et les services de stockage et de traitement des nœuds Fog.

— **La vue virtuelle** est représentée dans la couche médiane (Fog) de la Figure 4.5, qui comprend les différents agents responsable de la gestion des différents éléments du système.

— **La vue interactionnelle** est représentée entre les quatre couches par des flèches directionnelles. Cette interaction peut se faire entre les éléments physiques et virtuels.

Ces différentes vues, permettent aux personnes concernées de cibler chaque fonctionnalité d'un Fog système et de savoir comment imbriquer ces concepts afin de développer un système Fog ou bien d'élaborer un modèle de ce type de systèmes.

Dans le cadre de notre thèse, nous avons opté pour la modélisation d'un système Fog avant sa mise en œuvre, en considérant tous les aspects physiques, virtuels et interactionnels d'un tel système.

Description de l'architecture CLA*4*Fog

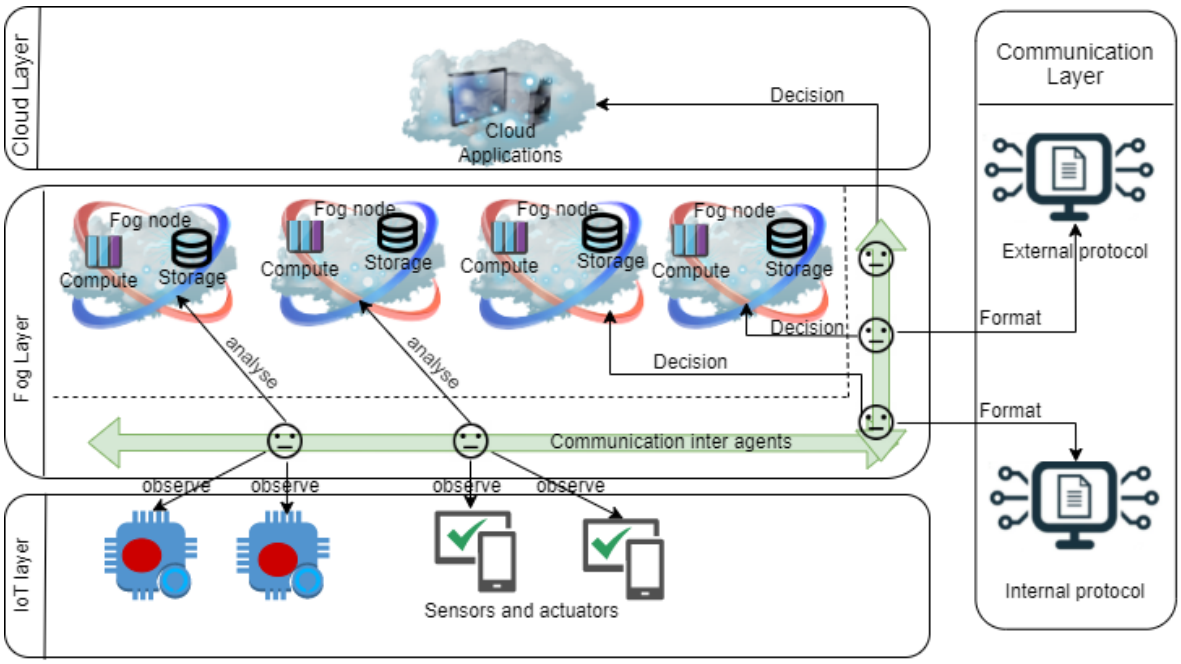


FIGURE 4.5 – Architecture en couche pour le Fog Computing CLA*4*Fog

Nous définissions dans la Figure 4.5 la description graphique de l'architecture CLA*4*Fog. Nous remarquons que cette description comprend quatre couches : trois horizontales et une verticale.

— La première couche horizontale contient les différents dispositifs impliqués en tant qu'objets dans un système Fog. Cette couche contient tous les dispositifs utilisés dans un système Fog. Ces dispositifs sont responsables de la génération des données à traiter en plus de l'affichage des informations nécessaires.

— La deuxième couche horizontale définit la couche Fog. Cette dernière contient tous les nœuds Fog impliqués dans le système. Chaque nœuds est composé d'une unité de calcule, une unité de stockage et d'un gestionnaire représenté par un agent intelligent. Ces nœuds sont responsables , en plus du stockage et des traitements de données, de la gestion globale du système. En effet, ce sont les agents contenus dans les nœuds Fog qui vont orchestrer le comportement globale du système Fog.

— La troisième couche horizontale représente la couche Cloud. Cette dernière contient les services Cloud impliqués dans le fonctionnement du système Fog.

— La couche verticale contient les différents protocoles de communication. A travers cette couche, toutes les données sont formatés et donc aptes à être analyser et traitées.

N'importe quel système Fog peut être représenté par l'architecture CLA*4*Fog. En effet, il est possible d'instancier cette architecture selon le système étudié. Chaque couche peut être composée sous forme de vues. Dans ce qui suit, nous détaillons cette décomposition afin de bien saisir les concepts présentés dans l'architecture.

Couche dispositifs

La couche dispositifs possède deux vues : Physique et Interactionnelle. les différents capteurs, les actionneurs, les données captées, les instructions déclenchées et les formats de données

constituent la vue physique de cette couche. les liens entre ces éléments et les protocoles de communication ainsi que les agents responsable de la gestion des dispositifs représentent la vue interactionnelle de cette couche. Dans le Tableau 4.2, nous regroupons les différents éléments de la couche dispositifs de l'architecture selon la vue qu'ils représentent.

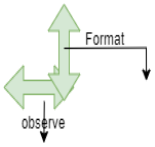
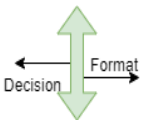
Couche communication La couche communication possède aussi deux vues : physique et interactionnelle. Les protocoles de communication (internes et externes) ainsi que les formats qu'ils contiennent sont des éléments physiques de l'architecture. Les différentes connexions entre les protocoles et les couches Fog Cloud à travers les agents responsables de la gestion de la communication entre couches représentent la vue interactionnelle de cette dernière. Dans le Tableau 4.2, nous regroupons les différents éléments de la couche communication de l'architecture selon leurs vues.

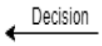
Couche Fog La couche Fog est la plus riche de l'architecture proposée. En effet cette couche possède les trois vues de l'architecture : physique, interactionnelle et virtuelle. Les nœuds Fog représentent la vue physique contenant les différents services Fog, les différentes connexions entre ces nœuds et les agents responsables de la gestion des dispositifs des protocoles de communication ainsi que des nœuds Fog et des services Cloud représentent la vue interactionnelle de la couche. De plus, tous les agents cités précédemment représentent la vue virtuelle de la couche et, par extension, de l'architecture globale. Effectivement, nous avons définis ces agents comme étant présents dans la couche Fog afin de représenter la distribution de ces agents dans la périphérie du réseau du système. Dans le Tableau 4.2, nous regroupons les différents éléments de la couche Fog de l'architecture selon ses trois vues.

Couche Cloud La couche Cloud représentant la couche la plus élevée de l'architecture et possède quant à elle deux vues : physique et interactionnelle. Les services Cloud sont les éléments physiques constituant la couche Cloud et la liaison entre ces systèmes et l'agent gestionnaire de la couche Cloud représente la vue interactionnelle. Dans le Tableau 4.2, nous montrons les différentes représentations de ces vues dans la couche concernée.

TABLE 4.2 – Vues de la couches Dispositifs

Élément architectural	**Représentation graphique**
Vue physique	
Capteur	
Donnée	
Actionneur	
Instruction	
Vue Interactionnelle	

Interaction agent/dispositif	
Interaction dispositifs/protocoles	
Vue physique	
Protocole	
Format	
Vue Interactionnelle	
Interaction Protocole/agents	
Vue Physique	
Nœud Fog	
Vue Interactionnelle	
Interaction nœud Fog/agent	
Interaction nœud Fog/Cloud	
Interaction nœud Fog/dispositifs	
Interaction nœud Fog/protocoles	
Vue Virtuelle	

Agents	
Vue physique	
Service Cloud	
Vue interactionnelle	
Interaction Cloud/agent	

Le mapping de la syntaxe abstraite vers la syntaxe concrète illustré dans le Tableau 4.3 consiste en la représentation de chaque éléments du méta modèle présenté dans la Section 4.4.1 par un élément de l'architecture CLA*4*Fog. Cette transcription pourrait servir à l'automatisation et l'implémentation d'un éditeur graphique de Fog-DSML.

TABLE 4.3 – Mapping de AS_{Fog} vers $CS*_{Fog}$

<table>
<tr><th colspan="2">Syntaxe abstraite</th><th colspan="2">Syntaxe concrète</th></tr>
<tr><td rowspan="12">Classes</td><td>Système_Fog</td><td colspan="2">Description architecturale</td></tr>
<tr><td>Composants</td><td colspan="2">Couches de l'architecture</td></tr>
<tr><td>Dispositifs</td><td colspan="2">Couche IoT</td></tr>
<tr><td>Fog</td><td colspan="2">Couche Fog</td></tr>
<tr><td>Cloud</td><td colspan="2">Couche Cloud</td></tr>
<tr><td>Communication</td><td colspan="2">Couche Communication</td></tr>
<tr><td>Besoin</td><td colspan="2">Point de vue</td></tr>
<tr><td>Partie_prenante</td><td colspan="2">Concepteur</td></tr>
<tr><td>Agents</td><td colspan="2">Agents intelligents</td></tr>
<tr><td>Physique</td><td colspan="2">Vue physique</td></tr>
<tr><td>Virtuel</td><td colspan="2">Vue virtuelle</td></tr>
<tr><td>Interactionnel</td><td colspan="2">Vue interactionnelle</td></tr>
<tr><td rowspan="13">Relations</td><td>1...* Communication
≪ intéragit ▶ ≫
1...* Dispositif & Cloud & Fog</td><td colspan="2">La couche communication est verticale, reliée aux trois autres couches</td></tr>
<tr><td rowspan="6">1 Physique ≪ illustre ▶ ≫
1...* Composants</td><td rowspan="4">Couche IoT</td><td>Capteurs</td></tr>
<tr><td>Donnée</td></tr>
<tr><td>Actionneurs</td></tr>
<tr><td>Action</td></tr>
<tr><td rowspan="2">Couche Communication</td><td>Protocole</td></tr>
<tr><td>Format</td></tr>
<tr><td>Couche Fog</td><td>Nœud Fog</td></tr>
<tr><td>Couche Cloud</td><td>Service Cloud</td></tr>
<tr><td rowspan="7">1 Interactionnel
≪ illustre ▶ ≫ 1...*
Communication</td><td rowspan="2">IoT</td><td>Agent/Dispositif</td></tr>
<tr><td>Dispositif/Protocole</td></tr>
<tr><td>Communication</td><td>Protocole/Agent</td></tr>
<tr><td rowspan="4">Fog</td><td>Nœud Fog/Agent</td></tr>
<tr><td>Nœud Fog/Cloud</td></tr>
<tr><td>Nœud Fog/dispositif</td></tr>
<tr><td>Nœud Fog/protocole</td></tr>
<tr><td>Cloud</td><td>Cloud/Agent</td></tr>
<tr><td>1 Virtuel
≪ illustre ▶ ≫ 1...*
Agents</td><td colspan="2">Plusieurs agents intelligents</td></tr>
<tr><td>1...* Agents (−◇) Fog</td><td colspan="2">Les agents sont installés dans les nœuds Fog et communiquent entre eux et avec les autres éléments des autres couches</td></tr>
<tr><td>1 Système_Fog
≪ identifie ▶ ≫ 1...*
Besoins</td><td colspan="2">La description architecturale identifie des points de vue fonctionnel et conceptuel</td></tr>
<tr><td>1...*
Partie_prenante
≪ possède ▶ ≫ 1...*
Besoin</td><td colspan="2">Les concepteurs ont les deux points de vue sur l'architecture Fog</td></tr>
<tr><td>1...*
Aspects_système
≪ répond ▶ ≫ 1...*
Besoin</td><td colspan="2">Les vues physique, virtuelle et interactionnelle répondent aux besoins conceptuels et fonctionnels des concepteurs</td></tr>
</table>

4.5 Conclusion

Dans ce chapitre, nous avons recensé plusieurs travaux existants dans la littérature liés à la conception des systèmes Fog. Nous avons divisé ces efforts de recherche en quatre classes basés sur les concepts suivants : MDE, SMA, Système à transition/événement et algèbres de processus. Nous avons par la suite étudié l'ensemble des solutions et approches présentées selon un certain nombres de critères, que nous avons jugé pertinents, afin de positionner notre approche par rapport aux efforts de recherche similaires, dans le but de préparer le lecteur à bien situer nos contributions. Nous avons ensuite introduit notre approche, un langage de modélisation spécifique aux systèmes Fog (Fog-DSML) et nous avons présenté ses différentes syntaxes. Nous avons, d'abord, défini sa syntaxe abstraite en proposant un méta-modèle sous forme de diagramme de classes. Par la suite nous avons déterminé la syntaxe concrète du langage en proposant une architecture multi-couches dédiée aux systèmes Fog CLA*4*Fog. Nous nous sommes basés sur la norme internationale ISO/IEC/IEEE 42010 :2011 afin de définir cette architecture selon différents points de vues permettant le raisonnement sur cette dernière. Nous avons aussi présenté les différentes vues de l'architecture offrant ainsi un support méthodologique permettant de décomposer un système Fog en un ensemble d'éléments compréhensibles. Ensuite, nous avons présenté le passage de la syntaxe abstraite vers la syntaxe concrète représentant la première fonction de mapping du langage Fog-DSML.

Chapitre 5

Sémantique basée BiAgents* pour Fog-DSML

Sommaire

5.1 Introduction

La définition des syntaxes du langage Fog-DSML nous permet de faire un premier pas dans la modélisation des systèmes Fog. En effet, la syntaxe abstraite décrit un système Fog et ses éléments. L'architecture CLA4Fog représentant la syntaxe concrète du langage proposé illustre de manière intuitive les vues physique, virtuelle et interactionnelle que peut avoir un système Fog. Cependant, il est impératif de définir une sémantique structurelle et comportementale liée à ce langage afin de donner une interprétation claire et rigoureuse des différents éléments d'un système Fog et leur interaction. Dans le cadre cette thèse, nous estimons que les méthodes formelles, caractérisées par leur efficacité, fiabilité et précision, incarnent une excellente solution pour réduire et maitriser la complexité liée à la modélisation des systèmes Fog.

Le comportement intelligent d'un système Fog est généralement décrit à travers les différentes communication entre le Cloud, le Fog et les différents dispositifs impliqués de manière à minimiser le temps de réponse lors de traitements critiques tout en conservant un traitement dans le Cloud lors de comportements classiques.

En vue de sa grande complexité, le comportement d'un système Fog peut entrainer des situations indésirables telles que l'instabilité dans la gestion des services et la dégradation sévère de la qualité de service et de la fiabilité du système. Ainsi, il est nécessaire de définir un modèle permettant de décrire précisément la structure des systèmes Fog. Un tel modèle sert à déterminer les changements potentiels pouvant être appliqués au niveau d'une architecture Fog, et d'en restreindre le champ de possibilités, afin d'assurer une dynamique correcte et prévenir les comportements non désirables.
Nous présentons dans ce chapitre la sémantique SD_{Fog} du langage Fog-DSML. Tout d'abord, nous présentons le formalisme sur lequel notre approche est basée, le formalisme BiAgents* étant une extension du formalisme BiAgent. Nous proposons un modèle basé sur cette extension afin de modéliser les différentes structures physique et virtuelle ainsi que les comportements d'un système Fog. Nous définissons les logiques spatiale et temporelle TQL et LTL permettant la définition de propriétés structurelles et comportementale des systèmes Fog. Nous définissons les propriétés de portabilité des données et d'interopérabilité syntaxique et sémantique à l'aide de ces logiques dans le but d'analyser le modèle proposé.

5.2 Extension des BiAgents

Le formlisme des BiAgents tel que défini par [Pereira et al., 2012] permet à un système modélisé par les bigraphes d'être observé par des agents et d'évoluer d'un état vers un autre à travers cette observation.

Nous enrichissons le formalisme des BiAgents [Pereira et al., 2012] en maintenant la séparation entre les concepts de structure physique et virtuelle, tout en spécifiant de manière plus précise le comportement de la structure virtuelle composée d'agents. En effet, un agent observe son environnement, l'analyse, prend une décision, peut la communiquer à d'autres agents et contrôle la partie bigraphique en déclenchant des règles de réaction, en plus de sa capacité à migrer à travers les différents nœuds et régions bigraphiques.

L'extension de ce formalisme va en adéquation avec les différents aspects des systèmes complexes tels que les systèmes ambiants en général. Les systèmes modernes basés sur le Fog, l'IoT, le Cot, l'IoE,etc. requièrent une forme d'intelligence dans la représentation de leur comportement qui peut être modélisée par ce formalisme étendu. En effet, le formalisme BiAgents* est composé d'un modèle physique et un autre virtuel qui définissent ensemble le comportement intelligent du système (Figure 5.1). Évidemment, un BiAgent* $=\mathcal{A}^* \bullet \mathcal{B}^*$ hérite la définition de

sa partie physique du formalisme des BiAgents (voir Définition 21 Chapitre 3). Sa dimension virtuelle $\mathcal{A}^*$ est donnée par une définition alternative et plus détaillée (Définition 24).

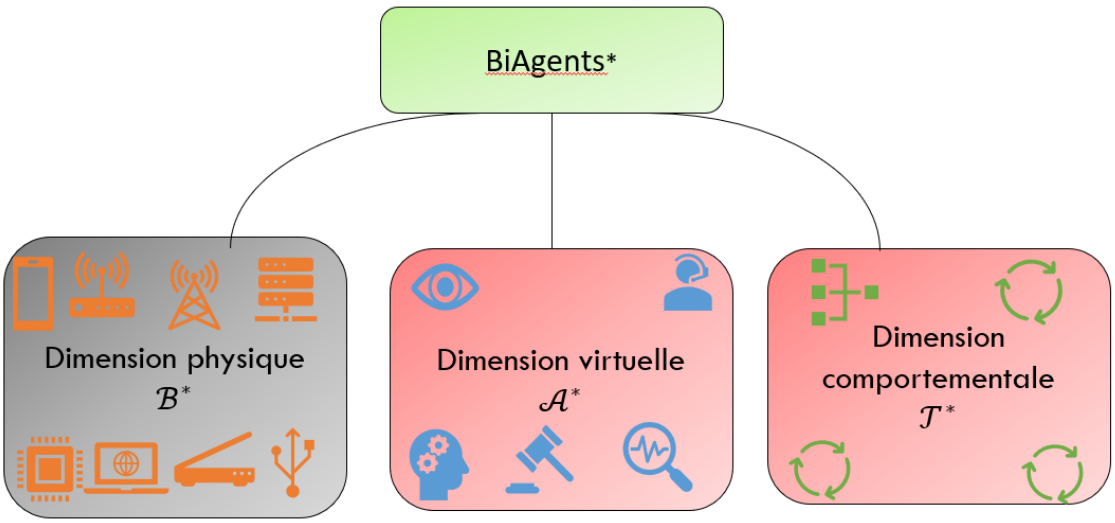


FIGURE 5.1 – Dimensions d'un BiAgent*

À partir d'une définition plus précise de l'ensemble des agents a_i qui forment la structure $\mathcal{A}^*$ d'un BiAgent*, nous en déduisons une troisième dimension aussi importante que les autres, il s'agit de la trace d'un BiAgent* (voir Figure 5.1). Nous détaillons dans la suite les dimensions virtuelle et comportementale d'un BiAgent*.

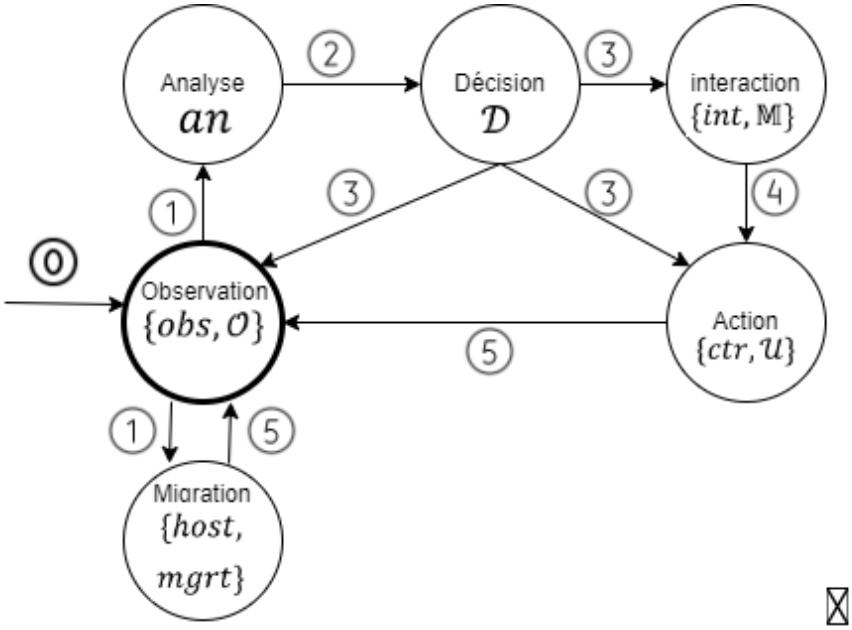


FIGURE 5.2 – États possibles d'un agent

Définition 24 La dimension virtuelle $\mathcal{A}^*$ du BiAgent* $= \mathcal{A}^* \bullet \mathcal{B}^*$ est définie par un ensemble d'agents a_i. Chaque a_i pouvant être dans un état S_{a_i} tel que $S_{a_i} = \{Observation, Migration,$

$Analyse, Decision, Interaction, Action\}$. Chacun des états $S_{a_i} \in \mathbb{S}_{a_i}$ est défini par les éléments suivants :

— Si $S_{a_i} = Observation$, $S_{a_i} = (obs, \mathcal{O})$ où $\mathcal{O}$ est l'ensemble des différents états du système (les bigraphes) que l'agent $a_i$observe ($\mathcal{O} \subseteq \mathbb{B}$) et *obs* est la fonction d'observation de l'état du système, elle retourne une observation selon un bigraphe et l'hôte de l'agent obs : $\mathbb{B} \times (V_{\mathbb{B}} \cup host) \to \mathcal{O}$

— Si $S_{a_i} = Migration$, $S_{a_i} = (host, mgrt)$ où *host* est l'hôte courant de l'agent a_i (un nœud ou une région) et *mgrt* est la fonction de migration qui donne le prochain hôte de l'agent selon l'hôte courant. $mgrt : (V_{\mathbb{B}} \cup host) \times \mathcal{O} \to (V_{\mathbb{B}} \cup host)$

— Si $S_{a_i} = Analyse$, $S_{a_i} = (an)$ où *an* est la fonction d'analyse du système, l'agent analyse son hôte ou ses nœuds fils et retourne sa décision qui est un chemin d'exécution possible $an : \mathcal{P}(\mathcal{O}) \times V_{\mathbb{B}} \to \mathcal{D}$

— Si $S_{a_i} = Decision$, $S_{a_i} = (\mathcal{D})$ où $\mathcal{D}$ est l'ensemble des décisions que l'agent a_i peut prendre après analyse ($\mathcal{D} \subseteq \mathcal{P}(\mathbb{L})$). Nous précisons que $\mathbb{L}$ est l'ensemble des labels des règles de réactions définies dans la structure physique $\mathcal{B}$

— Si $S_{a_i} = Interaction$, $S_{a_i} = (int, \mathbb{M})$ où *int* est la fonction d'interaction entre les agents. L'agent a_i envoie un message après analyse à un autre agent $a \subset \mathcal{A}^*, int : \mathcal{A}^* \times \mathcal{D} \to \mathbb{M}^a$ et $\mathbb{M}$ est l'ensemble des messages reçus par l'agent a_i

— Si $S_{a_i} = Action$, $S_{a_i} = (ctr, \mathcal{U})$ où *ctr* est la fonction d'application d'une décision prise par l'agent a_i après analyse ou réception de message par un autre agent, $ctr : \mathcal{D} \uplus \mathbb{M} \to \mathcal{P}(\mathcal{U})$. $\mathcal{P}(\mathcal{U})$ étant l'ensemble des sous ensembles de $\mathcal{U}$. $\mathcal{U}$ est l'ensemble des règles de réaction que l'agent a_i peut appliquer selon un nœud du bigraph (il spécifie l'ensemble des actions que a_i peut appliquer dans une partie spécifique du système ($\mathcal{U} \subseteq \mathbb{U}$)

La partie virtuelle des BiAgents* consiste en un ensemble d'agents. Chaque agent peut être dans un état donné à la fois. Initialement, il est à un état « observation », puis, il peut évoluer vers un état « analyse » ou « migration ». De l'état « analyse », l'agent peut passer à l'état de « décision ». Après avoir été dans l'état de décision, il peut revenir à l'état initial « observation » ou passer à l'état « interaction » avec d'autres agents, ou appliquer une « action » dans la structure du système (déclenchement d'une règle de réaction) ainsi qu'illustré par la Figure 5.2.

La nouvelle définition de la structure virtuelle de BiAgent* implique l'extension de la définition formelle d'une trace d'un BiAgent (présentée dans la Définition 23 du Chapitre 3 afin de définir le comportement intelligent d'un système modélisé par un BiAgent*.

Définition 25 (Trace*) Une trace T d'un BiAgent* est une séquence de paires de bigraphes $\langle B0, B1, ...\rangle$ et d'hôtes $(H0, H1, ...)$ tels que pour chaque $(B_i, H_i$ et (B_{i+1}, H_{i+1}), il y a une règle de réaction étendue R_i permettant $(B_i, H_i) \to (B_{i+1}, H_{i+1})$.

Nous notons $T = B_0 \overset{\text{R1}}{\prec} B_1 \overset{\text{R2}}{\prec} B_2 ... \overset{\text{Rn}}{\prec} B_n$.

Une règle R_i peut être :

— Une règle de réaction agissant sur la structure physique de BiAgents* R_p

— Une règle de transition affectant les états de l'agent R_v

Définition 26 (Projection*)

$$t^a = (B_i, h_j^a) \overset{\substack{obs \\ an \\ int}}{\prec} (B_i, h_j^a) \overset{\text{mgrt}}{\prec} (B_i, h_{j+1}^a) \overset{\text{R}}{\prec} (B_{i+1}, h_{j+1}^a) \prec ...$$

Où chaque B_i est le bigraphe contenant l'hôte ${h_i}^a$.

À présent, le passage d'un élément à l'autre dans une projection de trace se fait à travers deux type d'opérations, les fonctions modifiant la localité de l'agent ou la structure physique du système, et les fonctions qui font évoluer l'état de l'agent. La modification de l'état de l'agent n'a pas d'impact sur la structure du système modélisé. De ce fait, les fonctions d'observation (obs), d'analyse (an) et d'interaction (int) n'ont pas d'impact sur le couple (bigraphe, hôte), car ni la structure du bigraphe, ni l'hôte de l'agent ne changent. Inversement, les fonctions de migration ($mgrt$) et les règles de réaction (R) ont un effet sur ce couple. Après la fonction $mgrt$ l'hôte de l'agent change et après une règle de réaction R la structure physique du système évolue.

À travers les différentes définitions des dimensions du modèle BiAgents*, nous constatons que ce dernier est un modèle générique pouvant définir les systèmes complexes intelligents. Nous adoptons ce formalisme pour modéliser les systèmes Fog et raisonner correctement sur leur comportement.

5.3 Sémantique basée BiAgents* de Fog-DSML

$SD_{Fog} = \{\mathcal{B}^*{}_{Fog}, \mathcal{A}^*{}_{Fog}, \mathcal{T}^*{}_{Fog}\}$ est la sémantique de Fog-DSML composée de trois aspects : physique, virtuel et comportemental. Chaque élément de SD_{Fog} est dédié à la description précise des aspects d'un système Fog modélisé par Fog-DSML.

5.3.1 Aspect physique

La modélisation de la partie physique d'un système Fog décrit par le langage Fog-DSML se fait par la structure physique du modèle BiAgents*. En effet, $\mathcal{B}^*{}_{Fog}$ est constitué de l'ensemble des bigraphes représentant les différents états physiques du système Fog, l'ensemble des règle de réaction permettant le passage de l'un de ces états à l'autre, l'ensemble de contrôles qui spécifie quelle partie du système Fog change exactement après application de ces règles. Formellement, cette structure est définie par instanciation de la Définition 21 Chapitre 3.

Définition 27 Étant donné un système Fog F décrit par Fog-DSML, la sémantique précise de sa structure physique $\mathcal{B}^*{}_{Fog}$ est spécifiée par :

$$\mathcal{B}^*{}_{Fog} = (\mathbb{B}^*{}_{Fog}, \mathcal{R}^*{}_{Fog}, \mathbb{U}^*{}_{Fog}, {B_0}^*{}_{Fog}, F^*{}_{Fog})$$

avec :

- $\mathbb{B}^*{}_{Fog}$ est l'ensemble des bigraphes représentant les états du système Fog, chaque bigraphe représente un état à un moment donné.
- $\mathcal{R}^*{}_{Fog}$ est l'ensemble des règles de réaction labellisées modélisant les différentes modifications possibles (déplacement de dispositifs, de donnée, connexion, déconnexion, etc.) d'un état physique de F à un autre. L'ensemble des labels est noté $\mathbb{L}$.
- $\mathbb{U}^*{}_{Fog}$ est l'ensemble des actions $\mathcal{R}^*{}_{Fog} \times V_{\mathbb{B}^*{}_{Fog}}$ avec $V_{\mathbb{B}^*{}_{Fog}}$ l'ensemble des nœuds de chaque bigraphe $B_i \in \mathbb{B}^*{}_{Fog}$, chaque action est liée à un nœud spécifique appartenant à $V_{\mathbb{B}^*{}_{Fog}}$.
- B_{0Fog} représente l'état physique initial de F.
- $F^*{}_{Fog}$ est la fonction de transition que le modèle utilise pour atteindre un nouvel état $F^*{}_{Fog} : \mathbb{B}^*{}_{Fog} \times (\mathcal{R}^*{}_{Fog} \times V_{\mathbb{B}^*{}_{Fog}}) \rightarrow \mathbb{B}^*{}_{Fog}$.

Le Tableau 5.1 associe à chaque élément de la structure physique d'un système Fog l'abstraction équivalente dans la partie Bigraphe du modèle. Il s'agit du contrôle associé à chaque entité, de son arité (nombre de ports) et de son type associé. Des sortes sont utilisées pour distinguer

les types de nœuds à des contraintes structurelles, par exemple, certains nœuds doivent avoir une sorte spécifique de parent afin d'être en adéquation avec le paradigme Fog, tandis que les contrôles identifient les états et les paramètres qu'un nœud peut avoir. Nous catégorisons les éléments d'un système Fog comme indiqué dans le Tableau 5.1.

TABLE 5.1 – Discipline de typage ("*sorting*") associée à $\mathbb{B}^*{}_{Fog}$.

Élément du système Fog	**Arité**	**Sorte**
Couche IoT (région 0)		
Capteur	2	i
Actuateur	1	k
Donnée capturée	2	e
Couche Fog (région 1)		
Nœud Fog	2	f
Couche de communication (région 2)		
Protocole de communication	2	c
Formats de données	2	o
Couche Cloud (région 3)		
Serveurs Cloud	2	a
Services	1	x

La structure physique $\mathcal{B}^*{}_F$ du modèle BiAgents* permet d'exprimer différents concepts de l'architecture d'un système Fog. Ce formalisme permet de modéliser les aspects hiérarchiques de CLA*4*Fog en plus de la modélisation de ses éléments hétérogènes à l'aide de la discipline de *sorting*. De plus, la répartition et les liaisons entre ces éléments sont bien exprimées à l'aide des fonctions *link* et *prnt*.

En outre, certaines règles de formation ϕi peuvent être ajoutées pour la construction correcte des spécifications bigraphiques de tout système Fog. Ces règles précisent les contraintes structurelles à respecter pour former et gérer les structures physiques d'un système Fog, à titre d'exemple et comme le montre le Tableau 5.2.

— La règle ϕ0 spécifie que les capteurs n'hébergent que les données capturées.

— La règle ϕ1 spécifie que les actionneurs n'hébergent que des services.

— Les règles ϕ2 et ϕ3 spécifient que les services sont composées uniquement selon les données formatées capturées.

— Les règles ϕ4 et ϕ6 expriment que les nœuds Fog et les serveurs Cloud n'hébergent que des services ou des données capturées formatées.

— Dans la règle ϕ5, nous montrons que les protocoles produisent les formats utilisés pour formater les données capturées.

— La règle ϕ7 stipule que les nœuds Fog et les serveurs Cloud sont les seuls responsables de la production de services.

— La règle ϕ8 stipule qu'un format est toujours lié aux données capturées, ce qui permet à ces données de se déplacer à travers les différentes parties du système.

— Dans les règles ϕ9 à ϕ12, nous définissons structurellement les différentes couches du système Fog.

— Dans la règle ϕ13, les nœuds atomiques n'ont pas d'enfants.

Le comportement exprimant la dynamique des systèmes Fog est défini par un ensemble de règles de réaction génériques $\mathcal{R}^*{}_{Fog}$ dont les plus importantes sont résumées dans le Tableau

TABLE 5.2 – Règles de formation ϕ pour ${\mathbb{B}^*}_F$

Règle de formation	Description
$\phi 0$	Tous les fils des nœuds i sont de sorte e
$\phi 1$	Tous les fils des nœuds k sont de sorte x
$\phi 2$	Tous les fils des nœuds x sont de sorte o
$\phi 3$	Tous les fils des nœuds o sont de sorte e
$\phi 4$	Tous les fils des nœuds f sont de sorte ox
$\phi 5$	Tous les fils des nœuds c sont de sorte oe
$\phi 6$	Tous les fils des nœuds a sont de sorte ox
$\phi 7$	Dans un nœud x, un port peut être relié à un nœud de sorte fa
$\phi 8$	Dans un nœud de sorte o un port est toujours relié à un nœud de sorte e et l'autre port peut être relié à un nœud de sorte fxa
$\phi 9$	Tous les fils de la région 0 sont de sorte ik
$\phi 10$	Tous les fils de la région 1 sont de sorte c
$\phi 11$	Tous les fils de la région 2 sont de sorte f
$\phi 12$	Tous les fils de la région 3 sont de sorte a
$\phi 13$	Tous les nœuds de sorte e sont atomiques

5.3. Les éventuelles règles de réaction qui ne prennent pas en considération la partie virtuelle et intelligente du système sont classées selon leur fonctionnalité.

TABLE 5.3 – Règles de réaction possibles de ${\mathcal{B}^*}_F$ associées à un système Fog.

Description de la règle	**Forme algébrique**
Prendre en charge des données captées par les protocoles	$R1 \stackrel{\text{def}}{=} ((i.e)\|id \parallel (c.o)\|id) \rightarrow (i\|id) \parallel (c.(o\|e))\|id$
Ignorer les données captées	$R13 \stackrel{\text{def}}{=} (i\|id) \parallel (c.(o\|e))\|id \rightarrow (i\|id) \parallel (c.o)\|id$
Formater les données selon les protocoles de communication	$R2 \stackrel{\text{def}}{=} (c.(o\|e))\|id \rightarrow (c.(o.e))\|id$
Envoyer les données formatées aux nœuds Fog	$R3 \stackrel{\text{def}}{=} (c.(o.e))\|id \parallel (f.x)\|id \rightarrow c\|id \parallel (f.x)\|(o.e)\|id$
Générer un service dans le Fog	$R4 \stackrel{\text{def}}{=} (f.x)\|(o.e)\|id \rightarrow (f.(x.(o.e)))\|id$
Actionner un service Fog	$R5 \stackrel{\text{def}}{=} k\|id \parallel f.(x.(o.e))\|id \rightarrow (k.(x.(o.e)))\|id \parallel f\|id$
Envoyer les données formatées au Cloud	$R10 \stackrel{\text{def}}{=} (c.(o.e))\|id \parallel (a.(x))\|id \rightarrow c\|id \parallel (a.(x\|(o.e)))\|id$
Générer un service dans le Cloud	$R11 \stackrel{\text{def}}{=} (a.(x\|(o.e))\|id \rightarrow (a.(x.(o.e)))\|d1))\|id$
Actionner un service Cloud vers l'actuateur	$R12 \stackrel{\text{def}}{=} k\|id \parallel (a.(x.(o.e)))\|id \rightarrow (k.(x.(o.e)))\|id \parallel a\|id$

Nous remarquons que ces règles de réaction traduisent des actions inter et intra couches dans un système Fog.

Exemple

Nous considérons dans ce chapitre un simple exemple de système de reconnaissance de voix dans une société lambda. Ce système comprend un capteur de voix, un verrou, une alarme un nœud Fog et communique avec le serveurs distant de la société. Ces différentes entités sont connectées afin de fournir des services aux employés de la société. En effet, chaque personne doit prononcer son propre nom afin que la porte s'ouvre. Ce système reste à l'écoute des voix dans un couloir et un service Cloud autorise l'ouverture d'une porte s'il reconnait une personne faisant partie du personnel de la société. Si quelqu'un prononce le nom d'un employé mais que le système n'associe pas la voix au nom, une alarme est déclenchée grâce au nœud Fog. Ce système peut être représenté par la syntaxe concrète CS_{Fog} de Fog-DSML (Figure 5.3).

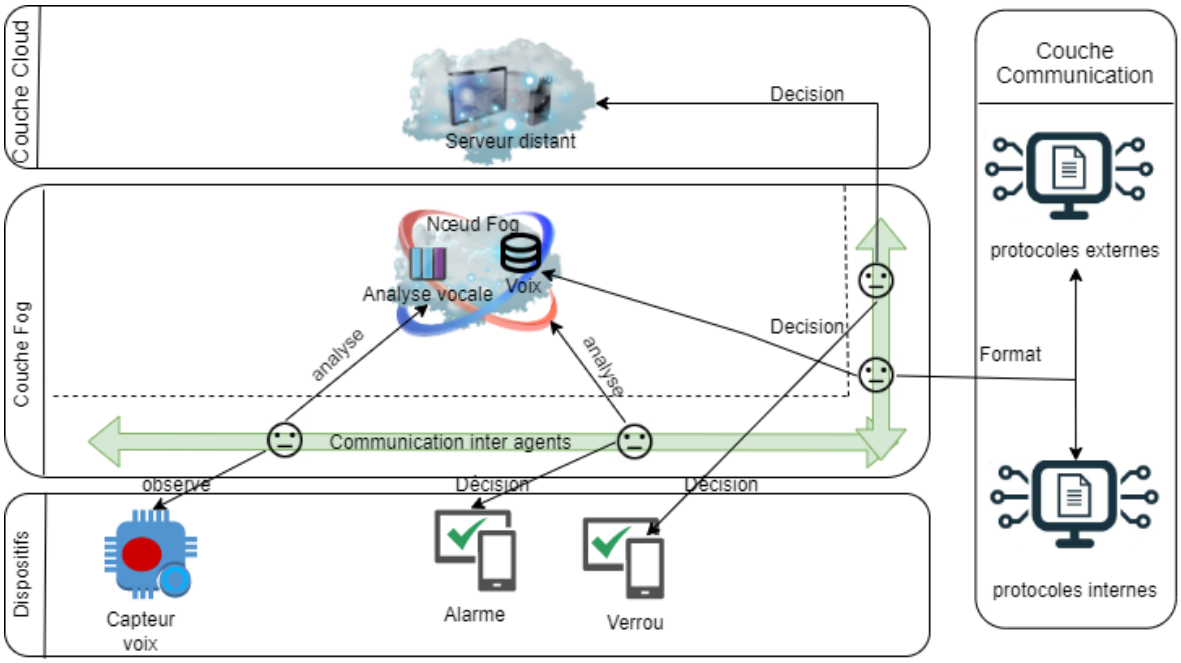


FIGURE 5.3 – Description d'architecture du système de reconnaissance de voix

Dans ce qui suit, nous définissons la sémantique du langage Fog-DSML et nous illustrons les différents concepts présentés selon l'exemple Figure 5.3.

Nous définissons la structure physique du système Fog de reconnaissance de voix présenté précédemment par :

$$\mathcal{B}_{ex} = (\mathbb{B}_{ex}, \mathcal{R}_{ex}, \mathbb{U}_{ex}, B_{exF}, F_{ex})$$

— Les différents bigraphes représentant les états physiques du système $\mathbb{B}_{ex} = \{B_F, B_O, B_A\}$

— Les règles de réaction qui font évoluer la structure physique du système $\mathcal{R}_{ex} = \{ouvrir, alerter\}$

— Les actions précisant l'application des règles de réaction selon les nœuds du bigraphe $\mathbb{U}_{ex} = \{(ouvrir, serviceCloud), (alerter, noeudFog)\}$

— L'état initial est défini par B_{exF} (voir Figure 5.5)

— La fonction de transition F_{ex} définit le passage d'un état à l'autre selon la fonction de contrôle et le bigraphe courant

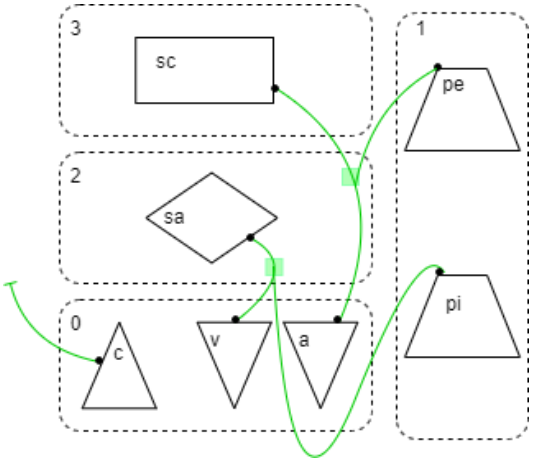


FIGURE 5.4 – Représentation graphique de l'état initial B_{exF}

Nous constatons à cette étape de modélisation que le système n'évoluera pas comme souhaité si nous appliquons uniquement ces règles sans la participation des agents. Le comportement intelligent d'un système Fog peut être représenté par la gestion de ses parties physiques par des entités virtuelles, constituées d'un ensemble d'agents. Chacun observe et analyse son environnement et ces entités contrôlent l'exécution du système. De plus, l'analyse et la communication entre ces entités permet de guider correctement cette exécution.

5.3.2 Aspect virtuel

L'aspect virtuel $\mathcal{A}^*_{Fog}$ de la sémantique SD_{Fog} du langage Fog-DSML est décrit par un ensemble d'agents tels que définis par le formalisme BiAgents*. Un système Fog peut être constitué de plusieurs types d'agents : agents capteurs, agents de contrôle de communication, agents de contrôle du Fog et agents de contrôle du Cloud. Chaque agent dans la sémantique SD_{Fog} représente un agent de contrôle d'une couche de l'architecture CLA*4*Fog. Chaque agent peut contrôler l'évolution du système en observant un état dans $\mathcal{O}$ avec la fonction *obs*, analyser son observation avec la fonction *an*, prendre une décision dans $\mathcal{D}$, communiquer sa décision avec la fonction *int* en envoyant un message au $\mathbb{M}$ d'un autre agent et en déclenchant l'action adéquate dans $\mathcal{U}$ au bon moment avec la fonction *ctr*. Il peut migrer d'un hôte à un autre avec la fonction *mgrt*.

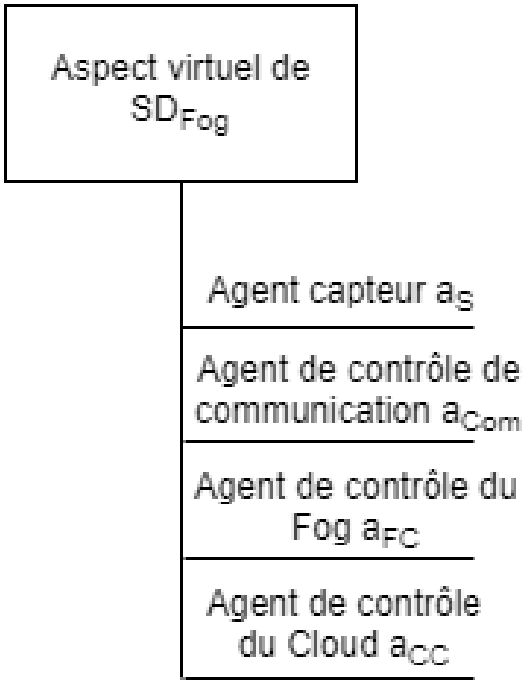


FIGURE 5.5 – Types d'agents de la sémantique SD_{Fog}

— a_S est le type d'agents responsables de la gestion des différents capteurs impliqués dans le système Fog. Il permet la capture des données et leur analyse et se coordonne avec les autres agents pour l'orientation de ces données.

— a_{Com} est le type d'agents responsable du contrôle de la communication entre les couches IoT, il permet le formatage des données et leur échange à haut niveau

— a_{FC} est le type d'agents de contrôle des nœuds Fog. Il contrôle les différents services fournis par les nœuds Fog et les oriente vers l'actionneur correspondant.

— a_{CC} est le type d'agents de contrôle des services Cloud. Il gère les services fournis par les serveurs Cloud et les oriente vers l'actionneur adéquat.

Définition 28 La sémantique de l'entité virtuelle $\mathcal{A}^*_{Fog} \in SD_{Fog}$ d'un système Fog est définie formellement par le tuple

$$\mathcal{A}^* = \{a_S, a_{Com}, a_{FC}, a_{CC}\}$$

où chaque agent peut être dans un état de $\mathbb{S}$. Les agents capteurs a_S observent les états initiaux d'un système Fog, les agents a_{Com} interagissent le plus avec les agents a_{FC} et a_{CC} et ces deux derniers contrôlent les applications dans le système Fog.

Exemple

Dans le système Fog de reconnaissance de voix, nous défiissons un agent de chaque type : $\mathcal{A} = \{a_{IoT}, a_{Com}, a_{FC}, a_{CC}\}$. Cependant, nous nous concentrons sur deux agents uniquement (voir Tableau 5.4)afin de familiariser le lecteur avec les notations les plus pertinentes de l'aspect virtuel de la sémantique du langage Fog-DSML :

TABLE 5.4 – Définitions formelles des agents a_IOT et a_FC

États S_i		a_IoT	a_FC
Observation	$\mathcal{O}$	$\mathcal{O}_{IoT} = \{B_{exF}, B_{exO}\}$	$\mathcal{O}_{FC} = \{B_{exF}, B_{exA}\}$
	obs	$obs_{IoT}(B_{exF}, 2) = obs_1$, $obs_{IoT}(B_{exO}, porte) = obs_2$	$obs_{FC}(B_{exF}, noeudFog) = obs_1$, $obs_{FC}(B_{exA}, alarme) = obs_2$
Migration	$host$	$host_{IoT} = 2$	$host_{FC} = noeudFog$
	$mgrt$	$mgrt_{IoT}(2, obs_1) = porte$	$mgrt_{FC}(noeudFog, obs_1) = alarme$
Analyse	an	$an_{IoT}(obs_2, porte) = ouvrir$	$an_{FC}(obs_2, alarme) = alerter$
Decision	$\mathcal{D}$	$\mathcal{D}_{IoT} = \{ouvrir\}$	$\mathcal{D}_{FC} = \{alerter\}$
Interaction	int	$int_{IoT}(FC, alerter) = \emptyset$	$int_{FC}(FC, alerter) =$ "Alerte ! "
	$\mathbb{M}$	$\mathbb{M}_{IoT} = \{$"Alerte !"$\}$	$\mathbb{M}_{FC} = \emptyset$
Action	ctr	$ctr_{IoT}(ouvrir) = \{(ouvrir, porte)\}$	$ctr_{FC}(alerter) = \{(alerter, alarme)\}$
	$\mathcal{U}$	$\mathcal{U}_{IoT} = \{(ouvrir, porte)\}$	$\mathcal{U}_{FC} = \{(alerter, alarme)\}$

Nous remarquons que chaque agent gère le système de son coté et que l'agent a_{FC} prévient l'agent a_IoT en cas d'alerte.

La modélisation des structures physique et virtuelles est suffisante afin de déterminer les détails structurels d'un système Fog. Cependant, il est nécessaire de combiner ces structures afin de déterminer le comportement intelligent global du système. Dans ce qui suit, nous utilisons le concept de traces $T^*{}_{Fog}$ afin de compléter la définition de la sémantique SD_{Fog}.

5.3.3 Aspect comportemental

Nous définissons l'aspect comportemental $T^*{}_{Fog}$ de SD_{Fog} par un ensemble de traces permettant l'expression du comportement intelligent d'un système Fog. En effet, les traces telles que définies par [Perrone et al., 2011] permettent la description du comportement non guidé d'un système. Avec la définition de $T^*{}_{Fog}$ (traces des BiAgents*) il est possible de guider ce comportement afin de représenter au mieux le comportement intelligent d'un système Fog.

Selon les différents types d'agents, il existe plusieurs possibilités de définition des projections de traces. En effet, chaque agent peut passer par les états d'observation, analyse, décision, interaction action et migration. Une projection de trace* sur un agent a commence forcément ainsi : $(B_0, h) \overset{obs}{\prec} (B_0, h)$ après observation il peut y avoir plusieurs possibilités :

— $(B_0, h) \overset{obs}{\prec} (B_0, h) \overset{mgrt}{\prec} (B_0, h')$

— $(B_0, h) \overset{obs}{\prec} (B_0, h) \overset{an}{\prec} (B_0, h)$

— $(B_0, h) \overset{obs}{\prec} (B_0, h) \overset{an}{\prec} (B_0, h) \overset{RR}{\prec} (B_1, h)$

— $(B_0, h) \overset{obs}{\prec} (B_0, h) \overset{an}{\prec} (B_0, h) \overset{int}{\prec} (B_0, h) \overset{RR}{\prec} (B_1, h)$

— $(B_0, h) \overset{obs}{\prec} (B_0, h) \overset{mgrt}{\prec} (B_0, h') \overset{obs}{\prec} (B_0, h')$

— $(B_0, h) \overset{obs}{\prec} (B_0, h) \overset{an}{\prec} (B_0, h) \overset{obs}{\prec} (B_0, h)$

— $(B_0, h) \overset{obs}{\prec} (B_0, h) \overset{an}{\prec} (B_0, h) \overset{RR}{\prec} (B_1, h) \overset{obs}{\prec} (B_1, h)$

— $(B_0, h) \overset{obs}{\prec} (B_0, h) \overset{an}{\prec} (B_0, h) \overset{int}{\prec} (B_0, h) \overset{RR}{\prec} (B_1, h) \overset{obs}{\prec} (B_1, h)$

Nous définissons les différents éléments composant la sémantique du langage Fog-DSML à travers la définition des dimensions physique, virtuelle et comportementale à l'aide du formalisme BiAgents*. La structure physique des BiAgents* permet de déterminer la localité, la connectivité et l'évolution des éléments physiques d'un système Fog à travers la définition des bigraphes représentant les états du système et des fonctions de transition d'un bigraphe à

l'autre. La structure virtuelle des BiAgents* précise la configuration des différents agents intelligents responsables du contrôle du système à travers leurs différentes fonctions d'analyse et d'interaction. L'unification de ces deux concepts ainsi que l'utilisation des projection de traces bigraphiques sur les agents définis permet de spécifier le comportement intelligent et guidé d'un système Fog.

Exemple

Nous continuons avec l'exemple du système de détection de voix afin d'illustrer l'aspect comportemental de la sémantique SD_{Fog}.
Nous présentons une interprétation précise des éléments constitutifs d'une trace (structures et comportements) et de leurs projections. Les événements possibles qui décorent une projection donnée sont liés aux états d'agent (Figure 5.2). La projection d'une trace sur un agent hérite de cette extension en enrichissant ces projections avec les fonctions d'analyse et d'interaction.

Comme le montre le Tableau 5.5, chaque trace définit une évolution du système Fog qui peut impliquer un ou plusieurs états dans $\mathbb{B}_{ex}$ appartenant à la structure physique $\mathcal{B}_{ex}$. Les différentes traces indiquées dans le Tableau 5.5 modélisent les différentes exécutions du système comme suit :

— La trace t_1 représente l'ouverture de la porte après reconnaissance de voix. Elle implique deux états possibles du système, B_{exF} : les portes sont fermées, B_{exO} : les portes sont ouvertes. Le passage de l'état B_{exF} à l'état B_{exO} est permis grâce à la règle de réaction *ouvrir*.

— La trace t_2 définit l'alerte en cas d'usurpation d'identité. Cette trace implique deux états possibles du système, B_{exF} : les portes sont fermées, B_{exA} : les alarmes sont déclenchées. Le passage de l'état B_{exF} à l'état B_{exA} est permis grâce à la règle de réaction *alerter*.

TABLE 5.5 – Traces modélisant le comportement du système de reconnaissance de voix

Comportement possible du système	**Définitions des traces**
Ouvrir les portes	$t_1 = B_F \overset{\text{ouvrir}}{\prec} B_O$
Déclencher le système d'alarme	$t_2 = B_F \overset{\text{alerter}}{\prec} B_A$

Le Tableau 5.6 présente les différentes projections des traces t_1 et t_2. Nous précisons que les fonctions *host* et *int* dans notre exemple sont définies par :

$int(FC, alerter) =$ ("Alerte !").
$h0(a_{FC}) = noeudFog\,;\ h1(a_{FC}) = alarme\,;\ h2(a_{IoT}) = region2\,;\ h3(a_{IoT}) = porte$

La projection de t_1 est la suivante : L'agent de contrôle du Fog a_{FC} observe le bigraph B_F depuis le nœud Fog et migre vers le microphone *mc*. Cet agent analyse le microphone et prend la décision d'appliquer la règle de réaction *alerter*. Cet agent informe l'agent gestionnaire des dispositifs a_{IoT} de sa décision. Après cette interaction, *alerter* est appliquée. Lorsque *alerter* est appliquée, l'état du système devient B_A.

TABLE 5.6 – Projetcion des traces t_1 et t_2.

Projection de t_1 sur a_{IoT}
$(B_F, h2) \overset{\text{obs}}{\prec} (B_F, h2) \overset{\text{mgrt}}{\prec} (B_F, h3) \overset{\text{obs}}{\prec} (B_F, h3) \overset{\text{an}}{\prec} (B_F, h3) \overset{\text{ouvrir}}{\prec} (B_O, h3)$
Projection de t_2 sur a_{FC}
$(B_F, h0) \overset{\text{obs}}{\prec} (B_F, h0) \overset{\text{mgrt}}{\prec} (B_F, h1) \overset{\text{obs}}{\prec} (B_F, h1) \overset{\text{an}}{\prec} (B_F, h1) \overset{\text{int1}}{\prec} (B_F, h1) \overset{\text{alerter}}{\prec} (B_A, h1)$

5.4 Expression et vérification des propriétés

En determinant la sémantique du langage Fog-DSML, il nous est possible d'analyser le modèle résultant à travers la vérification de propriétés inhérentes aux systèmes Fog. En effet, il est possible d'exprimer des propriétés structurelles et temporelles à l'aide de logiques adéquates et de procéder à la vérification de ces dernières à travers des prouveurs de théorèmes et des model-checkers.

La logique spatiale TQL (*Tree Query Logic*) [Conforti et al., 2003] permet de décrire les propriétés de plusieurs structures. Les modèles pour cette logique incluent des structures informatiques telles que les arbres, les graphes, les objets concurrents ainsi que les calculs de processus et le calcul ambiant. Cette logique est utilisée pour raisonner sur les mises à jour éventuelles des structures arborescentes et permet également le model checking de certaines propriétés pertinentes. TQL est inspirée de la logique ambiante, qui décrit des arbres finis étiquetés non ordonnés appelés *information trees*, notés, i.t.

Opérateurs et interprétation Cette logique permet l'utilisation des quantificateurs $\forall$ et $\exists$. Elle est basée sur quelques notations telles que :

- **0** représente un i.t vide.
- n[A] représente les i.t. à un seul bord étiquetés n et menant à un sous-arbre satisfaisant A.
- A|B signifie que le i.t. est divisible en deux i.t. satisfaisant A et B respectivement.
- **T** illustre tous les i.t.
- ¬A représente tous les i.t. qui ne satisfont pas A.
- A∧B signifie que i.t. satisfait A et B.
- A∨B représente i.t. satisfait A ou B.

Il est possible d'exprimer une formule F et la vérifier en s'assurant qu'un arbre T satisfait cette formule, $F(T \vDash F)$. Une formule F est valide si et seulement si, non F n'est pas satisfaisable, on note $\neg(\exists T.T \vDash \neg F)$. **Exemple** Nous pouvons représenter l'aspect physique du système de détection de voix graphiquement par la Figure 5.6.

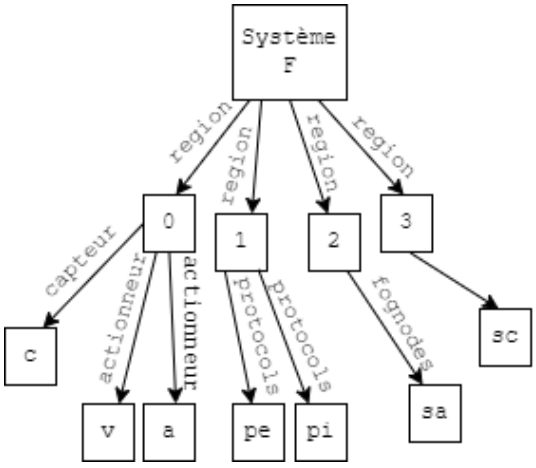


FIGURE 5.6 – Notation TQL du système de détection de voix F

TQL permet donc l'expression d'un système tel que le graphe de places d'un Bigraphe. La localité des éléments d'un système ainsi que leur connexion est vérifiable grâce à cette logique.

La logique LTL (*Linear Temporal Logic*)représente un système formel pour la description et le raisonnement qualitatif sur les changements durant le temps de la valeur de vérité des assertions. **Opérateurs et interprétations** Cette logique possède des opérateurs temporels permettant d'exprimer différentes situations :

— F $\diamond$ signifie "éventuellement".

— G $\Box$ signifie "toujours".

— X **O** exprime le prochain état.

— U veut dire "jusq'à".

Il est possible de vérifier une propriétés LTL d'un système si ce dernier est représentable sous forme d'ensemble d'états et de relations de transitions. Une formule LTL est donc vérifiée sur l'ensemble des états du système. **Exemple**

Toujours avec l'exemple du système de détection de voix, il est possible d'exprimer la propriété "*La porte est toujours fermée*". Cette propriété peut être exprimée comme suit :

$p = \Box f$ avec f un prédicat tel que $f =$ La porte est fermée. Cette assertion signifie quand dans tous les états du système, la porte est fermée, aucune voix n'est reconnue et l'état initial demeure stable (voir Figure 5.7).

FIGURE 5.7 – Propriété Gf ($\Box f$)

L'aspect hiérarchique de la syntaxe concrète de Fog-DSML ainsi que sa sémantique structurelle et dynamique nous permet d'utiliser les logiques TQL et LTL afin d'exprimer des propriétés structurelles et temporelles des systèmes Fog. Il est possible d'exprimer un système Fog sous forme d'arbre et de procéder à la vérification des propriétés structurelles et sous forme d'états pour la vérification des propriétés temporelles. Dans ce qui suit, nous exprimons les propriétés d'interopérabilité et de portabilité avec leurs dimensions structurelles et temporelles.

Définition des propriétés à vérifier

Les systèmes Fog sont des systèmes où la communication est primordiale. L'interprétation des données par les différents éléments du système et leur échange sont des critères extrêmement importants dans la conception d'un tel système. De ce fait, nous estimons que les propriétés d'interopérabilité et de portabilité des données sont d'une grande importance et qu'il est nécessaire de vérifier ces dernières dans le modèle que nous proposons.

5.4.1 Définition de l'interopérabilité

L'interopérabilité est définie selon [Noura et al., 2017] et [Noura et al., 2019] comme étant divisé en cinq types :

1. **Interopérabilité des objets :** interopérabilité d'un système intelligent, permettant l'incorporation et l'interopérabilité de systèmes hétérogènes dans une variété de protocoles et de normes de communication fournis par ces objets
2. **Interopérabilité des réseaux :** l'adressage, le routage, l'optimisation des ressources, la sécurité et la qualité du service devraient être traités par les cadres qui permettent le partage continu des messages entre les différents réseaux
3. **Interopérabilité syntaxique :** l'interopérabilité des formats, la structure de données utilisée pour partager les messages entre différents types de systèmes. Pour chaque schéma, une interface est nécessaire. Le contenu du message doit être codé jusqu'à ce qu'il puisse être transmis dans un fichier tel que XML ou JSON. Le message est codé par l'émetteur et décodé par le récepteur, la difficulté se produit lorsque les règles de décodage du récepteur sont en conflit avec les règles de codage de l'émetteur.
4. **Interopérabilité sémantique :** elle permet à des agents, programmes et applications distincts de partager des informations avec un sens précis. Il fournit un contexte aux informations contenues dans une structure syntaxique. Il peut recevoir des données d'une API supportant des formats de données tels que JSON, XML ou CSV.
5. **Interopérabilité des plateformes :** interopérabilité dans divers systèmes opérationnels, langages de programmation, structures de données, architectures, mécanismes d'accès et données. Nous devons tenir compte des différents modèles d'API et d'information pour chaque plateforme et adapter leur cadre à chacune de ces plateformes. elle facilite la communication entre les plateformes ayant des domaines distincts.

Nous définissons les propriétés d'interopérabilité syntaxique et sémantique dans un système Fog. Les formules TQL expriment les prédicats (voir Tableau 5.8) utilisés afin de définir les propriétés temporelles LTL (voir Tableau 5.7).

Table 5.7 – Propriétés d'interopérabilité d'un système Fog.

Propriété	**Définition**	**Formule LTL**
Interopérabilité syntaxique	Les données doivent être formatées après leur saisie et ce format doit être compréhensible par les nœuds de traitement du système et les actionneurs.	$syntacticalInteroperability = \diamond((b1 \vee b2) \wedge b4)$
Interopérabilité sémantique	Toute information transmise dans de nombreux types de systèmes et sous-systèmes doit être une information significative (une information significative est une information pouvant être analysée par un agent)	$semanticInteroperability = \square(a1 \wedge a2 \wedge a3 \wedge a4)$

TABLE 5.8 – Définition des prédicats utilisé pour les propriétés d'interopérabilité.

Prédicats	**Définition**	**Formule TQL**
$b1$	Le format est hébergé dans un service Fog	$F[region[2[fognodes[format]$ $]\|\ \mathbf{T}\ ]]$
$b2$	Le format est hébergé dans un service Cloud	$F[region[3[cloudservice[format]$ $]\|\ \mathbf{T}\ ]]$
$b4$	Le format est hébergé dans un service présent dans l'actionneur	$F[region[0[actuator[format]$ $]\|\ \mathbf{T}\ ]]$
$a1$	L'agent capteur analyse les informations	$F[region[0[devices[sensor[data$ $[SensorAgent[\mathbf{0}]]]\|\ \mathbf{T}\ ]]\|\mathbf{T}]$
$a2$	L'agent de contrôle de la sécurité de la communication analyse le format de l'information	$F[region[1[protocols[format$ $[ComAgent[\mathbf{0}]]\|\ \mathbf{T}\ ]]\|\mathbf{T}]$
$a3$	L'agent de contrôle Fog analyse les informations	$F[region[2[fognodes[fogservice$ $[FogAgent[\mathbf{0}]]\|\ \mathbf{T}\ ]]\|\mathbf{T}]$
$a4$	L'agent de contrôle Cloud analyse les informations	$F[region[3[Cloudserver[Cloudservice$ $[CloudAgent[\mathbf{0}]]\|\ \mathbf{T}\ ]]\|\mathbf{T}]$

5.4.2 Définition de la portabilité

La portabilité est la capacité de déplacer des entités logicielles entre différentes plateformes d'exécution sans avoir à les réécrire partiellement ou entièrement.

La propriété de portabilité peut être exprimée sous forme de formule LTL tel que présentée dans le Tableau 5.9 à l'aide des prédicats exprimés en TQL (voir Tableau 5.10).

TABLE 5.9 – Propriétés de portabilité dans un système Fog.

Propriété	**Définition**	**Formule LTL**
Portabilité des données	Les données capturées doivent pouvoir circuler entre les différents éléments du système	$portability = \Box(q1 \vee q2 \vee q3 \vee q4 \vee q5)$

Les prédicats utilisés sont exprimés selon la logique TQL dans le Tableau 5.10

TABLE 5.10 – Définition des prédicats utilisés pour l'expression de la propriété de portabilité.

Prédicats	**Définition**	**Formule TQL**
$q1$	Les données sont hébergées dans un capteur	$F[region[0[devices[sensor[data]$ $]\|\ \mathbf{T}\ ]]\|\mathbf{T}]$
$q2$	Les données sont hébergées dans un protocole de communication	$F[region[1[protocol[data]$ $]\|\ \mathbf{T}\ ]\|\mathbf{T}]$
$q3$	Les données sont hébergées dans un nœud Fog	$F[region[2[fognodes[data]$ $]\|\ \mathbf{T}\ ]\|\mathbf{T}]$
$q4$	Les données sont hébergées dans un système Cloud	$F[region[3[cloudservices[data]$ $]\|\ \mathbf{T}\ ]\|\mathbf{T}]$
$q5$	Les données sont hébergées dans un actionneur	$F[region[0[devices[actuator[data]$ $]\|\ \mathbf{T}\ ]]\|\mathbf{T}]$

5.5 Conclusion

Nous avons présenté dans ce chapitre la sémantique du langage proposé Fog-DSML. Cette sémantique est basée sur un modèle formel BiAgents* avec ses dimensions physique et virtuelle

en plus de la spécification des aspects comportementaux à l'aide des traces d'évolutions et ses projections sur les agents définis.
BiAgents* est une extension des BiAgents [Pereira et al., 2012] dédiée à la modélisation systèmes composés d'éléments hétérogènes dispersés géographiquement, connectés, intelligents et communicants.
Nous avons insisté sur l'aspect virtuel de la sémantique SD_{Fog}, la structure physique par rapport au modèle initial BiAgent n'a pas été changé. Nous avons défini les différents états par lesquelles les entités virtuelles (Agents) passent afin de gérer un système Fog et nous avons précisé comment chaque agent perçoit le système et agit en conséquence.
L'aspect comportementale de SD_{Fog} est défini à partir des différentes opérations identifiées, règles de réactions ainsi que les changement d'états des agents. Les différentes stratégies d'exécution décrivant une logique pour la gestion d'un système Fog et le contrôle de ses services sont définies sous forme de traces* T^* (traces de BiAgents*). Le choix de la bonne stratégie se fait selon les données et les différentes décisions des agents responsables de la gestion du système. Nous avons introduit les logiques spatiale et temporelle TQL et LTL respectivement et nous avons exprimé les propriétés d'interopérabilité et de portabilité selon leurs dimensions structurelles et temporelles dans le but d'analyser un système Fog à travers la vérification de ces propriétés. Dans le prochain chapitre, nous procédons à l'exécution du modèle proposé ainsi que la vérification des propriétés LTL définies.

Chapitre 6

Évaluation par cas d'étude : Système Fog LIS

Sommaire

6.1 Introduction

Les BiAgents* représentent un excellent moyen pour modéliser les différents aspects physiques, virtuels et comportementaux des systèmes Fog. Néanmoins, il n'existe aucun outil développé autour de ce formalisme, qui permet d'exprimer ses différents concepts en termes de comportement des BRS et d'actions d'agents. Il existe des outils développés autours des BRS tels que BigMC [Perrone et al., 2012], Bigrapher [Sevegnani and Calder, 2016] et BPLTool [Højsgaard and Glenstrup, 2011] mais qui ne conviennent pas à la vérification d'un modèle bigraphique. Par exemple, BigMC qui est le seul model-checker dédié aux BRS ne permet pas l'expression des propriété complexes et ne supporte pas leur logique de typage. Bigred [Faithfull et al., 2013] permet d'éditer et de visualiser uniquement des systèmes réactifs bigraphiques.

Dans le cadre de cette thèse, nous optons pour le langage Maude stratégie comme l'alternative la plus adéquate à ces outils. Ainsi que présenté dans le Chapitre 3, Maude stratégie est une extension du langage Maude qui est un langage de spécification formelle de haut niveau, basé sur la logique de réécriture et la logique équationnelle. Ce langage représente la meilleure option permettant de prendre en charge les aspects physiques, virtuels et comportementaux du modèle BiAgent* défini pour modéliser les systèmes Fog. De plus, l'environnement Maude et son extension permet de valider les spécifications obtenues et analyser le comportement des systèmes Fog à travers un nombre de techniques développées autour de ce langage.

Dans ce chapitre, nous décrivons un système Fog réel d'inspection des bagages dans un terminal d'aéroport LIS (*Luggage Inspection system*) en utilisant le langage Fog-DSML. Nous instancions d'abord la syntaxe concrète du langage (l'architecture CLA*4*Fog) selon le cas d'étude avant de passer à la détermination de sa sémantique selon ses dimensions physique, virtuelle et comportementale. Ensuite, nous passons à l'encodage du modèle sémantique vers le langage Maude stratégie permettant ainsi l'exécution des différents scénarios du système. Cette transformation permet aussi la vérification des propriétés d'interopérabilité syntaxique et sémantique ainsi que la portabilité des données dans ce système vérifiant ainsi que le modèle proposé est correctement défini.

6.2 Description informelle de LIS : *Luggage Inspection System*

Dans cette section, nous instancions l'architecture CLA*4*Fog selon une étude de cas concrète. Comme l'ont déclaré Rhonda Dirvin et Matt Vasey (membres d'OpenFog Consortiom), certains systèmes IoT illustrent mieux que d'autres l'utilité du Fog computing. Afin d'assurer la sécurité publique, des caméras vidéo sont utilisées dans plusieurs secteurs de la vie quotidienne : parkings, grands bâtiments et divers espaces publics et privés. Avec l'aide d'une architecture Fog, le traitement vidéo est intelligemment partitionné entre les nœuds co-situés avec les caméras et le Cloud. Cela permet le suivi en temps réel, la détection d'anomalies et la collecte d'informations à partir de données saisies à de longs intervalles de temps. Notre choix est de modéliser un système de vidéo surveillance pour les terminaux d'aéroports. Cette dernière permet de lutter contre le terrorisme, de surveiller en temps réel les différentes zones, de faire des analyses vidéo avancées, et plus encore. En utilisant une architecture Fog, la transmission vidéo est intelligemment partitionnée entre les nœuds Fog situés entre les dispositifs impliqués et le Cloud. Cela permet d'établir des rapports en temps réel, d'identifier les anomalies et de recueillir des informations à partir des données recueillies sur de longues périodes. La Figure 6.1 illustre la configuration générique inspirée de [STAC, 2010] de tout aéroport et de ses éventuels services opérationnels.

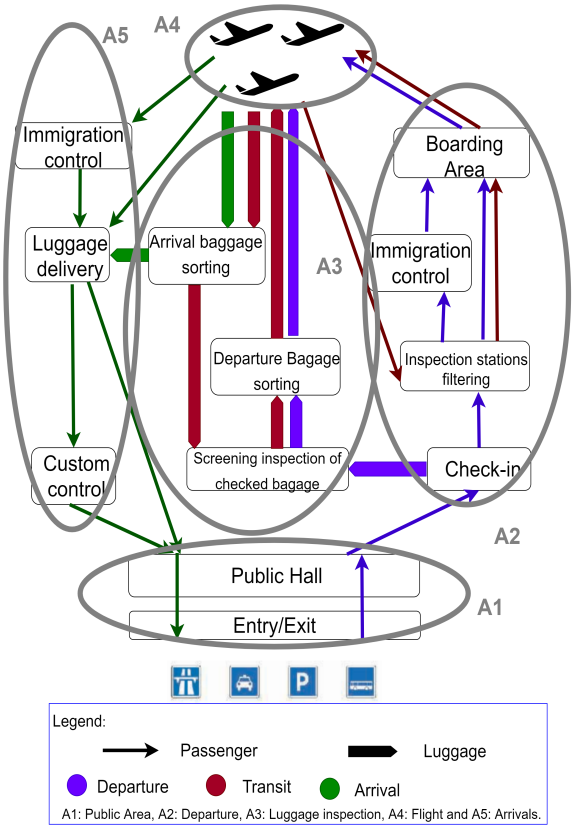


FIGURE 6.1 – Zones de surveillance d'un terminal d'aéroport

Afin d'assurer la sécurité de l'aéroport, un système de surveillance devrait être équipé de caméras à toutes les entrées et sorties pour enregistrer la vidéo de toute personne passant par l'aéroport. Le système permettrait de suivre toutes les zones ouvertes afin d'obtenir une image globale continue de l'activité de l'aéroport. Les caméras d'enregistrement des compagnies aériennes obtiennent des photos des départs et des procédures des passagers. Le système devrait surveiller les points de contrôle de sécurité pour déceler les encombrements, les comportements criminels, les confrontations et les risques éventuels pour les passagers. La surveillance des tramways ou autres moyens de transport utilisés pour relier un tronçon de l'aéroport à un autre doit être placée. Les terminaux et portes individuels doivent être vérifiés pour les comportements irréguliers en phase d'embarquement et pour les voyageurs suspects. Les caméras dans les zones de réclamation des bagages devraient être en mesure de dissuader la fraude et de surveiller les objets potentiellement dangereux. Les couloirs et toutes les zones utilisées pour passer par les terminaux doivent également être suivis. Les caméras offrent une vue sur les zones réglementées et veillent à ce que seuls les travailleurs autorisés y aient accès. Un aéroport peut être divisé en 5 zones de surveillance, comme le montre la Figure 6.1. A1 : Espace public, A2 : Départ, A3 : Inspection des bagages, A4 : Vols et A5 : Arrivées.

Dans le cadre de cette thèse, nous nous intéressons à l'inspection des bagages (A3). Dans un terminal spécifique, après l'enregistrement, un agent de vol place un code à barres dans chaque sac avant de se rendre aux bacs souterrains où le mouvement des bagages est surveillé. Le système de surveillance vérifie :

1. Si les lignes sont occupées (en comptant le nombre de plateaux)
2. Si les codes à barres sont identifiables

3. Si les sacs sont acceptables (les sacs inacceptables sont en forme ronde, ont de longues sangles ou sont fermés avec des cordes)

La première étape de la modélisation de l'étude de cas LIS consiste à décrire son architecture.

6.3 Spécification de LIS dans Fog-DSML

6.3.1 Architecture CLA*4*Fog pour LIS

Nous instancions l'architecture proposée en définissant tous les éléments intégrés dans ses quatre couches :

1. La couche dispositifs contient trois capteurs : Scanner de bagages, Scanner de code barres et une caméra en plus d'un actionneur : un écran.
2. La couche de communication comprend des protocoles internes et externes qui formatent des données aux formats JSON et XML respectivement.
3. La couche Fog comprend deux principaux nœuds : les systèmes de reconnaissance et d'alerte et tous les agents hébergés initialement dans la couche Fog. Dans ce système, nous définissons trois agents de capteurs (un agent pour chaque capteur), un agent de contrôle Fog, un agent interne de contrôle de la sécurité des communications, un agent externe de contrôle de la sécurité des communications et enfin un agent de contrôle du Cloud. Chaque agent gère un élément physique de l'architecture et par conséquent, participe à la gestion générale du système.
4. La couche Cloud contient un service Cloud chargé d'analyser et de stocker les données.

Il existe différentes interactions entre les éléments de l'architecture (physique et virtuelle). Les éléments physiques partagent des données afin de les analyser, les formater ou les stocker et les éléments virtuels communiquent entre eux afin d'orchestrer le comportement du système.

Les différents capteurs collectent des données du monde physique et les différents agents de capteurs migrent vers leurs interfaces, puis observent ces données, les analysent. Si jamais ces agents détectent une donnée anormale, (pouvant représenter une tentative de piratage), cette donnée est supprimée. Sinon, l'agent capteur en question envoie à l'agent de contrôle de sécurité de communication interne ces données. Ce dernier décide s'il y a une anomalie ou non. Si à partir de la caméra il y a plus ou moins de 10 000 plateaux comptés, ou le code à barres détecté à partir du scanner de code à barres est un code à barres identifiable, ou un sac dangereux est détecté (à partir du scanner des bagages), il y a une anomalie. l'agent de contrôle de la sécurité des communications internes formate les données captées en utilisant les protocoles de communication internes définis. Ensuite, il communique avec l'agent de contrôle Fog et envoie les données formatées aux nœuds Fog. L'agent de contrôle Fog traite les données formatées. Avec l'aide des services Fog d'alerte et de reconnaissance, le détecteur d'anomalie (l'agent) avertit les utilisateurs en affichant cet avertissement sur les écrans.

En l'absence d'anomalie, l'agent de détection concerné transmet les données recueillies à l'agent externe de sécurité des communications. Ce dernier recueille les données, les formate à l'aide du protocole de communication externe et les envoie au Service Cloud d'analyse et de stockage. L'agent de sécurité de communication externe interagit ensuite avec l'agent Cloud et l'informe des données envoyées au système d'analyse. Après cela, l'agent Cloud affichera la confirmation de stockage sur les écrans.

Nous précisons que la vue d'interaction de l'architecture est celle qui permet le bon fonctionnement de l'architecture, apportant à l'exécution un aspect intelligent et contrôlé.

6.4 Sémantique de LIS

Aspect physique

Nous illustrons à présent les notations BiAgents* pour définir graphiquement la structure physique de ce système comme le montre la Figure 6.2. Le bigraphe est composé de quatre régions. Chaque région représente une couche de l'architecture CLA*4*Fog. 0 couche IoT, 1 couche communication, 2 couche Fog et 3 couche Cloud.

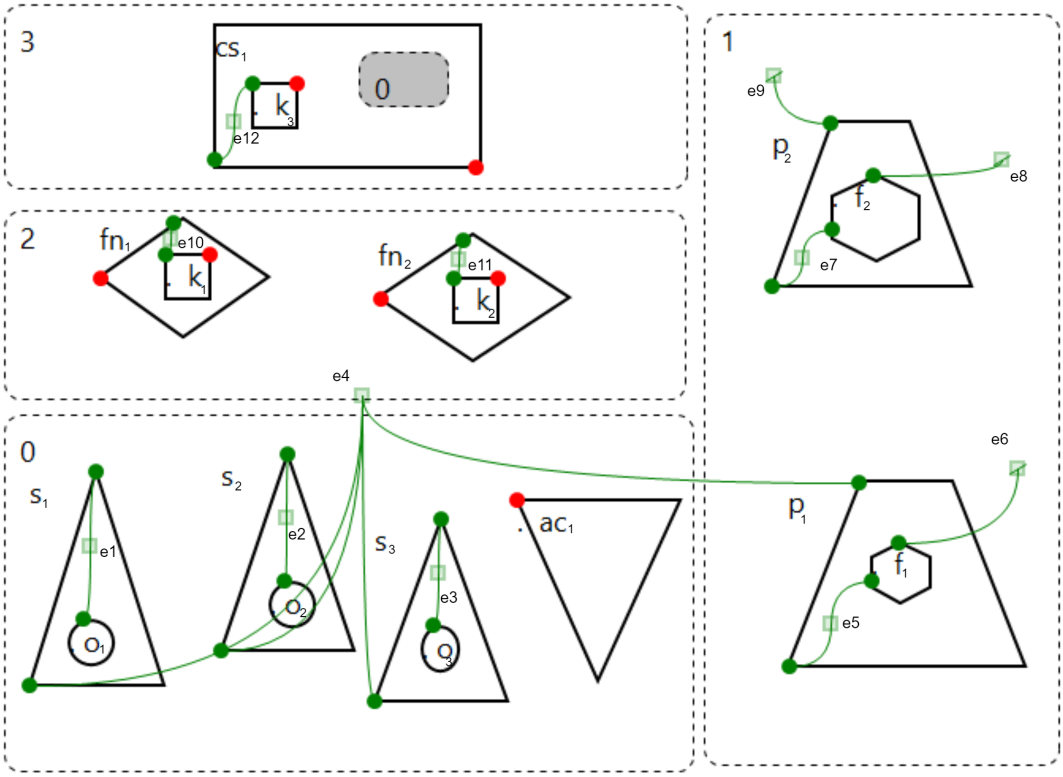


FIGURE 6.2 – Représentation graphique de $\mathcal{B}_F$

Nous définissons précisément la structure physique du cas d'étude LIS par :

$$\mathcal{B}_F = (\mathbb{B}, \mathcal{R}, \mathbb{U}, B_0, F)$$

— L'ensemble de tous les états physiques possibles du système modélisé par des bigraphes constituant $\mathbb{B} = \{B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10}, B_{11}, B_{12}, B_{13}\}$.

— L'ensemble des règles de réaction permettant à la partie physique du système d'évoluer d'un état à un autre $\mathcal{R} = \{R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8, R_9, R_{10}, R_{11}, R_{12}, R_{13}, R_{14}\}$.

— L'ensemble des actions définissant l'application de chaque règle de réaction en fonction d'un nœud spécifique $\mathbb{U} = \{(R_1, s), (R_2, o), (R_3, f), (R_4, k), (R_5, ac), (R_6, k), (R_7, ac), (R_8, s), (R_9, o), (R_{10}, f), (R_{11}, o), (R_{12}, ac), (R_{13}, p), (R_{14}, p)\}$.

— L'état initial du système est modélisé par un bigraphe $B_0 = (V, E, ctrl, prnt, link) : I \to J$ avec V l'ensemble de tous les nœuds, E l'ensemble de tous les hyperarcs, la fonction $ctrl$ représente le nombre de ports de chaque nœud et la fonction $prnt$ associe à chaque nœud un parent (nœud ou région) (Figure 6.2). La fonction $link$ relie chaque port de nœud à un hyperarc. Par exemple e_1 est relié au port unique de o_1 et au deuxième port de s_1. I

est l'interface interne de B_0 $\langle$ nombre de sites, ensemble de noms internes $\rangle = \langle 1, \emptyset \rangle$, J est l'interface externe $\langle$ nombre de régions, ensemble des noms externes$\rangle = \langle 4, \emptyset \rangle$.

— La fonction de transition modélise le passage d'un état à un autre en associant un bigraphe à un bigraphe de départ et un contrôle. $F(B_0, (R_1, s)) = B_1$; $F(B_1, (R_2, o)) = B_2$; $F(B_2, (R_3, f)) = B_3$; $F(B_3, (R_4, k)) = B_4$; $F(B_4, (R_5, ac)) = B_5$; $F(B_3, (R_6, k)) = B_6$; $F(B_6, (R_7, ac)) = B_7$; $F(B_0, (R_8, s)) = B_8$; $F(B_8, (R_9, o)) = B_9$; $F(B_9, (R_{10}, f)) = B_{10}$; $F(B_{10}, (R_{11}, o)) = B_{11}$; $F(B_{11}, (R_{12}, ac)) = B_{12}$; $F(B_1, (R_{13}, cl)) = B_{13}$; $F(B_8, (R_{14}, cl)) = B_{13}$.

Nous détaillons les différentes règles de réaction définies dans l'ensemble $\mathcal{R}$ dans le Tableau 6.1.

TABLE 6.1 – Règles de réaction de $\mathcal{B}_F$ associées à LIS.

Couches	**Description de la règle**	**Forme algébrique**
IoT → Communication	Envoyer les données aux protocoles internes	$R1 \stackrel{\text{def}}{=} ((s.o)\|id \parallel (p_1.f_1)\|(p_2.f_2)\|id) \rightarrow (s\|id) \parallel (p_1.(f_1\|o))\|(p_2.f_2)\|id$
	Envoyer les données aux protocoles externes	$R8 \stackrel{\text{def}}{=} ((s.o)\|id \parallel (p_1.f_1)\|(p_2.f_2)\|id) \rightarrow (s\|id) \parallel (p_1.f_1)\|(p_2.(f_2\|o))\|id$
Communication	Supprimer les données de la couche de communication interne	$R13 \stackrel{\text{def}}{=} (s\|id) \parallel (p_1.(f_1\|o))\|(p_2.f_2)\|id \rightarrow (s\|id) \parallel (p_1.f_1)\|(p_2.f_2)\|id$
	Supprimer les données de la couche de communication externe	$R14 \stackrel{\text{def}}{=} (s\|id) \parallel (p_1.f_1)\|(p_2.(f_2\|o))\|id \rightarrow (s\|id) \parallel (p_1.f_1)\|(p_2.f_2)\|id$
	Formater les données selon les protocoles de communication internes	$R2 \stackrel{\text{def}}{=} (p_1.(f_1\|o))\|(p_2.f_2)\|id \rightarrow (p_1.(f_1.o))\|(p_2.f_2)\|id$
	Formater les données selon les protocoles de communication externes	$R9 \stackrel{\text{def}}{=} (p_1.f_1)\|(p_2.(f_2\|o))\|id \rightarrow (p_1.f_1)\|(p_2.(f_2.o))\|id$
Communication → Fog	Envoyer les données formatées à la couche Fog	$R3 \stackrel{\text{def}}{=} (p_1.(f_1.o))\|id \parallel (fn_1.k_1)\|(fn_2.k_2)\|id \rightarrow p_1\|id \parallel (fn_1.k_1)\|(fn_2.k_2)\|(f_1.o)\|id$
Fog	Créer une instruction dans le système de reconnaissance	$R4 \stackrel{\text{def}}{=} (fn_1.k_1)\|(fn_2.k_2)\|(f_1.o)\|id \rightarrow (fn_1.(k_1.(f_1.o)))\|(fn_2.k_2)\|id$
	Créer une instruction dans le système d'alerte	$R6 \stackrel{\text{def}}{=} (fn_1.k_1)\|(fn_2.k_2)\|(f_1.o)\|id \rightarrow (fn_1.k_1)\|(fn_2.(k_2.(f_1.o)))\|id$
Fog → IoT	Envoyer une instruction depuis le système de reconnaissance vers l'actuateur	$R5 \stackrel{\text{def}}{=} ac_1\|id \parallel fn_1.(k1.(f_1.o)))\|(fn_2.k2)\|id \rightarrow (ac_1.(k1.(f_1.o)))\|id \parallel fn_1\|(fn_2.k2)\|id$
	Envoyer une instruction depuis le système d'alerte vers l'actuateur	$R7 \stackrel{\text{def}}{=} ac_1\|id \parallel (fn_1.k1)\|(fn_2.(k2.(f_1.o)))\|id \rightarrow (ac_1.(k_2.(f_1.o)))\|id \parallel (fn_1.k_1)\|fn_2\|id$
Communication → Cloud	Envoyer les données formatées à la couche Cloud	$R10 \stackrel{\text{def}}{=} (p_2.(f_2.o))\|id \parallel (cs_1.(k_3\|d1))\|id \rightarrow p_2\|id \parallel (cs_1.(k_3\|d1)\|(f_2.o))\|id$
Cloud	Créer une instruction dans le système d'analyse et de stockage	$R11 \stackrel{\text{def}}{=} (cs_1.(k_3\|d1)\|(f_2.o))\|id \rightarrow (cs_1.((k_3.(f_2.o))\|d1))\|id$
Cloud → IoT	Envoyer une instruction depuis le Cloud vers l'actuateur	$R12 \stackrel{\text{def}}{=} ac_1\|id \parallel (cs_1.((k_3.(f_2.o))\|d1))\|id \rightarrow (ac_1.(k_3.(f_2.o)))\|id \parallel (cs_1.d1)\|id$

Aspect virtuel

Dans le système LIS, il y a quatre types d'agents : le contrôle de capteur, le contrôle Fog, le contrôle de communication et le contrôle Cloud. Selon le système Fog modélisé, il peut y avoir de nombreuses instanciations de chaque type d'agent pour que chacun se concentre sur une tâche spécifique. La partie virtuelle de la modélisation BiAgent* du système LIS est définie comme suit :
$\mathcal{A}_F = a_{s1}, a_{s2}, a_{s3}, a_{FC}, a_{ICom}, a_{ECom}, a_{CC}$:
Trois agents de contrôle de capteur, un agent de contrôle Fog, deux agents de contrôle de communication et un agent de contrôle Cloud.

La configuration initiale du système considéré est matérialisée par : les différents capteurs contenant des données capturées ; les différents formats s'appuyaient sur les protocoles de communication et les actions éventuelles générées dans les nœuds Fog et les applications Cloud et les agents multiples définis dans la couche Fog. La forme algébrique du système Fog F modélisant le système LIS en se concentrant sur sa localité (c'est-à-dire le graphe de place) est noté comme suit : $F \stackrel{\text{def}}{=} ((s_1.o_1)|(s_2.o_2)|(s_3.o_3)|ac_1) \parallel (fn_1.k_1)|(fn_2.k_2)|a_{s1}|a_{s2}|a_{s3}|a_{FC}|a_{ICom}|a_{ECom}|a_{CC}) \parallel (p_1.f_1)|(p_2.f_2) \parallel (cs_1.(k_3|d0))$.

Les différents agents considérés dans ce système sont définis comme suit :

Table 6.2 – L'agent capteur

$a_{S1} = (\mathcal{O}_{S1}, obs_{S1}, host_{S1}, mgrt_{S1}, an_{S1}, \mathcal{D}_{S1}, int_{S1}, \mathbb{M}_{S1}, ctr_{S1}, \mathcal{U}_{S1})$	
Observation	$\mathcal{O}_{S1} = \{B_0, B_1\}$
	$obs_{S1}(B_0, 1) = obs_1, obs_{S1}(B_1, o_1) = obs_2, obs_{S1}(B_8, o_1) = obs_3$
Migration	$host_{S1} = region1$
	$mgrt_{S1}(region1, obs_1) = o_1$; $mgrt_{S1}(o_1, obs_2) = region1$; $mgrt_{S1}(o_1, obs_3) = region1$
Analyse	$an_{S1}(obs_1, o_1) = R_1$ or R_8
Décision	$\mathcal{D}_{S1} = \{R_1, R_8\}$
Interaction	$int_{S1}(S1, R_1) =$"send to Fog", $int_{S1}(S1, R_8) =$"send to Cloud"
	$\mathbb{M}_{S1} = \emptyset$
Action	$ctr_{S1}(R_1) = (R_1, s_1)$; $ctr_{S1}(R_8) = (R_8, s_1)$
	$\mathcal{U}_{S1} = \{(R_1, s_1), (R_8, s_1)\}$

Table 6.3 – L'agent capteur a_{S2}

$a_{S2} = (\mathcal{O}_{S2}, obs_{S2}, host_{S2}, mgrt_{S2}, an_{S2}, \mathcal{D}_{S2}, int_{S2}, \mathbb{M}_{S2}, ctr_{S2}, \mathcal{U}_{S2})$	
Observation	$\mathcal{O}_{S2} = \{B_0, B_1\}$
	$obs_{S2}(B_0, 1) = obs_1, obs_{S2}(B_1, o_2) = obs_2, obs_{S2}(B_8, o_2) = obs_3$
Migration	$host_{S2} = region1$
	$mgrt_{S2}(region1, obs_1) = o_2$; $mgrt_{S2}(o_2, obs_2) = region1$; $mgrt_{S2}(o_2, obs_3) = region1$
Analyse	$an_{S2}(obs_1, o_2) = R_1$ or R_8
Décision	$\mathcal{D}_{S2} = \{R_1, R_8\}$
Interaction	$int_{S2}(S2, R_1) =$"send to Fog", $int_{S2}(S2, R_8) =$"send to Cloud"
	$\mathbb{M}_{S2} = \emptyset$
Action	$ctr_{S2}(R_1) = (R_1, s_2)$; $ctr_{S2}(R_8) = (R_8, s_2)$
	$\mathcal{U}_{S2} = \{(R_1, s_2), (R_8, s_2)\}$

TABLE 6.4 – L'agent capteur a_{S3}

$a_{S3} = (\mathcal{O}_{S3}, obs_{S3}, host_{S3}, mgrt_{S3}, an_{S3}, \mathcal{D}_{S3}, int_{S3}, \mathbb{M}_{S3}, ctr_{S3}, \mathcal{U}_{S3})$	
Observation	$\mathcal{O}_{S3} = \{B_0, B_1\}$
	$obs_{S3}(B_0, 1) = obs_1, obs_{S3}(B_1, o_3) = obs_2, obs_{S3}(B_8, o_3) = obs_3$
Migration	$host_{S3} = region1$
	$mgrt_{S3}(region1, obs_1) = o_3$; $mgrt_{S3}(o_3, obs_3) = region1$; $mgrt_{S3}(o_3, obs_3) = region1$
Analyse	$an_{S3}(obs_1, o_3) = R_1$ or R_8
Décision	$\mathcal{D}_{S3} = \{R_1, R_8\}$
Interaction	$int_{S3}(S3, R_1) =$"send to Fog", $int_{S3}(S3, R_8) =$"send to Cloud"
	$\mathbb{M}_{S3} = \emptyset$
Action	$ctr_{S3}(R_1) = (R_1, s_3)$; $ctr_{S3}(R_8) = (R_8, s_3)$
	$\mathcal{U}_{S3} = \{(R_1, s_3), (R_8, s_3)\}$

TABLE 6.5 – L'agent de contrôle de communication $ICom$

$a_{ICom} = (\mathcal{O}_{ICom}, obs_{ICom}, host_{ICom}, mgrt_{ICom}, an_{ICom}, \mathcal{D}_{ICom}, int_{ICom}, \mathbb{M}_{ICom}, ctr_{ICom}, \mathcal{U}_{ICom})$	
Observation	$\mathcal{O}_{ICom} = B_1, B_3, B_{13}$
	$obs_{ICom}(B_1, 1) = obs_1, obs_{ICom}(B_3, f_1) = obs_2, obs_{ICom}(B_{13}, p_1) = obs_3$
Migration	$host_{ICom} = region1$
	$mgrt_{ICom}(region1, obs_1) = f_1$; $mgrt_{ICom}(f_1, obs_2) = region1$; $mgrt_{ICom}(p_1, obs_3) = region1$
Analyse	$an_{ICom}(obs_1, f_1) = R_2$ or R_{13}
Décision	$\mathcal{D}_{ICom} = \{R_2, R_{13}\}$
Interaction	$int_{ICom}(ICom, R_2) =$ "sending to Fog"
	$\mathbb{M}_{ICom} = \{$"send to Fog"$\}$
Action	$ctr_{ICom}(R_2) = \{(R_2, f_1), (R_3, f_1)\}$; $ctr_{ICom}(R_{13}) = (R_{13}, p_1)$
	$\mathcal{U}_{ICom} = \{(R_2, f_1), (R_3, f_1), (R_{13}, p_1)\}$

TABLE 6.6 – L'agent de contrôle de communication $ECom$

$a_{ECom} = (\mathcal{O}_{ECom}, obs_{ECom}, host_{ECom}, mgrt_{ECom}, an_{ECom}, \mathcal{D}_{ECom}, int_{ECom}, \mathbb{M}_{ECom}, ctr_{ECom}, \mathcal{U}_{ECom})$	
Observation	$\mathcal{O}_{ECom} = B_8, B_{10}, B_{13}$
	$obs_{ECom}(B_8, region1) = obs_1, obs_{ECom}(B_{10}, f_2) = obs_2, obs_{ECom}(B_{13}, f_2) = obs_3$
Migration	$host_{ECom} = region1$
	$mgrt_{ECom}(region1, obs_1) = f_2$; $mgrt_{ECom}(f_2, obs_2) = region1$; $mgrt_{ECom}(f_2, obs_3) = region1$
Analyse	$an_{ECom}(obs_2, f_1) = R_4$, $an_{ECom}(obs_3, f_2) = R_6$
Décision	$\mathcal{D}_{ECom} = \{R_9, R_{14}\}$
Interaction	$int_{ECom}(ECom, R_9) =$ "sending to Cloud"
	$\mathbb{M}_{ECom} = \{$"send to Cloud"$\}$
Action	$ctr_{ECom}(R_9) = \{(R_9, o), (R_{10}, f_2)\}$; $ctr_{ECom}(R_{14}) = (R_{14}, p_2)$
	$\mathcal{U}_{ECom} = \{(R_9, O), (R_{10}, f_2), (R_{14}, p_2)\}$

TABLE 6.7 – L'agent de contrôle du Fog a_{FC}

$a_{FC} = (\mathcal{O}_{FC}, obs_{FC}, host_{FC}, mgrt_{FC}, an_{FC}, \mathcal{D}_{FC}, int_{FC}, \mathbb{M}_{FC}, ctr_{FC}, \mathcal{U}_{FC})$	
Observation	$\mathcal{O}_{FC} = B_3, B_5, B_7$
	$obs_{FC}(B_3, 1) = obs_1, obs_{FC}(B_5, f_1) = obs_2, obs_{FC}(B_7, f_2) = obs_3$
Migration	$host_{FC} = region1$
	$mgrt_{FC}(region1, obs_1) = f_1\,;\, mgrt_{FC}(f_1, obs_2) = region1\,;\, mgrt_{FC}(f_2, obs_3) = region1$
Analyse	$an_{FC}(obs_2, f_1) = R_4,\ an_{FC}(obs_3, f_2) = R_6$
Décision	$\mathcal{D}_{FC} = \{R_4, R_6\}$
Interaction	$int_{FC}() = \emptyset$
	$\mathbb{M}_{FC} = \{$"sending to Fog"$\}$
Action	$ctr_{FC}(R_4) = \{(R_4, k_1), (R_5, ac_1)\}\,;\, ctr_{FC}(R_6) = \{(R_6, k_2), (R_7, ac_1)\}$
	$\mathcal{U}_{FC} = \{(R_4, k_1), (R_5, ac_1), (R_6, k_2), (R_7, ac_1)\}$

TABLE 6.8 – L'agent de contrôle du Cloud a_{CC}

$a_{CC} = (\mathcal{O}_{CC}, obs_{CC}, host_{CC}, mgrt_{CC}, an_{CC}, \mathcal{D}_{CC}, int_{CC}, \mathbb{M}_{CC}, ctr_{CC}, \mathcal{U}_{CC})$	
Observation	$\mathcal{O}_{CC} = \{B_{10}, B_{12}\}$
	$obs_{CC}(B_{10}, region1) = obs_1, obs_{CC}(B_{12}, k_3) = obs_2$
Migration	$host_{CC} = region1$
	$mgrt_{CC}(region1, obs_1) = k_3\,;\, mgrt_{CC}(k_3, obs_2) = region1$
Analyse	$an_{CC}(obs_1, k_3) = R_{11}$
Décision	$\mathcal{D}_{CC} = \{R_{11}\}$
Interaction	$int_{CC}() = \emptyset$
	$\mathbb{M}_{CC} = \{$"sending to Cloud"$\}$
Action	$ctr_{CC}(R_{11}) = (R_{11}, k_3); ctr_{CC}(R_{12})(R_{12}, ac_1)\}$
	$\mathcal{U}_{CC} = \{(R_{11}, k_3), (R_{12}, ac_1)\}$

Aspect comportemental

À présent, afin de modéliser le comportement intelligent du système pris en compte, nous définissons l'ensemble de traces modélisant son évolution physique ainsi que les projections de ces dernières sur les agents définis afin de modéliser son évolution globale. Nous définissons neuf traces comme le montre le Tableau 6.9. Ces traces représentent les différents chemins que peut exécuter le système LIS

— La trace t_1 représente l'envoi des données capturées par les capteurs vers les protocoles de communication internes qui se trouvent dans la couche de communication. La modélisation de cette trace implique deux états possible du système LIS, B_0 : les données sont dans les interfaces capteurs, B_1 : les données sont dans les protocoles de communication interne. Le passage de l'état B_0 à l'état B_1 est permis grâce à la règle de réaction R_1.

— La trace t_2 définit l'envoi des données capturées par les capteurs vers les protocoles de communication externes qui se trouvent dans la couche de communication. La modélisation de cette trace implique deux états possible du système LIS, B_0 : les données sont dans les interfaces capteurs, B_8 : les données sont dans les protocoles de communication externe. Le passage de l'état B_0 à l'état B_8 est permis grâce à la règle de réaction R_8.

— La trace t_3 explique que, lorsque les données capturées sont envoyées au protocole interne, le système formate ces données et les envoie à la couche Fog. Dans la définition de BiA-gents *, la modélisation des traces de ce comportement implique trois états LIS possibles, B_1 : les données sont dans le protocole interne, B_2 : les données sont formatées dans le

protocole interne et B_3 : les données sont dans le nœud Fog correspondant. Le passage d'un état à un autre est modélisé par deux règles de réaction R_2 et R_3.

— La trace t_4 représente le déclenchement d'une action dans le nœud Fog "Système de reconnaissance". Lorsque les données formatées sont dans ce dernier, le système crée une instruction en adéquation avec ces données. Ensuite, cette instruction est envoyée depuis le système de reconnaissance vers l'actuateur correspondant. La modélisation de cette trace implique trois états physiques B_3, B_4 et B_5. B_3 représente la présence des données formatées dans le nœud Fog "Système de reconnaissance". B_4 est l'état du système où une instruction est construite selon les données formatées et B_5 représente un des états finaux du système, une instruction est contenue dans un actuateur. Le passage d'un état à l'autre est permis à travers l'application des règles de réaction R_4 et R_5.

— La trace t_5 de la même manière que la trace t_4, la trace t_5 représente le déclenchement d'une action dans un nœud Fog. Sauf que ce dernier est le "Système d'alerte". Les différents états par lesquels passent cette trace sont B_3, B_6 et B_7. Chaque état représentant un état transitoire de l'actionnement d'une instruction selon les données formatées et le nœud Fog correspondant.

— La trace t_6 explique que, lorsque les données capturées sont envoyées au protocole externes, le système formate ces données et les envoie à la couche Cloud. Dans la définition des BiA-gents *, la modélisation des traces de ce comportement implique trois états LIS possibles, B_8 : les données sont dans les protocoles de communication externe, B_9 : les données sont formatées selon les protocole externes et B_{10} : les données sont dans les serveurs Cloud. Le passage d'un état à un autre est modélisé par deux règles de réaction R_9 et R_{10}.

— La trace t_7 représente le déclenchement d'une action dans les serveurs Cloud. Lorsque les données formatées sont dans ce dernier, le système crée une instruction en adéquation avec ces données. Ensuite, cette instruction est envoyée depuis le Cloud vers l'actuateur correspondant. La modélisation de cette trace implique trois états physiques B_{10}, B_{11} et B_{12}. B_{10} représente la présence des données formatées dans les serveurs Cloud. B_{11} est l'état du système où une instruction est construite selon les données formatées au niveau du Cloud et B_{12} représente un des états finaux du système, une instruction générée par les serveurs Cloud est contenue dans un actuateur. Le passage d'un état à l'autre est permis à travers l'application des règles de réaction R_{11} et R_{12}.

— La trace t_8 représente une détection de danger au niveau des protocoles de communication interne. Si jamais le système détecte une anomalie concernant les données, ce dernier supprime directement ces données pour éviter une quelconque transgression de la sécurité du système. La modélisation de cette trace se fait en deux états un état de détection B_1 et un état après suppression de données B_{13}. Le passage de B_1 vers B_{13} se fait à travers la règle de réaction R_{13}.

— La trace t_9 de la même manière que la trace t_8, représente une détection de danger. Cette dernière se fait du coté protocoles de communication externes en supprimer les données susceptible d'être dangereuses. La modélisation de cette trace se fait en deux états un état de détection B_8 et un état après suppression de données B_{14}. Le passage de B_8 vers B_{14} se fait à travers la règle de réaction R_{14}.

TABLE 6.9 – Traces modélisant le comportement physique du système LIS

Comportement possible de LIS	**Définitions des traces**
Envoyer les données aux protocoles de communication internes	$t_1 = B_0 \overset{R1}{\prec} B_1$
Envoyer les données aux protocoles de communication externes	$t_2 = B_0 \overset{R8}{\prec} B_8$
Traitement des données capturées destinées aux nœuds Fog	$t_3 = B_1 \overset{R2}{\prec} B_2 \overset{R3}{\prec} B_3$
Déclenchement d'action selon le système de reconnaissance	$t_4 = B_3 \overset{R4}{\prec} B_4 \overset{R5}{\prec} B_5$
Déclenchement d'action selon le système d'alerte	$t_5 = B_3 \overset{R6}{\prec} B_6 \overset{R7}{\prec} B_7$
Traitement des données saisies destinées aux systèmes Cloud	$t_6 = B_8 \overset{R9}{\prec} B_9 \overset{R10}{\prec} B_{10}$
Déclenchement d'action selon le système d'analyse et de stockage	$t_7 = B_{10} \overset{R11}{\prec} B_{11} \overset{R12}{\prec} B_{12}$
Détection du piratage des nœuds Fog	$t_8 = B_1 \overset{R13}{\prec} B_{13}$
Détection du piratage des systèmes Cloud	$t_9 = B_8 \overset{R14}{\prec} B_{13}$

Le comportement plus complexe d'un système Fog tel que LIS peut être obtenu en composant des traces (opérateur $\ll ; \gg$). Nous présentons ici les différentes compositions possibles des traces t_1 à t_9

- $t_1; t_3; t_4$. Pour détecter un sac dangereux dans la zone d'inspection des bagages, le système exécute la trace t_1 puis t_3 et enfin t_4. Cela signifie d'abord que lorsque les données sont saisies (B_0), le système les envoie au protocole de communication interne ($R1$). Ensuite, les données saisies sont formatées ($R2$) et envoyées à la couche Fog ($R3$). Enfin, le système de reconnaissance installé dans le nœud Fog correspondant déclenche une action ($R4$) qui affiche une alerte dans les écrans ($R5$).
- $t_1; t_3; t_5$. Afin de détecter un nombre anormal de valises/plateaux, le système exécute les traces t_1 puis t_3 et enfin t_5
- $t_1; t_8$. La détection de données malveillante se fait à travers l'exécution des traces t_1 puis t_8.
- $t_2; t_6; t_7$. L'analyse et le stockage de données dans le Cloud est modélisé par l'exécution des traces t_2, t_6 et t_7 de manière successive.
- $t_2; t_9$. La détection et suppression de formats malveillants de données est spécifiée par l'application des traces t_2 puis t_9.

Le Tableau 6.10 présente les différentes projections des traces $t_1 - t_9$ définies sur les différents agents constituant le système LIS. Nous précisons que les fonctions *host* et *int* dans notre exemple sont définies par :

$int(a, R1) =$ ("send to Fog") if $a = a_{S1}$ or a_{S2} or a_{S3} ;

$int(a, R8) =$ ("send to Cloud") if $a = a_{S1}$ or a_{S2} or a_{S3} ;

$int(a_{ICom}, R2) =$ ("sending to Fog") ; $int(a_{ECom}, R9) =$ ("sending to Cloud").

$h0(a) = region2$ if $a = a_{S1}$ or a_{S2} or a_{S3} or a_{ICom} or a_{ECom} or a_{FC} or a_{cc} ;

$h1(a_{S1}) = o_1$; $h2(a_{S2}) = o_2$; $h3(a_{S3}) = o_3$; $h4(a_{ICom}) = f_1$; $h5(a_{FC}) = f_1$ or $h5(a_{FC}) = f_2$, $h6(a_{ECom}) = f_2$; $h7(a_{CC}) = k_3$

Par exemple, la projection de t_3 est la suivante : L'agent de contrôle de la sécurité des communications internes a_{ICom} observe le bigraph B_1 depuis la région Fog et migre vers le format f_1. Cet agent analyse le format et prend la décision d'appliquer la règle de réaction R_2. Cet agent informe l'agent de contrôle Fog a_{FC} de sa décision. Après cette interaction, R_2 est appliqué suivi de R_3. Lorsque R_3 est appliqué, l'état du système devient B_3. a_{ICom} observe ce bigraph et retourne à son hôte initial, la région Fog.

TABLE 6.10 – Traces projections examples.

Projection des traces sur les agents définis
Projection de t_1 **sur** a_{s1}, a_{s2}, a_{s3}
$(B_0,h0) \overset{obs}{\prec} (B_0,h0) \overset{mgrt}{\prec} (B_0,h) \overset{an}{\prec} (B_0,h) \overset{int1}{\prec} (B_0,h) \overset{R1}{\prec} (B_1,h) \overset{obs}{\prec} (B_1,h) \overset{mgrt}{\prec} (B_1,h0)$ $*(h = h1 \text{ ou } h2 \text{ ou } h3)$
Projection de t_2 **sur** a_{s1}, a_{s2}, a_{s3}
$(B_0,h0) \overset{obs}{\prec} (B_0,h0) \overset{mgrt}{\prec} (B_0,h) \overset{an}{\prec} (B_0,h) \overset{int2}{\prec} (B_0,h) \overset{R8}{\prec} (B_8,h) \overset{obs}{\prec} (B_8,h) \overset{mgrt}{\prec} (B_8,h0)$
Projection de t_3 **sur** a_{ICom}
$(B_1,h0) \overset{obs}{\prec} (B_1,h0) \overset{mgrt}{\prec} (B_1,h4) \overset{an2}{\prec} (B_1,h4) \overset{int3}{\prec} (B_1,h4) \overset{R2}{\prec} (B_2,h4) \overset{R3}{\prec} (B_3,h4) \overset{obs}{\prec} (B_3,h4) \overset{mgrt}{\prec} (B_3,h0)$
Projection de t_8 **sur** a_{ICom}
$B_1,h0) \overset{obs}{\prec} (B_1,h0) \overset{mgrt}{\prec} (B_1,h4) \overset{an2}{\prec} (B_1,h4) \overset{R13}{\prec} (B_{13},h4) \overset{obs}{\prec} (B_{13},h4) \overset{mgrt}{\prec} (B_{13},h0)$
Projection de t_4 **sur** a_{FC}
$(B_3,h0) \overset{obs}{\prec} (B_3,h0) \overset{mgrt}{\prec} (B_3,h5) \overset{an3}{\prec} (B_3,h5) \overset{R4}{\prec} (B_4,h5) \overset{R5}{\prec} (B_5,h5) \overset{obs}{\prec} (B_5,h5) \overset{mgrt}{\prec} (B_5,h0)$
Projection de t_5 **sur** a_{FC}
$(B_3,h0) \overset{obs}{\prec} (B_3,h0) \overset{mgrt}{\prec} (B_3,h5) \overset{an3}{\prec})(B_3,h5) \overset{R6}{\prec} (B_6,h5) \overset{R7}{\prec} (B_7,h5) \overset{obs}{\prec} (B_7,h5) \overset{mgrt}{\prec} (B_7,h0)$
Projection de t_6 **sur** a_{ECom}
$(B_8,h0) \overset{obs}{\prec} (B_8,h0) \overset{mgrt}{\prec} (B_8,h6) \overset{an4}{\prec} (B_8,h6) \overset{int4}{\prec} (B_8,h6) \overset{R9}{\prec} (B_9,h6) \overset{R10}{\prec} (B_{10},h6) \overset{obs}{\prec} (B_{10},h6) \overset{mgrt}{\prec} (B_{10},h0)$
Projection de t_9 **sur** a_{ECom}
$(B_8,h0) \overset{obs}{\prec} (B_8,h0) \overset{mgrt}{\prec} (B_8,h6) \overset{an4}{\prec} (B_8,h6) \overset{R14}{\prec} (B_{13},h6) \overset{obs}{\prec} (B_{13},h6) \overset{mgrt}{\prec} (B_{13},h0)$
Projection de t_7 **sur** a_{CC}
$(B_{10},h0) \overset{obs}{\prec} (B_{10},h0) \overset{mgrt}{\prec} (B_{10},h7) \overset{an5}{\prec} (B_{10},h7) \overset{R11}{\prec} (B_{11},h7) \overset{R12}{\prec} (B_{12},h7) \overset{obs}{\prec} (B_{12},h7) \overset{mgrt}{\prec} (B_{12},h0)$

Nous avons ainsi terminé la modélisation physique, virtuelle et comportementale du cas d'étude défini. Nous avons détaillé les différentes définitions permettant d'illustrer l'apport du modèle proposé afin de spécifier un système Fog concret. Dans ce qui suit, nous procédons à l'exécution de ce système et à la vérification de propriétés inhérentes aux systèmes Fog.

6.5 Analyse formelle de LIS

Dans cette Section, nous proposons un moyen de transformer, dans la langage Maude stratégie, les spécifications BiAgents* tel qu'illustré par la Figure 6.3.

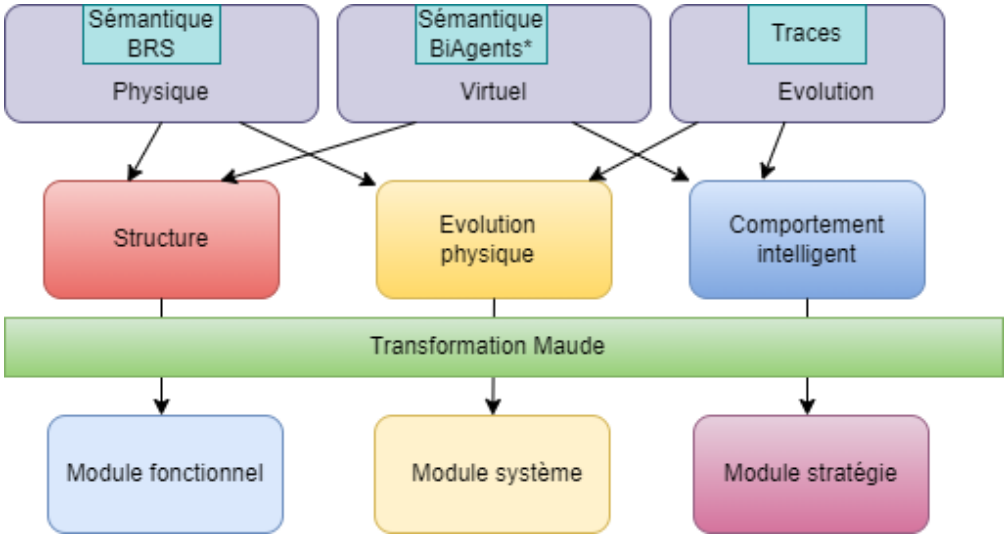


FIGURE 6.3 – Principe de transformation des spécification BiAgents* dans Maude strategy

Ainsi que présenté dans le Chapitre 3, Maude[Clavel et al., 2005] est un langage formel de spécification de haut niveau basé sur des logiques équationnelle et de réécriture. Un programme Maude décrit une théorie logique, son calcul met en œuvre une inférence logique basée sur les axiomes du programme. Maude stratégie est composé de trois modules :

1. **Le module fonctionnel** implémente une théorie équationnelle mathématiquement décrite par une paire $(\Sigma, E \cup A)$, où Σ est la signature du module qui spécifie les sortes, sous-sortes, types et opérateurs. E est la collection d'équations (possiblement conditionnelle) et A est l'ensemble des attributs équationnels déclarés pour certains opérateurs tels que $assoc$ pour associatif, $comm$ pour commutatif, etc...
2. **Le module stratégie** SM est utilisé pour guider le processus de réécriture et l'utilisation de ces règles de réécriture qui influence le comportement du système. Ce module contient les sortes des stratégies ainsi que leurs définitions sous forme de suites ordonnées de règles de réécriture.
3. **Le module système** qui implémente une théorie de réécriture en tant que 3-uple $(\Sigma, E \cup A, S(R, SM))$, où $(\Sigma, E \cup A)$ représente la partie module fonctionnel et S défini la sémantique décrivant le comportement du système. S est définie par : R (la collecte de règles de réécriture éventuellement conditionnelles) et SM, le module de stratégie.

Ainsi, les aspects physiques des spécifications BiAgents* (signature et logique de typage), sont implémentés dans les module fonctionnels DevicesSpec, CommunicationSpec, FogSpec, CloudSpec et AgentSpec selon les couches du système Fog spécifiés et les agents orchestrateurs tel que représenté dans la Figure 6.4. Les opérations et équations déclarées définissent des constructeurs et des axiomes qui construisent les éléments du système et capturent leurs états. De même, la dynamique introduite par la structure physique des BiAgents* ainsi que les fonctions de la structure virtuelle est traduite dans un module système LuggageInspectionBehaviour définissant un ensemble de règles de réécriture R conditionnelles (où non conditionnelles). De plus, le comportement intelligent d'un système Fog permettant de définir plusieurs scénarios alternatifs à travers les traces Bigraphiques et leurs projections sur les agents est encodé dans un module stratégie LuggageInspectionStrategies définissant un ensemble de stratégies et de sous stratégies composées de règles de réécritures.

Concrètement, la spécification dans le langage Maude stratégie permet d'encoder les spécifications BiAgents* de manière à unifier leurs différents concepts selon un seul langage en préservant leur sémantique. Cela permet d'implémenter les différents aspects d'un système Fog et donc de contrôler son comportement complexe en raisonnant sur son état global. La Figure 6.4 donne une vision d'ensemble du principe d'encodage dans Maude stratégie des spécifications BiAgents*.

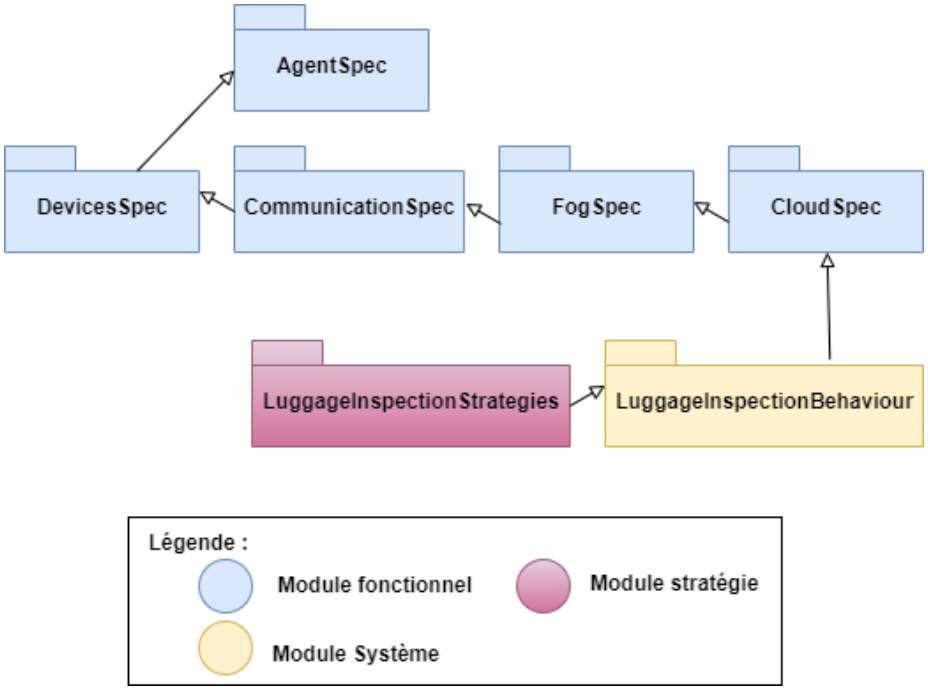


FIGURE 6.4 – Modules du système LIS dans Maude strategy

6.5.1 Exécution des scénarios

Dans cette section, nous procédons à l'exécution du système LIS à l'aide de Maude stratégie en séparant ce système en trois parties : Structure, Evolution dynamique et comportement intelligent.

La sémantique du langage Fog-DSML pour les systèmes Fog est traduite dans les modules fonctionnels `AgentSpec`, `DevicesSpec`, `CommunicationSpec`, `FogSpec` et `CloudSpec`. Ces modules intègrent la sémantique de typage (sortes), les relations de sous-sortes et les opérations algébriques utilisée pour définir la structure des différents éléments d'un système Fog. En vue de la complexité des comportements définis, nous présentons ici une transformation de la sémantique vers le langage Maude stratégie considérant le système LIS. Le module fonctionnel `AgentSpec` comprend trois parties. La première définit structurellement les différents agents du système, la deuxième donne les différentes opérations que peuvent effectuer ces agents représentant aussi l'état dans lequel se trouvent ces derniers et la troisième comprend les différentes équations permettant aux agents d'orchestrer le comportement du système selon les données traitées.

Le Tableau 6.11 regroupe les déclarations les plus pertinentes définies dans ce module.

Les Tableaux 6.12 6.13 6.14 et 6.15 contiennent le code le plus important des différents modules fonctionnels DevicesSpec, ComunicationSpec, FogSpec et CloudSpec avec les définition structurelles des éléments du système ainsi que les opérations servant à faire évoluer l'état du système.

Les sortes précédemment définies (i : capteur, k : actuateur, f : nœud Fog, c : protocole de communication, a : service Cloud) sont préservées et exprimées par les sortes Sensor, Ac-

tuator, FogNode et CloudService respectivement. Les types des agents (capteur, contrôle Fog , contrôle de communication , contrôle du Cloud) sont aussi préservés et exprimés par les sortes SensorAgent, FogControl, CommunicationSecurityControl et CloudControl respectivement.

La transformation dans Maude stratégie permet également d'enrichir la spécification des agents du systèmes Fog en définissant explicitement leurs fonctions d'analyse. Syntaxiquement, une sorte est déclarée *via* le mot-clé `sort` et est définie par un constructeur (`ctor`) associé. Afin de capturer les différents états du système, nous définissons un ensemble d'opérations et prédicats (équations). Ce sont des opérations déclarées par le mot-clé `op` qui prennent en paramètres différentes structures et qui renvoient des informations les concernant comme le format de données communiquées ou leur valeur ou une valeur de vérité concernant la nature du prédicat.

encoder les différentes prises de décision du système.

TABLE 6.11 – Principales déclarations du module fonctionnel AgentSpec

Module Fonctionnel AgentSpec
Sortes et structures

`sorts Agent System Host Data Format Layer Action .`
`SensorAgent FogControl CommunicationSecurityControl CloudControl .`
`subsorts SensorAgent < Agent .`
`subsorts FogControl < Agent .`
`subsorts CommunicationSecurityControl < Agent .`
`subsorts CloudControl < Agent .`
`op sensor [_,_,_] : Nat Nat Host -> SensorAgent [ctor] .`
`op fc [_,_,_] : Nat Nat Host -> FogControl [ctor] .`
`op icom [_,_,_] : Nat Nat Host -> CommunicationSecurityControl [ctor] .`
`op ecom [_,_,_] : Nat Nat Host -> CommunicationSecurityControl [ctor] .`
`op cc [_,_,_] : Nat Nat Host -> CloudControl [ctor] .`

Opérations

`op observe(_,_) : Nat System -> System .`
`op analyse(_,_) : Nat Data -> Bool .`
`op interaction (_,_,_) : Nat Nat String -> System .`
`op mgrt(_,_,_) : Nat Host Host -> System .`

Équations

`ceq analyse (x,l) = true if l == true /\ x == 1 .`
`ceq analyse (x,l) = false if l == false /\ x == 1 .`
`ceq analyse(x , k ) = true if k > 10000 /\ x == 3 .`
`ceq analyse (x , k) = false if k < 10000 /\ x == 3 .`
`ceq analyse(x , d ) = true if ( x == 2) /\ ( d > 8000 ) .`
`ceq analyse(x , d ) = false if ( x == 2) /\ ( d < 8000 ) .`
`ceq analyse( x, act ) = true if x == 7 /\ act == "store" .`
`ceq analyse( x , act ) = true if x == 4 /\ act == "display" .`
`ceq analyse( x , f ) = true if x == 5 /\ f == "JSON" .`
`ceq analyse ( x , f ) = false if x == 5 /\ f =/= "JSON" .`
`ceq analyse ( x , f ) = true if ( x == 6 ) /\ f == "XML" .`
`ceq analyse ( x , f ) = false if ( x == 6 ) /\ f =/= "XML" .`

TABLE 6.12 – Principales déclarations du module fonctionnel DevicesSpec

Module Fonctionnel DevicesSpec
Sortes et structures
sort Sensor . subsorts Sensor < Host . sort Actuator . subsorts Actuator < Host . sort DLayer . op Dlayer[_,_,_,_] : Sensor Sensor Sensor Actuator -> DLayer [ctor] . op bagsScanner : -> Sensor [ctor] . op barCodeScanner : -> Sensor [ctor] . op camera : -> Sensor [ctor] . op screen : -> Actuator [ctor] .
Opérations
op hostData [_,_] : Data Agent -> Data . op capData [_,_] : Sensor Data -> Sensor . op actionate[_,_] : Actuator Action -> Actuator . op buildData (_) : Nat -> Data . op buildData (_) : Rat -> Data . op buildData (_) : Bool -> Data .

TABLE 6.13 – Principales déclarations du module fonctionnel CommunicationSpec

Module Fonctionnel CommunicationSpec
Sortes et structures
sort ComLayer . sort Protocol . subsorts Protocol < Host . sort CommLayer . subsorts CommLayer < Host . op Comlayer[_,_] : CommLayer CommLayer -> ComLayer [ctor] . op buildComLayer [_,_] : Protocol Format -> CommLayer [ctor] . op buildComLayer [_,_,_] : Protocol Data Format -> CommLayer [ctor] . op buildComLayer [_] : Protocol -> CommLayer [ctor] . op _|_ : Protocol Protocol -> Protocol [ctor] .
Opérations
op internP : -> Protocol . op externP : -> Protocol . op internF : -> Format . op externF : -> Format . op formatData (_,_) : Format Data -> Format . op hostFormat [_,_] : Format Agent -> Format .

TABLE 6.14 – Principales déclarations du module fonctionnel FogSpec

Module Fonctionnel FogSpec
Sortes et structures
sort FogNode . subsorts FogNode < Host . sort FLayer . subsorts FLayer < Host . op Flayer[_,_,_,_,_,_,_,_,_] : FogNode FogNode SensorAgent SensorAgent SensorAgent FogControl CommunicationSecurityControl CommunicationSecurityControl CloudControl -> FLayer [ctor] .
Opérations
op fogAction [_,_] : FogNode Action -> FogNode . op buildAction[_,_] : String Format -> Action . op AlertSyst : -> FogNode . op RecogSyst : -> FogNode . op formatAction [_,_] : Action Format -> Action . op hostData [_,_] : Data Agent -> Data . op hostFormat [_,_] : Format Agent -> Format . op hostAction [_,_] : Action Agent -> Action .

TABLE 6.15 – Principales déclarations du module fonctionnel CloudSpec

Module Fonctionnel CloudSpec
Sortes et structures
sort CloudSystem . sort ClLayer . subsorts ClLayer < Host . op CloudLayer [_] : CloudSystem -> ClLayer [ctor] . op CloudLayer [_,_] : CloudSystem Format -> ClLayer [ctor] . op buildCloudSystem [_,_] : CloudSystem Action -> CloudSystem [ctor] .
Opérations
op analysisSyst : -> CloudSystem . op analysisStorageSyste [_] : Action -> CloudSystem . op null : -> Action [ctor] .

Les différentes règles de réaction exprimant l'évolution de la structure physique d'un système Fog ainsi que les changements d'états des différents agents qui gèrent le système sont encodées dans un module système Maude stratégie nommé LuggageInspectionBehaviour. Les règles de réécriture encodant les actions de changement d'état physique (Les différentes règles de réaction bigraphiques) sont exprimées par des fonctions de réécriture Maude. Les fonctions d'interaction et d'analyse des différents agents sont sont exprimées par des règles de réécriture conditionnelles. Le Tableau 6.16 regroupe les définitions les plus pertinentes de ce module. Les règles de réécriture conditionnelles de Maude consistent à réécrire la partie gauche de la règle en sa partie droite si les conditions de déclenchement spécifiées sont vérifiées. La partie gauche de la règle décrit un état du système et sa partie droite exprime une reconfiguration désirée. Les conditions de déclenchement sont exprimées *via* les différentes décisions des agents définis dans le module fonctionnel `AgentSpec`.

Le mot-clé crl pour "conditional rewrite rule" déclare une règle de réécriture conditionnelle. La partie gauche est réécrite en la partie à droite du symbole (=>) si la condition spécifiée est vérifiée.

Le mot-clé rl pour "rewrite rule" déclare une règle de réécriture sans condition. La partie gauche est réécrite en la partie à droite du symbole (=>) si jamais le système atteint la

configuration encodée par la partie gauche.

TABLE 6.16 – Principales déclarations du module système LuggageInspectionBehaviour

Module Système LuggageInspectionBehaviour
Fonctions
op buildSystem <_,_,_,_> : DLayer ComLayer FLayer ClLayer -> System [ctor] . var i : Bool . var number : Nat . var number2 : Rat . var intF : String . var extF : String . var Fogact : String . var Cloudact : String .
Règles de réécriture
rl [nom-règle] : terme => terme' if condition(s)
États des agents
observe (1 , Système) observe (2 , Système) observe (3 , Système) mgrt(1 , FLayer , bagsScanner) . mgrt (2 , FLayer , barCodeScanner) . mgrt (3 , FLayer , camera) . crl [analyse] : term => interaction (agentID,agentID, "message") if analyse(agentID , Data) . crl [analyse] : term => term' if analyse(agentID , Data) .

Chaque trace définie dans la sémantique de Fog-DSML peut être définie par une stratégie, Ainsi, les composition de traces sont aussi définies par une composition de stratégies, telle que définie dans un module de stratégie Maude stratégie. Chaque règle de réaction ou série de règles de réaction est déclenchée par le résultat de l'analyse de l'agent approprié. Chaque boucle d'évolution de l'état des agents est représentée par une stratégie simple comme l'indique la stratégie S1 dans le Tableau 6.17. Ces stratégies sont une combinaison de règles de réécriture séquentielle (observation, migration, etc.).

TABLE 6.17 – Principales déclarations du module de stratégies LuggageInspectionStrategies

Module Stratégie LuggageInspectionStrategies
Sortes de stratégies
strat S @ System . strat ST1 @ System . strat ST @ System . S1 := (observation1 ; migration1) . ...
Composition de stratégies
Stratégies composées à l'aide de l'opérateur (;;...) sd ST1 := (S1 ; S11 ; S41 ; S411) .
Stratégie Globale
sd ST := (ST1 | ST2 | ... | ST9) * .

6.5.2 Vérification formelle des propriétés

La vérification formelle incarne une méthode très efficace pour assurer l'absence d'erreurs dans un système donné. La vérification formelle de modèles ou le "model checking" [Chechik

and Gannon, 2001] [Baier and Katoen, 2008] 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 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.

Dans cette Section, nous proposons une approche de vérification formelle des propriétés d'interopérabilité et de portabilité des données dans les systèmes Fog selon les stratégies d'orchestration définies. Nous nous basons sur le modelchecker de Maude stratégie qui repose sur la logique temporelle linéaire LTL.

Encodage des propriétés dans Maude

Le Fog computing fait plusieurs promesses en terme de garantie de propriétés inhérentes aux systèmes distribués. Dans notre travail de thèse, nous nous sommes concentré sur les aspects les plus importants, selon notre point de vue, que l'on retrouve dans la majorité des systèmes distribués.
Afin de permettre au model-checker de Maude stratégie de raisonner sur l'interopérabilité et la portabilité de données de ces systèmes Fog, il est primordial d'exprimer dans le langage Maude stratégie la sémantique du langage Fog-DSML ainsi que les formules propositionnelles de la logique LTL (comportements désirés). Pour accomplir cette tâche, Maude stratégie permet de définir ces spécifications dans un module système, PropertiesChecking, dédié à la spécification des propriétés du système. Ce module permet de définir les différentes formules LTL et fournit une mécanique pour la vérification de leur satisfaction. Ensuite, ce nouveau module sera associé la théorie de réécriture, définie dans le module système LuggageInspectionBehaviour, ainsi que le module stratégie LuggageInspectionStrategies décrivant les comportements du modèle spécifié. La Figure 6.5 donne une vue d'ensemble de modules composant notre solution complète pour la spécification, l'exécution et la vérification formelle de l'interopérabilité et de la portabilité des données des systèmes Fog dans Maude stratégie. Nous précisons la sorte (parmi celles déclarées précédemment) qui représentera les états considérés par le model-checker. Ensuite, nous définissons dans Maude stratégie les prédicats pertinents représentant les différents états *via* la relation de satisfaction $\mid =$. Quant aux formules propositionnelles LTL, elles sont directement encodées en tant que propriétés dans le langage Maude stratégie.

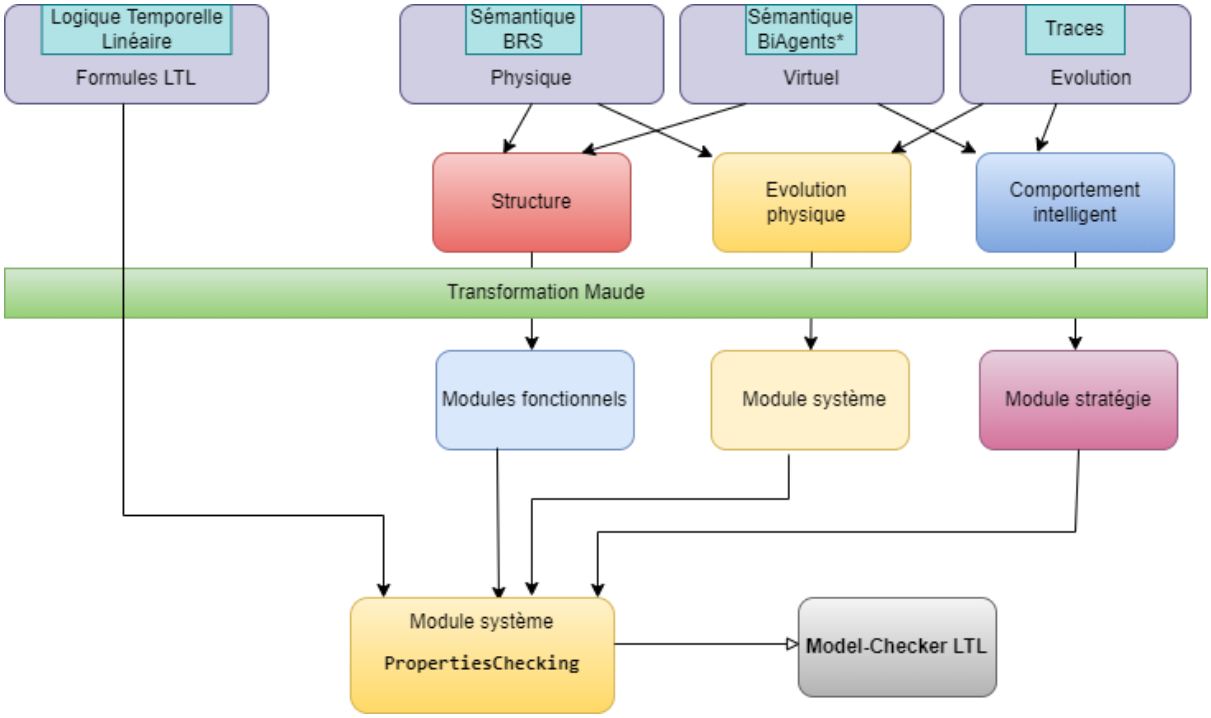


FIGURE 6.5 – Vue d'ensemble de la solution de spécification, d'exécution et de vérification dans Maude stratégie

Une fois le module PropertiesChecking défini, le model-checker LTL de Maude stratégie peut enfin être exécuté afin de vérifier les propriétés spécifiées. Le model-checker est lancé *via* la commande modelCheck et prend en entrée (1) une configuration initiale du système (2) une propriété sous forme d'une formule LTL à vérifier et la stratégie à adopter. En sortie, il retourne la valeur booléenne True (vrai) si la propriété est satisfaite, ou un contre-exemple quand elle est violée au cours de l'exécution du système [Clavel et al., 2016].

Afin d'illustrer le fonctionnement du model-checker, nous considérons le cas d'étude LIS dont nous voudrions vérifier certaines propriétés. Nous explicitons la configuration initiale de ce systèmes syntaxiquement selon notre définition des systèmes Fog de la manière suivante :

```
   Dlayer[  bagsScanner , barCodeScanner, camera, screen ] ,
Comlayer [ buildComLayer[ internP , FormatType( "JSON" ) ] ,
buildComLayer[ externP , FormatType( "XML" ) ] ] ,
Flayer[  fogAction[ AlertSyst , ActionInstruction( "display" ) ] ,
fogAction[ RecogSyst , ActionInstruction( "display" ) ] ,
sensor[ 1 , 1 ; 2 , FLayer] , sensor[ 2 , 3 ; 4 , FLayer ] ,
sensor [ 3 , 5 ; 6 , FLayer ] , fc[ 4 , 7 ; 8 ,  FLayer ] ,
csc1[ 5 , 9 ; 10 , FLayer ] , csc2[ 6 , 11 ; 12 , FLayer] ,
cc[ 7 , 13 ; 14  , FLayer ]  ] ,
 CloudLayer[  analysisStorageSyste[ ActionInstruction( "store" ) ] ]  .
```

Le model-checker est exécuté avec la configuration initiale et la propriété `semanticInteroperability` à vérifier. Nous rappelons que cette propriété tente de vérifier que en permanence, les données, les formats et les actions qui y sont liés peuvent être analysés par tous les agents du système. La Figure 6.6 montre le résultat du model-checker avec les

entrées mentionnées où le résultat True (vrai) est retourné, indiquant que la propriété a bien été assurée.

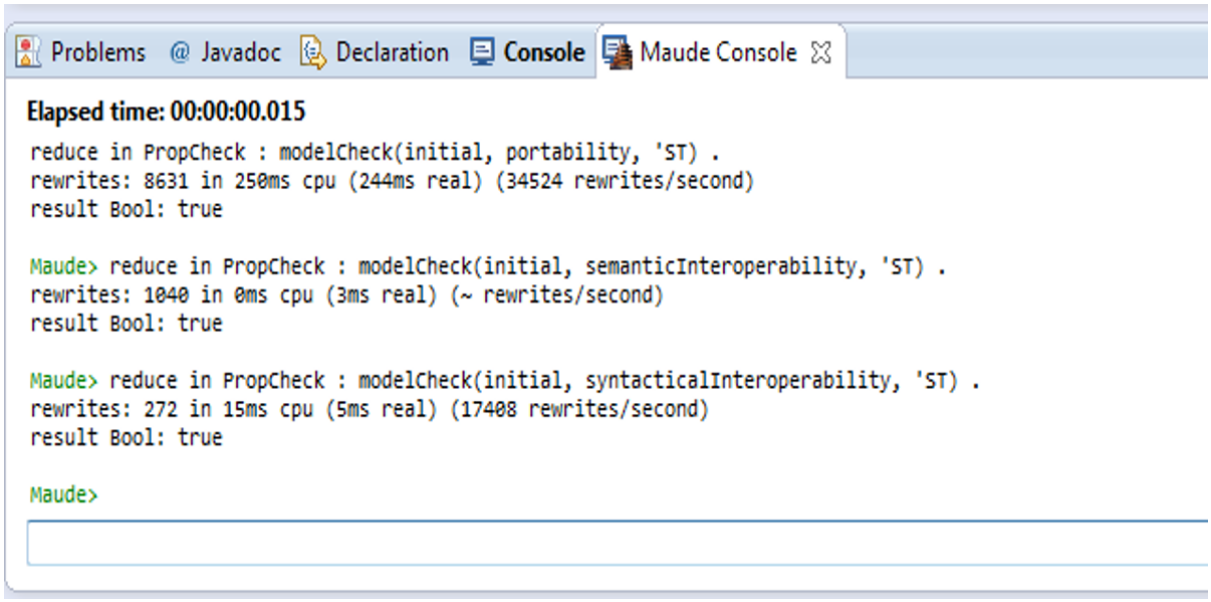


FIGURE 6.6 – Résultat de la vérification de l'interopérabilité sémantique sous LTL Maude stratégie

De la même manière, nous pouvons soumettre le système LIS à la vérification formelle de son comportement interopérable selon l'interopérabilité syntaxique et la portabilité des données. Le résultat true est renvoyé pour la vérification des propriétés "`syntacticalInteroperability`" et "`portability`" (voir Figures 6.6).

6.6 Conclusion

Dans ce chapitre, nous avons présenté l'évaluation du langage Fog-DSML par un cas d'étude LIS. Nous avons modélisé les différents scénarios de ce cas d'étude à l'aide du langage Fog-DSML. Nous avons aussi évalué ce langage par une transformation de sa sémantique vers le langage de spécification formelle Maude stratégie. Cette transformation permet de préserver la sémantique structurelle (physique et virtuelle) des systèmes Fog tout en l'enrichissant d'aspects qualitatifs. Maude stratégie permet également d'implémenter les stratégies de gestion des services afin de permettre leur exécution de manière générique et autonomique. Nous avons procédé à la vérification formelle de l'interopérabilité syntaxique et sémantique en plus de la portabilité des données du système LIS modélisé, grâce aux outils fournis par le système Maude stratégie. Cette vérification se base sur une technique de model-checking supportée par la logique temporelle linéaire LTL.

Chapitre 7

Conclusion générale

Sommaire

Le Fog computing est un nouveau paradigme qui promet une valeur économique et scientifique substantielle pour l'industrie et le monde universitaire dans les années à venir. Cependant, la concrétisation des systèmes Fog et la nature de leurs technologies ont entraîné des défis en termes de traitement des données et de gestion des services par rapport aux paradigmes de mise en réseau actuels (par exemple, le Cloud computing) en raison de l'ajout d'une couche intermédiaire entre les objets et les serveurs distants. Les recherches scientifiques de la littérature sont encore à leurs débuts. Bien que les aspects technologiques du Fog computing sont très avancés, il y a un manque de solutions concrètes basées sur le Fog qui répondent à l'ensemble des exigences de l'IoT étendu par le Fog computing pour la conception et la modélisation de ces systèmes. Par conséquent, cette thèse entre en action pour répondre à l'exigence de développer une solution complète pour faire face aux limitations/défis de l'abstraction et le développement des systèmes Fog.

Certaines solutions académiques et commerciales ont été proposées afin de diminuer la complexité de tels systèmes. Entre autres, les approches semi-formelles ou formelles existantes et qui contribuent à la conception des systèmes Fog se concentrent essentiellement sur l'aspect architectural, structurel ou bien comportemental de ces systèmes. Ces approches proposent des solutions simples et limitées puisqu'elles prennent en compte certains aspects au détriment d'autres et ne prennent pas en compte les dépendances entre les architectures, les différentes structures et le comportement intelligent des systèmes Fog.

7.1 Contributions

Dans cette thèse, nous proposons une solution à fondements formels pour la spécification structurelle et comportementale des systèmes Fog et l'analyse de ces systèmes à travers une vérification qualitative des propriétés inhérentes aux systèmes Fog. Notre approche repose sur la proposition d'un DSML dédié au Fog computing. La définition d'un langage de modélisation spécifique au domaine à travers la séparation des syntaxes et sémantique d'un système Fog permet de diminuer la complexité de conception et d'analyse de ce type de systèmes.

Pour accomplir de manière incrémentale cet objectif et s'assurer que chaque contribution apporte une amélioration dans le contexte de cette recherche, nous avons suivi les étapes suivantes :

1. **Architecture multi-couches pour le Fog computing.** Nous avons proposé une architecture dédiée à l'étude des systèmes Fog. Cette architecture multi-couches catégorise les différents éléments d'un système Fog selon leurs rôles et les répartie à travers quatre couches. Basée sur la norme ISO/IEC/ IEEE 42010 :2011, cette architecture possède différentes vues physique, virtuelle et interactionnelle faisant abstraction de la complexité de ces systèmes.

2. **Extension du formalisme BiAgents.** Nous avons enrichi l'extension des BRS par des agents notée BiAgents en redéfinissant les agents d'un point de vue fonctionnel, nous avons ajouté la notion d'interaction entre eux afin de leur permettre d'orchestrer leurs décisions. Á présent, les BiAgents* reflètent mieux les différents aspects des systèmes Fog.

3. **Sémantique structurelle et comportementale des systèmes Fog.** Nous avons utilisé le formalisme BiAgents* afin de modéliser les aspects physique, virtuels et comportementaux des systèmes Fog en général. Nous avons modélisé la structure physique de ce type de systèmes à l'aide des BRS. De ce fait, la localité et la connectivité des entités impliqués sont naturellement spécifiés. Nous avons défini la structure virtuelle d'un système Fog par un ensemble d'agents responsables chacun d'un type d'entités à gérer. Enfin, nous avons déterminé les différentes stratégies d'exécution utilisées pour représenter le comportement intelligent d'un système Fog d'orchestration des services à travers des traces bigraphiques et leurs projections sur les différents agents définis. Le langage Fog-DSML a hérité des résultats de recherche précédents pour lui fournir une syntaxe abstraite, une syntaxe concrète et une sémantique opérationnelle.

4. **Spécification des stratégies d'exécution.** Nous avons utilisé le langage Maude stratégie afin de spécifier la structure et le comportement des systèmes Fog à travers l'encodage des différents concepts et dimensions des BiAgents* en Maude stratégie.

5. **Vérification formelle des propriétés d'interopérabilité et de portabilité des données.** Nous avons procédé à la vérification formelle des propriétés d'interopérabilité syntaxique et sémantique et de portabilité des données du modèle à travers l'élaboration des propriétés LTL correspondantes. Nous avons exprimer les formules LTL à l'aide de prédicats sous forme de formules TQL afin d'exprimer les propriétés spatiales de ces derniers. Nous avons procédé à la vérification des propriétés LTL à travers le model-checker intégré dans l'outil Maude stratégie permettant de prendre en compte les stratégies définies dans la vérification des propriétés.

6. **Évaluation par cas d'étude LIS.** Nous avons évalué notre approche à travers le cas d'étude LIS (système de surveillance des bagages dans un aéroport). Nous avons représenté la structure et le comportement de ce système à l'aide du langage Fog-DSML. Nous avons implémenté les différents scénarios d'exécution à l'aide de l'outil Maude stratégie.

7.2 Perspectives

Plusieurs pistes de recherche peuvent être entreprises pour donner une suite aux travaux scientifiques décrits dans cette thèse. Nous identifions dans la suite quelques travaux que nous

planifions de réaliser.

Validation quantitative des propriétés des systèmes Fog. Cette vérification concerne les messages échangés entre les différents agents du système et entre les différents systèmes. Ainsi, nous pourrions respecter certaines contraintes de communication imposées par les utilisateurs. Le caractère probabiliste pourrait être ajouté au comportement des agents (états et trace)

Création d'un outil entièrement automatisé pour Fog-DSML. Depuis la phase de modélisation jusqu'à l'analyse des résultats obtenus lors de la vérification des propriétés, il serait particulièrement intéressant de concevoir un outil entièrement automatisé autour du DSML proposé. L'outil devrait idéalement comprendre une interface graphique simple permettant à l'utilisateur de modéliser facilement des systèmes Fog à l'aide du formalisme BiAgents* suggéré. Le but de cette modélisation est de créer un traducteur qui permettra aux modèles BiAgents* d'être convertis en spécifications Maude stratégie. Un dernier composant permettrait l'exécution et l'analyse des comportements intelligents indiqués afin de poursuivre la vérification qualitative en fonction des exigences de l'utilisateur. Enfin, cet outil permettrait d'intégrer les capacités de simulation des systèmes Fog modélisés, ce qui faciliterait l'administration.

Transformation au niveau métaèmodèles des BiAgents* et des éléments de l'architecture CLA*4*Fog pourrait apporter une solution idéale pour l'implémentation et le déploiement des systèmes Fog. Beaucoup de détails pourront être ajoutés quant à la définition des BiAgents* pour prendre en considération d'autres aspects liés aux systèmes Fog en général et les systèmes cyber physiques en particuliers.

Bibliographie

[rub, 2019] (2019). Model checking strategy-controlled rewriting systems (extended version). Technical Report 2/19. Extended version of a paper presented at FSCD 2019 (DOI : 10.4230/LIPIcs.FSCD.2019.34).

[Abbas et al., 2018] Abbas, H., Shaheen, S., Elhoseny, M., Singh, A. K., and Alkhambashi, M. (2018). Systems thinking for developing sustainable complex smart cities based on self-regulated agent systems and fog computing. *Sustainable Computing : Informatics and Systems*, 19 :204–213. `https://www.sciencedirect.com/science/article/pii/S2210537918300295`.

[Abd Elaziz et al., 2021] Abd Elaziz, M., Abualigah, L., Ibrahim, R. A., and Attiya, I. (2021). Iot workflow scheduling using intelligent arithmetic optimization algorithm in fog computing. *Computational intelligence and neuroscience*, 2021.

[Ai et al., 2018] Ai, Y., Peng, M., and Zhang, K. (2018). Edge computing technologies for internet of things : a primer. *Digital Communications and Networks*, 4(2) :77–86.

[Asadi et al., 2021] Asadi, M., Fathy, M., Mahini, H., and Rahmani, A. M. (2021). An evolutionary game approach to safety-aware speed recommendation in fog/cloud-based intelligent transportation systems. *IEEE Transactions on Intelligent Transportation Systems*.

[Association et al., 2018] Association, I. S. et al. (2018). Ieee standard for adoption of openfog reference architecture for fog computing. *IEEE Std 1934-2018*, pages 1–176.

[Baier and Katoen, 2008] Baier, C. and Katoen, J.-P. (2008). *Principles of model checking*. MIT press.

[Barriga et al., 2021] Barriga, J. A., Clemente, P. J., Sosa-Sánchez, E., and Prieto, Á. E. (2021). Simulateiot : Domain specific language to design, code generation and execute iot simulation environments. *IEEE Access*, 9 :92531–92552.

[Benzadri, 2016] Benzadri, Z. (2016). *Spécification et Vérification Formelle des Systèmes Cloud*. Thesis, Université Constantine 2 - Abdelhamid Mehri.

[Benzadri et al., 2021] Benzadri, Z., Bouheroum, A., and Belala, F. (2021). A formal framework for secure fog architectures : Application to guarantee reliability and availability. *International Journal of Organizational and Collective Intelligence (IJOCI)*, 11(2) :51–74.

[Bonomi et al., 2012] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16.

[Borque and Fairley, 2014] Borque, P. and Fairley, R. (2014). Guide to the software engineering body of knowledge version 3.0. *IEEE Computer Society Staff*.

[Chechik and Gannon, 2001] Chechik, M. and Gannon, J. (2001). Automatic analysis of consistency between requirements and designs. *IEEE transactions on Software Engineering*, 27(7) :651–672.

[Chen et al., 2020] Chen, X., Ding, J., Lu, Z., and Zhan, T. (2020). An efficient formal modeling framework for hybrid cloud-fog systems. *IEEE Transactions on Network Science and Engineering*, 8(1) :447–462.

[Cherfia, 2016] Cherfia, T. A. (2016). *Bigraphical Reactive Systems Based Framework for Design and Analysis of Context-Aware Systems.* Theses, Université Constantine 2 - Abdelhamid Mehri. `https://tel.archives-ouvertes.fr/tel-01258398`.

[Clark et al., 2004] Clark, T., Evans, A., Sammut, P., and Willans, J. (2004). An executable metamodelling facility for domain specific language design.

[Clavel et al., 2005] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martı-Oliet, N., Meseguer, J., and Talcott, C. (2005). Maude manual (version 2.1). *SRI International, Menlo Park*.

[Clavel et al., 2016] Clavel, M., Durán, F., Eker, S., Escobar, S., Lincoln, P., Martı-Oliet, N., Meseguer, J., and Talcott, C. (2016). Maude Manual (Version 2.7. 1).

[Clavel et al., 2007] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., and Talcott, C. (2007). *All about maude-a high-performance logical framework : how to specify, program and verify systems in rewriting logic.* Springer-Verlag.

[Clavel et al., 2002] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martı-Oliet, N., Meseguer, J., and Quesada, J. F. (2002). Maude : Specification and programming in rewriting logic. *Theoretical Computer Science*, 285(2) :187–243.

[Clavel et al., 1996] Clavel, M., Eker, S., Lincoln, P., and Meseguer, J. (1996). Principles of maude. *Electronic Notes in Theoretical Computer Science*, 4 :65–89.

[Combemale, 2008] Combemale, B. (2008). *Approche de métamodélisation pour la simulation et la vérification de modèle–Application à l'ingénierie des procédés.* PhD thesis, Institut National Polytechnique de Toulouse-INPT.

[Conforti et al., 2003] Conforti, G., Ghelli, G., Flesca, S., Greco, S., Saccà, D., and Zumpano, E. (2003). Spatial tree logics to reason about semistructured data. *language*, 17 :16.

[Da Silva, 2015] Da Silva, A. R. (2015). Model-driven engineering : A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43 :139–155.

[Damgaard and Birkedal, 2006] Damgaard, T. C. and Birkedal, L. (2006). Axiomatizing binding bigraphs. *Nordic Journal of Computing*, 13(1/2) :58.

[Debois and Damgaard, 2005] Debois, S. and Damgaard, T. C. (2005). Bigraphs by example. Technical report, Citeseer.

[Díaz-de Arcaya et al., 2020] Díaz-de Arcaya, J., Miñón, R., Torre-Bastida, A. I., Del Ser, J., and Almeida, A. (2020). Padl : a language for the operationalization of distributed analytical pipelines over edge/fog computing environments. In *2020 5th International Conference on Smart and Sustainable Technologies (SpliTech)*, pages 1–6. IEEE.

[Eker et al., 2007] Eker, S., Martí-Oliet, N., Meseguer, J., and Verdejo, A. (2007). Deduction, strategies, and rewriting. *Electronic Notes in Theoretical Computer Science*, 174(11) :3–25.

[Faisandier, 2011] Faisandier, A. (2011). Ingenierie des systemes complexes : Complements et mise en oeuvre. *MAP systeme*.

[Faithfull et al., 2013] Faithfull, A. J., Perrone, G., and Hildebrandt, T. T. (2013). Big red : A development environment for bigraphs. *Electronic Communications of the EASST*, 61.

[Gantz and Reinsel, 2011] Gantz, J. and Reinsel, D. (2011). Extracting value from chaos. *IDC iview*, 1142(2011) :1–12.

[Gharbi et al., 2021] Gharbi, C., Hsairi, L., and Zagrouba, E. (2021). A secure integrated fog cloud-iot architecture based on multi-agents system and blockchain. In *ICAART (2)*, pages 1184–1191.

[Greenfield and Short, 2003] Greenfield, J. and Short, K. (2003). Software factories : assembling applications with patterns, models, frameworks and tools. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 16–27.

[Grohmann and Miculan, 2007] Grohmann, D. and Miculan, M. (2007). Directed bigraphs. *Electronic Notes in Theoretical Computer Science*, 173 :121–137.

[Guo et al., 2020] Guo, Y., Zhang, Z., and Guo, Y. (2020). Fog-centric authenticated key agreement scheme without trusted parties. *IEEE Systems Journal*.

[Hernández-Nieves et al., 2020] Hernández-Nieves, E., Hernández, G., Gil-González, A.-B., Rodríguez-González, S., and Corchado, J. M. (2020). Fog computing architecture for personalized recommendation of banking products. *Expert Systems with Applications*, 140 :112900.

[Højsgaard and Glenstrup, 2011] Højsgaard, E. and Glenstrup, A. J. (2011). The bpl tool : A tool for experimenting with bigraphical reactive systems. *Bigraphical Languages and their Simulation*, page 85.

[IEEE, 2018] IEEE (2018). Ieee standard for adoption of openfog reference architecture for fog computing. *IEEE Std 1934-2018*, pages 1–176.

[Iorga et al., 2018] Iorga, M., Feldman, L., Barton, R., Martin, M. J., Goren, N. S., Mahmoudi, C., et al. (2018). Fog computing conceptual model. *NIST SP, National Institute of Standards and Technology*.

[ISO/IEC/IEEE, 2011] ISO/IEC/IEEE (2011). Iso/iec/ieee systems and software engineering—architecture description.

[Kahlaoui, 2011] Kahlaoui, A. (2011). *Méthode pour la définition des langages dédiés basée sur le métamodèle ISO/IEC 24744*. PhD thesis, École de technologie supérieure.

[Khebbeb, 2019] Khebbeb, K. (2019). *Formalisation et évaluation de stratégies d'élasticité multi-couches dans le Cloud*. Theses, LIUPPA-Laboratoire Informatique de l'Université de Pau et des Pays de l ....

[Khebbeb et al., 2020] Khebbeb, K., Hameurlain, N., and Belala, F. (2020). A maude-based rewriting approach to model and verify cloud/fog self-adaptation and orchestration. *Journal of Systems Architecture*, 110 :101821.

[Kochovski and Stankovski, 2021] Kochovski, P. and Stankovski, V. (2021). Building applications for smart and safe construction with the decenter fog computing and brokerage platform. *Automation in construction*, 124 :103562.

[Kramer, 2007] Kramer, J. (2007). Is abstraction the key to computing ? *Communications of the ACM*, 50(4) :36–42.

[Krim, 2017] Krim, M. (2017). Comprendre le fog computing en sept questions. `https://www.journaldunet.com/solutions/cloud-computing/1193318-comprendre-le-fog-computing-en-sept-questions/`.

[Krivine et al., 2008] Krivine, J., Milner, R., and Troina, A. (2008). Stochastic bigraphs. *Electronic Notes in Theoretical Computer Science*, 218 :73–96.

[La_Liste_Inconnue, 2022] La_Liste_Inconnue (2022). [youtube] nikola tesla avait prédit l'avenir de 2025 ! https://www.youtube.com/watch?v=VIB61b8D-LQ.

[Lee et al., 2020] Lee, K., Silva, B. N., and Han, K. (2020). Deep learning entrusted to fog nodes (dlefn) based smart agriculture. *Applied Sciences*, 10(4) :1544.

[Li et al., 2022] Li, X., Chen, T., Cheng, Q., and Ma, J. (2022). An efficient and authenticated key establishment scheme based on fog computing for healthcare system. *Frontiers of Computer Science*, 16(4) :1–12.

[Marir et al., 2018] Marir, S., Belala, F., and Hameurlain, N. (2018). A formal model for interaction specification and analysis in iot applications. In *International Conference on Model and Data Engineering*, pages 371–384. Springer.

[Marir et al., 2020] Marir, S., Belala, F., and Hameurlain, N. (2020). Formal modeling iot systems on the basis of biagents* and maude. In *2020 International Conference on Advanced Aspects of Software Engineering (ICAASE)*, pages 1–7. IEEE.

[Marir et al., 2022] Marir, S., Belala, F., and Hameurlain, N. (2022). A strategy-based formal approach for fog systems analysis. *Future Internet*, 14(2) :52.

[Marir et al., 2017] Marir, S., Kitouni, R., Benzadri, Z., and Belala, F. (2017). Biagent-based model for iot applications. In *International Conference on Service-Oriented Computing*, pages 111–123. Springer.

[Martí-Oliet et al., 2009] Martí-Oliet, N., Meseguer, J., and Verdejo, A. (2009). A rewriting semantics for maude strategies. *Electronic Notes in Theoretical Computer Science*, 238(3) :227–247.

[Mcafee and Brynjolfsson, 2012] Mcafee, A. and Brynjolfsson, E. (2012). Hbr. org spotlight on big data big data : The management revolution. *Harv. Bus. Rev*, pages 3–9.

[McKendrick,] McKendrick, J. Fog computing : a new iot architecture ? *RTInsights.com*.

[Milner, 1980] Milner, R. (1980). A calculus of communicating systems. *LNCS*, 92.

[Milner, 1993] Milner, R. (1993). Action calculi, or syntactic action structures. In *International Symposium on Mathematical Foundations of Computer Science*, pages 105–121. Springer.

[Milner, 2008] Milner, R. (2008). Bigraphs and their algebra. *Electronic Notes in Theoretical Computer Science*, 209(0) :5–19.

[Milner, 2009] Milner, R. (2009). *The space and motion of communicating agents*. Cambridge University Press.

[Milner et al., 1992] Milner, R., Parrow, J., and Walker, D. (1992). A calculus of mobile processes, i. *Information and computation*, 100(1) :1–40.

[MOF, 2003] MOF, O. (2003). Omg meta object facility (mof) 2.0 core specification version 2.0 : Final adopted specification.

[Mouradian et al., 2017] Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J., and Polakos, P. A. (2017). A comprehensive survey on fog computing : State-of-the-art and research challenges. *IEEE communications surveys & tutorials*, 20(1) :416–464.

[Murtaza et al., 2020] Murtaza, F., Akhunzada, A., ul Islam, S., Boudjadar, J., and Buyya, R. (2020). Qos-aware service provisioning in fog computing. *Journal of Network and Computer Applications*, 165 :102674.

[Mutlag et al., 2021] Mutlag, A. A., Ghani, M. K. A., Mohammed, M. A., Lakhan, A., Mohd, O., Abdulkareem, K. H., and Garcia-Zapirain, B. (2021). Multi-agent systems in fog–cloud computing for critical healthcare task management model (chtm) used for ecg monitoring. *Sensors*, 21(20) :6923.

[Noble et al., 1997] Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., and Walker, K. R. (1997). Agile application-aware adaptation for mobility. *ACM SIGOPS Operating Systems Review*, 31(5) :276–287.

[Norman and Kuras, 2006] Norman, D. O. and Kuras, M. L. (2006). Engineering complex systems. In *Complex Engineered Systems*, pages 206–245. Springer.

[Noura et al., 2017] Noura, M., Atiquzzaman, M., and Gaedke, M. (2017). Interoperability in internet of things infrastructure : Classification, challenges, and future work. In *International Conference on Internet of Things as a Service*, pages 11–18. Springer.

[Noura et al., 2019] Noura, M., Atiquzzaman, M., and Gaedke, M. (2019). Interoperability in internet of things : Taxonomies and open challenges. *Mobile Networks and Applications*, 24(3) :796–809.

[Patel and Patel, 2016] Patel, K. and Patel, S. (2016). *Internet of Things-IOT : Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges*. IJESC.

[Pereira et al., 2012] Pereira, E., Kirsch, C., and Sengupta, R. (2012). Biagents–a bigraphical agent model for structure-aware computation. *Cyber-Physical Cloud Computing Working Papers, CPCC Berkeley*, pages 1–13.

[Perrone et al., 2011] Perrone, G., Debois, S., and Hildebrandt, T. (2011). Bigraphical refinement. *arXiv preprint arXiv :1106.4091*.

[Perrone et al., 2012] Perrone, G., Debois, S., and Hildebrandt, T. T. (2012). A model checker for bigraphs. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 1320–1325. ACM.

[Petrovic and Tosic, 2020] Petrovic, N. and Tosic, M. (2020). Smada-fog : Semantic model driven approach to deployment and adaptivity in fog computing. *Simulation Modelling Practice and Theory*, 101 :102033.

[Ravandi and Papapanagiotou, 2017] Ravandi, B. and Papapanagiotou, I. (2017). A self-learning scheduling in cloud software defined block storage. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 415–422. IEEE.

[Rivera, 2010] Rivera, J. E. (2010). *On the semantics of real-time domain specific modeling languages*. PhD thesis, University of Malaga.

[Roig et al., 2021a] Roig, P. J., Alcaraz, S., Gilly, K., Bernad, C., and Juiz, C. (2021a). Modeling of a generic edge computing application design. *Sensors*, 21(21) :7276.

[Roig et al., 2021b] Roig, P. J., Alcaraz, S., Gilly, K., and Juiz, C. (2021b). Algebraic modelling of a generic fog scenario for moving iot devices. In Thampi, S. M., Lloret Mauri, J., Fernando, X., Boppana, R., Geetha, S., and Sikora, A., editors, *Applied Soft Computing and Communication Networks*, pages 1–16, Singapore. Springer Singapore.

[Roig et al., 2021c] Roig, P. J., Alcaraz, S., Gilly, K., and Juiz, C. (2021c). Modelling a plain n-hypercube topology for migration in fog computing. In Thampi, S. M., Gelenbe, E., Atiquzzaman, M., Chaudhary, V., and Li, K.-C., editors, *Advances in Computing and Network Communications*, pages 595–608, Singapore. Springer Singapore.

[Sahli, 2017] Sahli, H. (2017). *A Formal Framework for Modelling and Verifying Cloud-based Elastic Systems*. Theses, Université Constantine 2 - Abdelhamid Mehri. `https://hal.archives-ouvertes.fr/tel-01484662`.

[Sahli et al., 2019] Sahli, H., Ledoux, T., and Rutten, É. (2019). Modeling self-adaptive fog systems using bigraphs. In *International Conference on Software Engineering and Formal Methods*, pages 252–268. Springer.

[Satyanarayanan, 2001] Satyanarayanan, M. (2001). Pervasive computing : Vision and challenges. *IEEE Personal communications*, 8(4) :10–17.

[Satyanarayanan et al., 2009] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4) :14–23.

[Sevegnani and Calder, 2015] Sevegnani, M. and Calder, M. (2015). Bigraphs with sharing. *Theoretical Computer Science*, 577 :43–73.

[Sevegnani and Calder, 2016] Sevegnani, M. and Calder, M. (2016). BigraphER : rewriting and analysis engine for bigraphs. In *International Conference on Computer Aided Verification*, pages 494–501. Springer.

[Silva and Andrade, 2021] Silva, R. A. d. S. and Andrade, R. M. (2021). D-creea : Dsml for creating educational analog card games. In *2021 20th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 49–58. IEEE.

[Smaali, 2017] Smaali, S. (2017). *DySAL : Un langage formel spécifique à la modélisation des architectures logicielles dynamiques*. PhD thesis, Université de Constantine 2 Abdelhamid Mehri.

[STAC, 2010] STAC (2010). *Capacité des aérogares passagers*. dgac. [Guide technique].

[Tolvanen, 2006] Tolvanen, J.-P. (2006). Domain-specific modeling : How to start defining your own language. *DevX. com*, 81.

[Warmer and Kleppe, 2003] Warmer, J. B. and Kleppe, A. G. (2003). *The object constraint language : getting your models ready for MDA*. Addison-Wesley Professional.

[Wu et al., 2011] Wu, Y., Hernandez, F., Clarke, P. J., and France, R. (2011). A dsml for coordinating user-centric communication services. In *2011 IEEE 35th Annual Computer Software and Applications Conference*, pages 93–102. IEEE.

[Xiao and Krunz, 2021] Xiao, Y. and Krunz, M. (2021). Adaptivefog : A modelling and optimization framework for fog computing in intelligent transportation systems. *IEEE Transactions on Mobile Computing*.

[Yi et al., 2015] Yi, S., Hao, Z., Qin, Z., and Li, Q. (2015). Fog computing : Platform and applications. In *2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb)*, pages 73–78. IEEE.

[Yousefpour et al., 2019] Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., and Jue, J. P. (2019). All one needs to know about fog computing and related edge computing paradigms : A complete survey. *Journal of Systems Architecture*, 98 :289–330.

[Zahra et al., 2017] Zahra, S., Alam, M., Javaid, Q., Wahid, A., Javaid, N., Malik, S. U. R., and Khan, M. K. (2017). Fog computing over iot : A secure deployment and formal verification. *IEEE Access*, 5 :27132–27144.

[Zhang et al., 2015] Zhang, B., Mor, N., Kolb, J., Chan, D. S., Lutz, K., Allman, E., Wawrzynek, J., Lee, E., and Kubiatowicz, J. (2015). The cloud is not enough : Saving {IoT} from the cloud. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*.